Inteligencia Artificial

UNIVERSITAT DE BARCELONA

Resolución de problemas (III.3)

Búsqueda en juegos

Búsqueda en juegos

UNIVERSITAT DE BARCELONA

- Tipos de juegos
- Juegos de suma cero de dos jugadores.
  - Minimax
  - Poda alfa-beta
- Juegos contra la naturaleza.
  - Repaso de probabilidades
  - Expectimax
- Juegos de suma no cero.

Estado del arte

- Damas: Chinook gana el campeonato del mundo en 1994. Actualmente resuelto.
- Ajedrez: Deep Blue vence a Gary Kasparov en 1997.
- Othello (reversi): Los campeones humanos se niegan a jugar contra los ordenadores, que son mucho mejores.
- Go: Los campeones humanos empiezan a verse amenazados por los ordenadores, que todavía pierden.

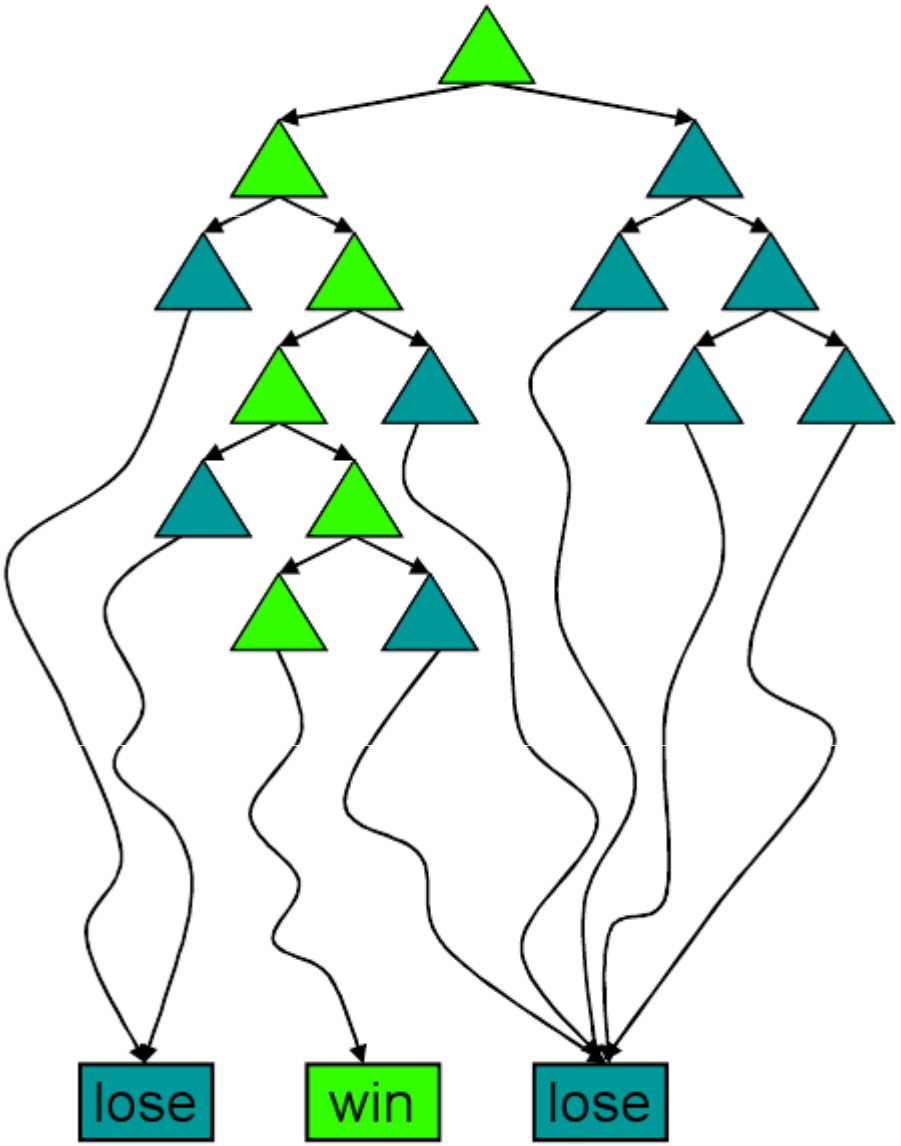
Tipos de juegos

- Determinista vs. Estocástico.
- Uno, dos, tres, … jugadores.
- Información completa/incompleta.

- El objetivo de los algoritmos es calcular una política que recomiende el movimiento a realizar en cada estado.

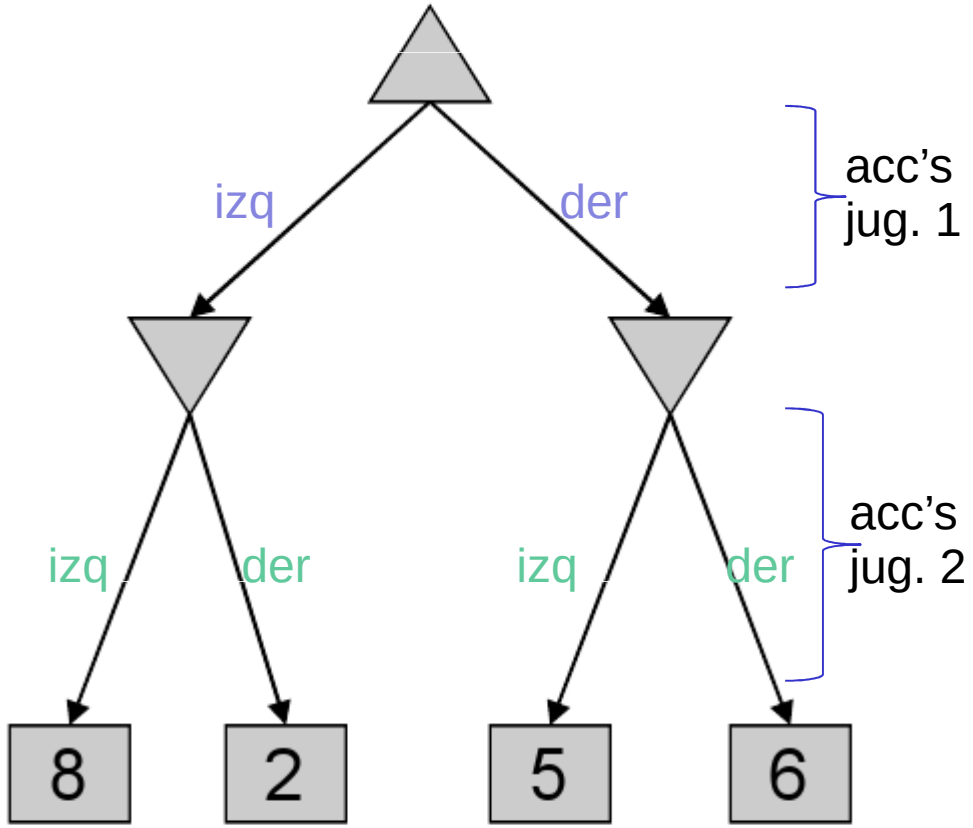
Deterministas de un jugador

- Información perfecta:
  - Conocemos las reglas, las acciones y el objetivo
  - 8-puzzle, Cubo Rubik
- … son simplemente búsqueda.
- Reinterpretación:
  - Cada nodo almacena un valor: el mejor que puede conseguir por esa rama.
- Tras la búsqueda se pueden escoger los movimientos que llevan al mejor nodo.



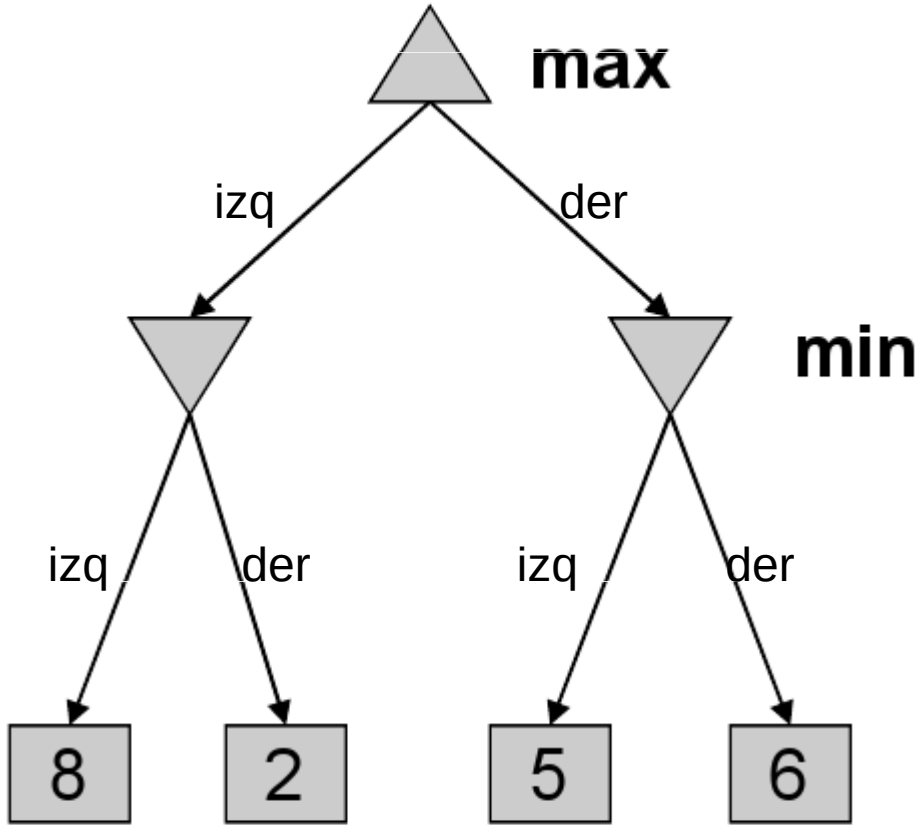
Deterministas de dos jugadores

- Ej: Tres en raya, ajedrez, damas.
- Juegos de suma cero:
  - Lo que un jugador gana lo pierde el otro.
- ¿Qué estrategia seguir ?

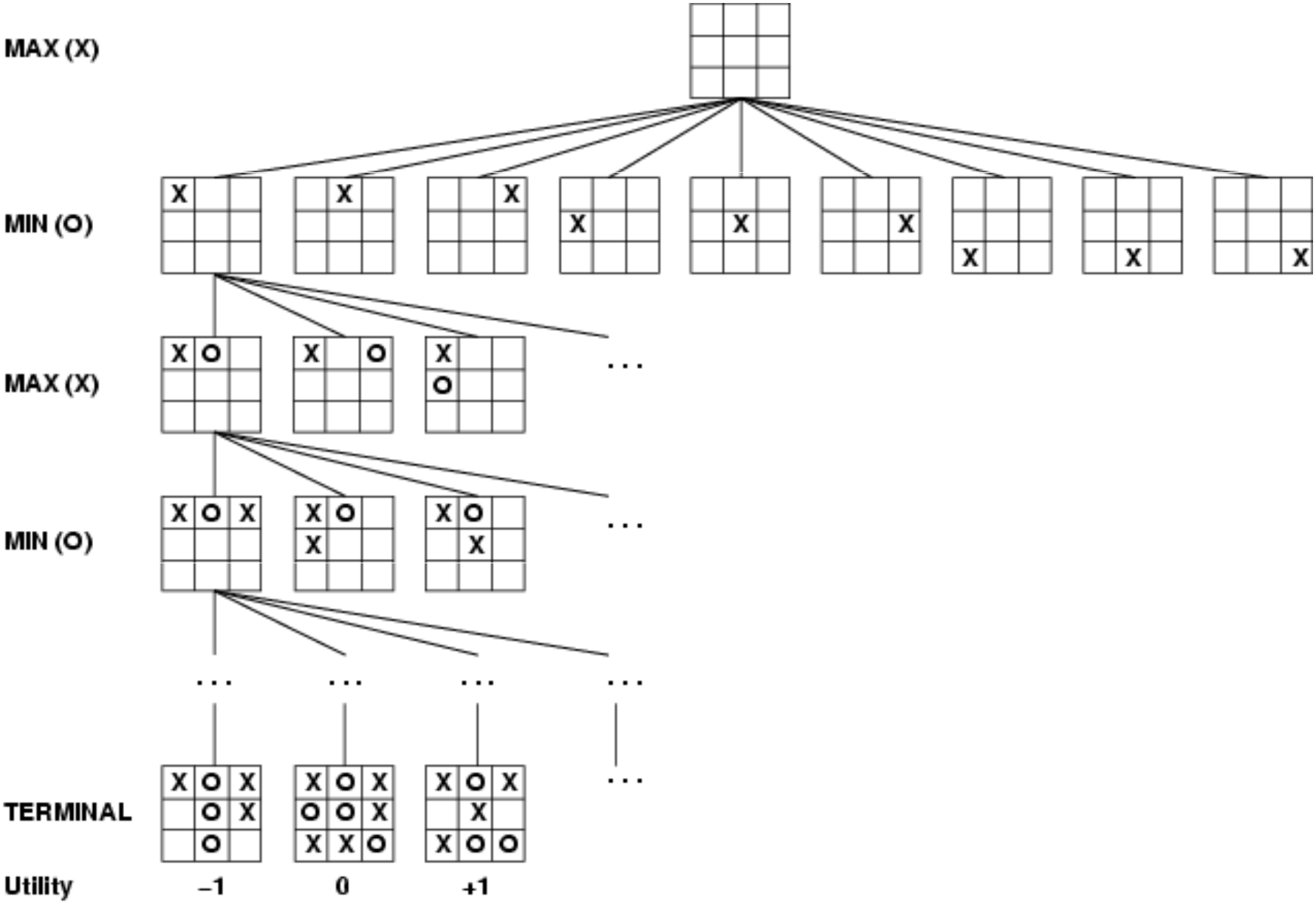


Deterministas de dos jugadores

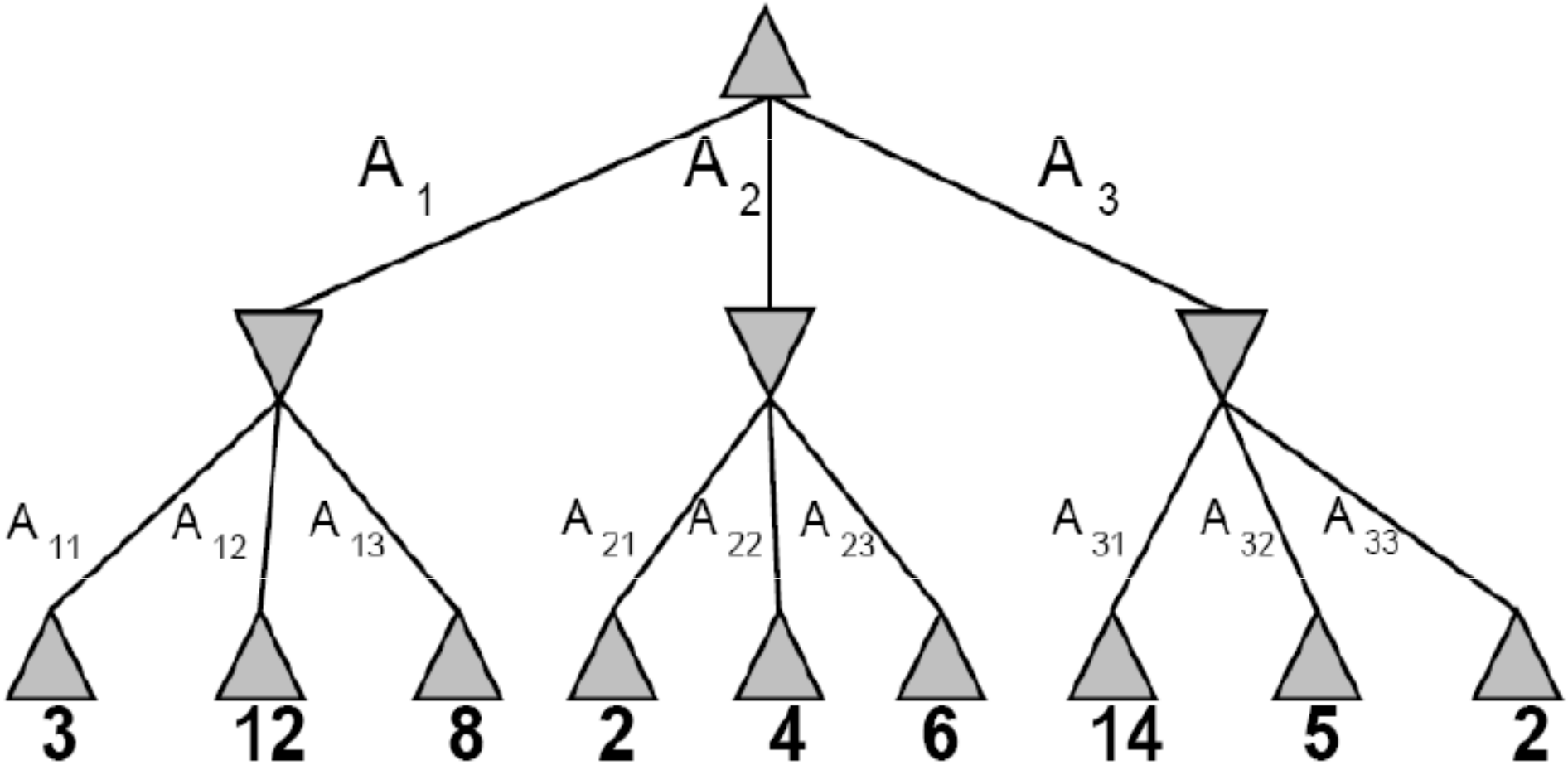
- Ej: Tres en raya, ajedrez, damas.
- Juegos de suma cero:
  - Lo que un jugador gana lo pierde el otro.
- Minimax:
  - Escoger el movimiento con mayor valor minimax.
  - ¿Cuál es la mejor jugada posible suponiendo que nuestro rival es perfecto?



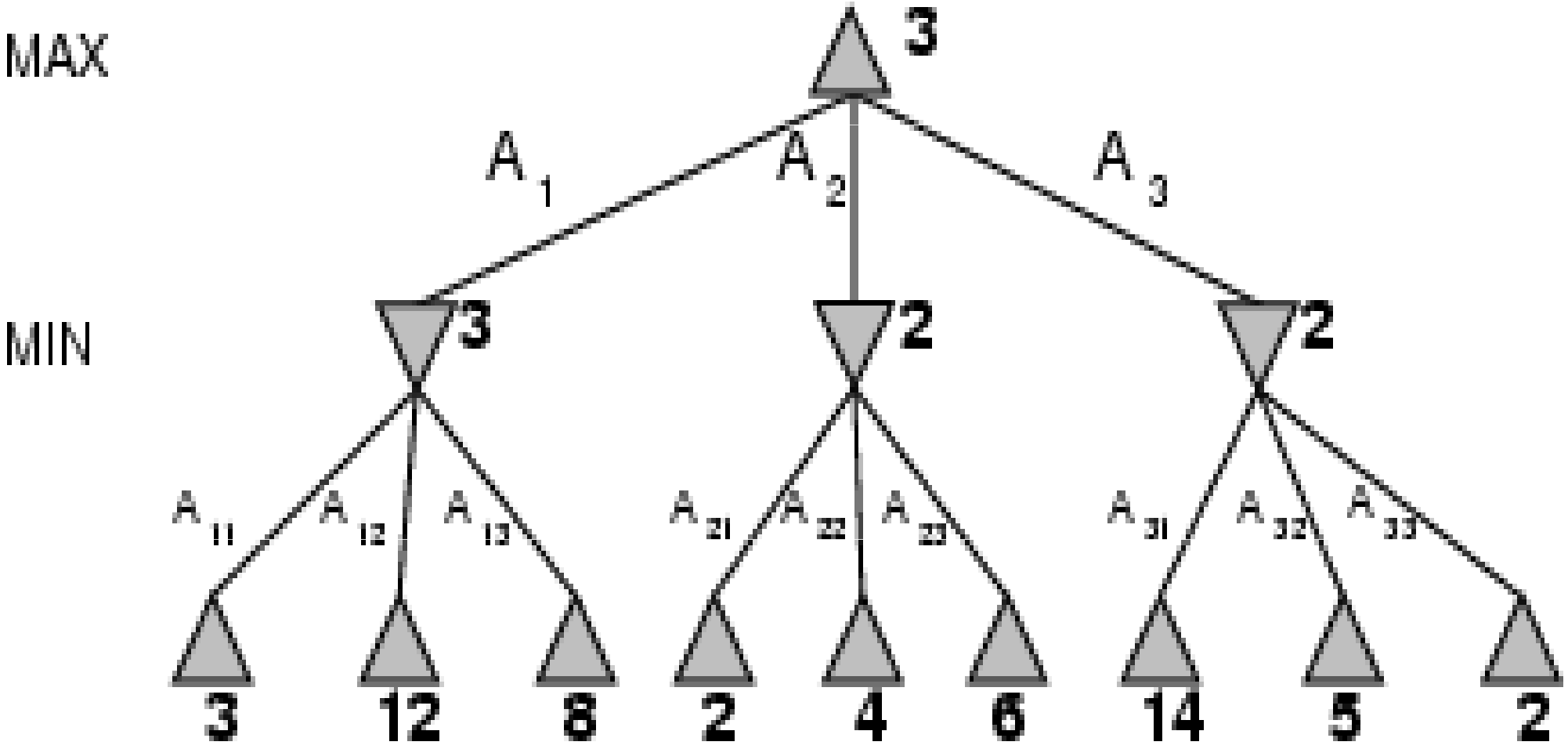
Árbol de juego (2 jugadores, determinista, por turnos)



Ejemplo minimax



Ejemplo minimax



Algoritmo minimax

UNIVERSITAT DE BARCELONA

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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
```

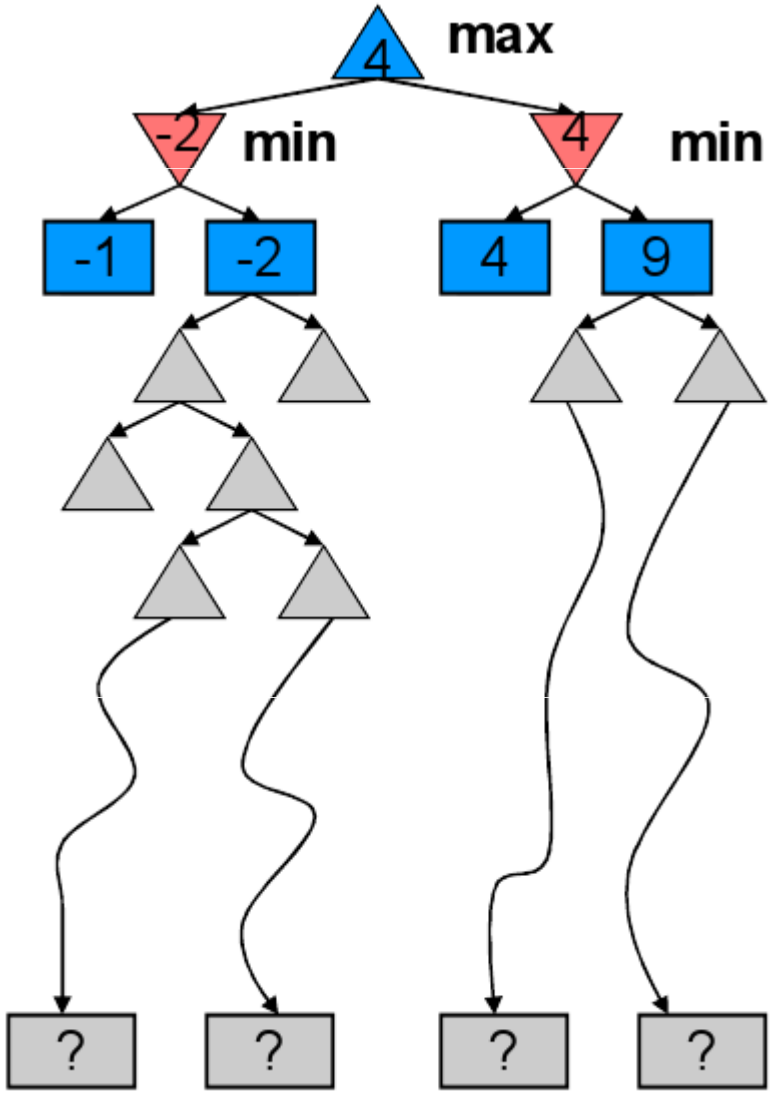
Propiedades minimax

- Óptimo contra un enemigo perfecto. ¿Y si no lo es?
- Complejidad temporal: $O(b^m)$
- Complejidad espacial: $O(b \cdot m)$
- Completo si el árbol es finito.

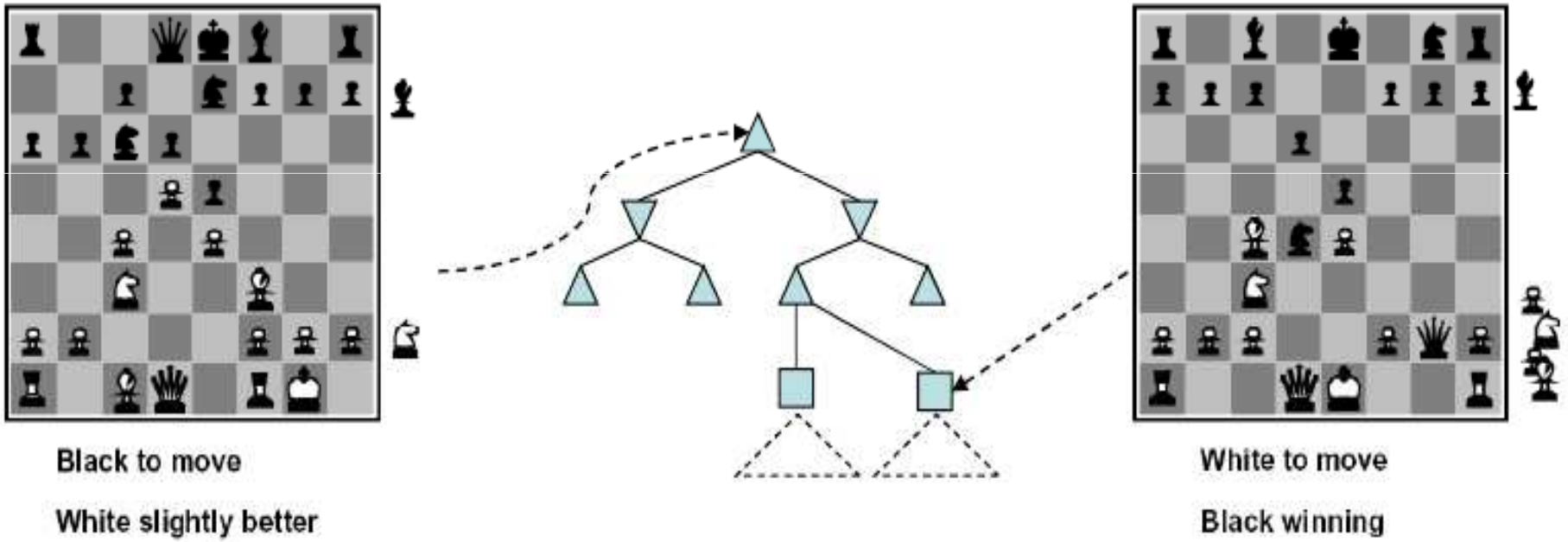
- En ajedrez b~35, m~100 → La solución exacta es totalmente intratable.

Reduciendo el gasto computacional

- Buscar hasta una profundidad determinada.
- Reemplazar las utilidades de los nodos terminales por una evaluación heurística.
- La garantía de optimalidad desaparece.
- La profundidad es clave.



Funciones de evaluación

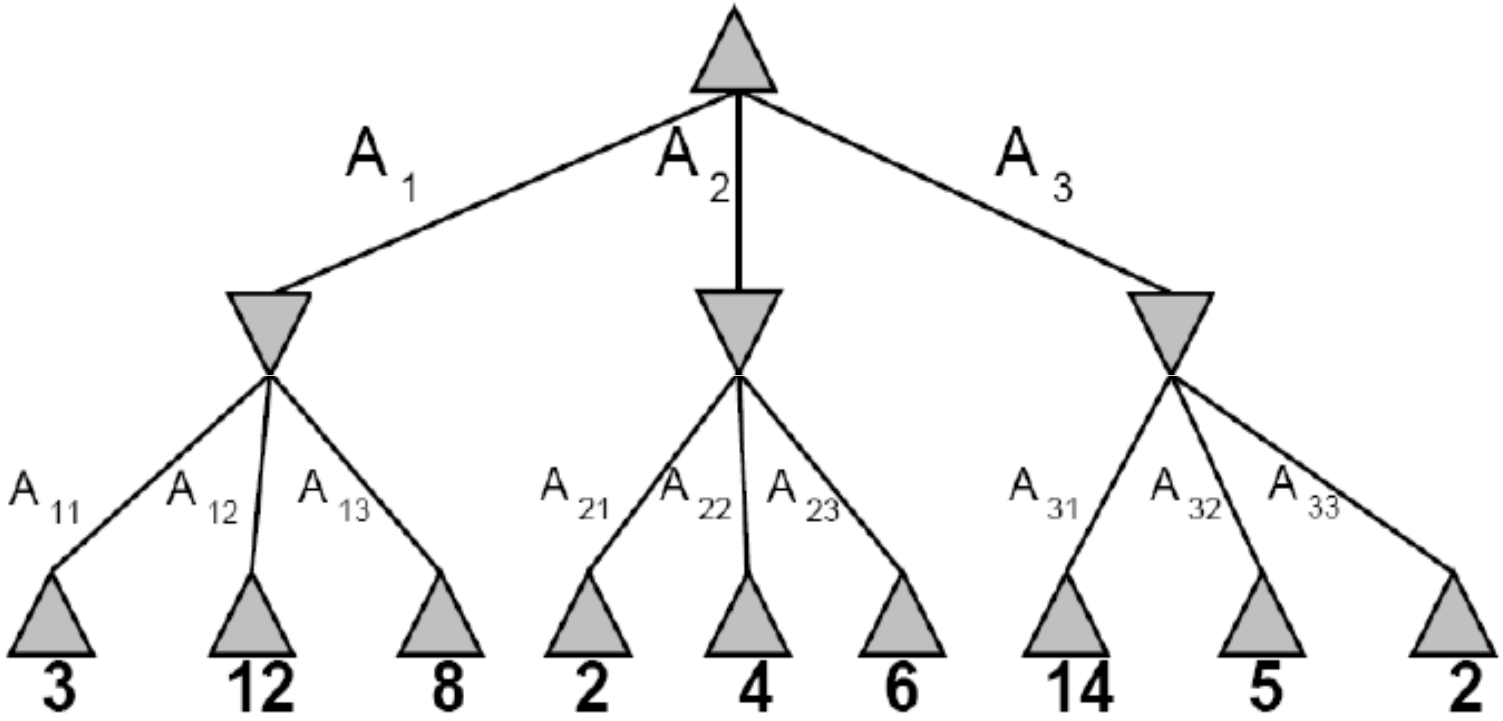


- Debe parecerse lo más posible a la utilidad de esa posición.
- Habitualmente:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

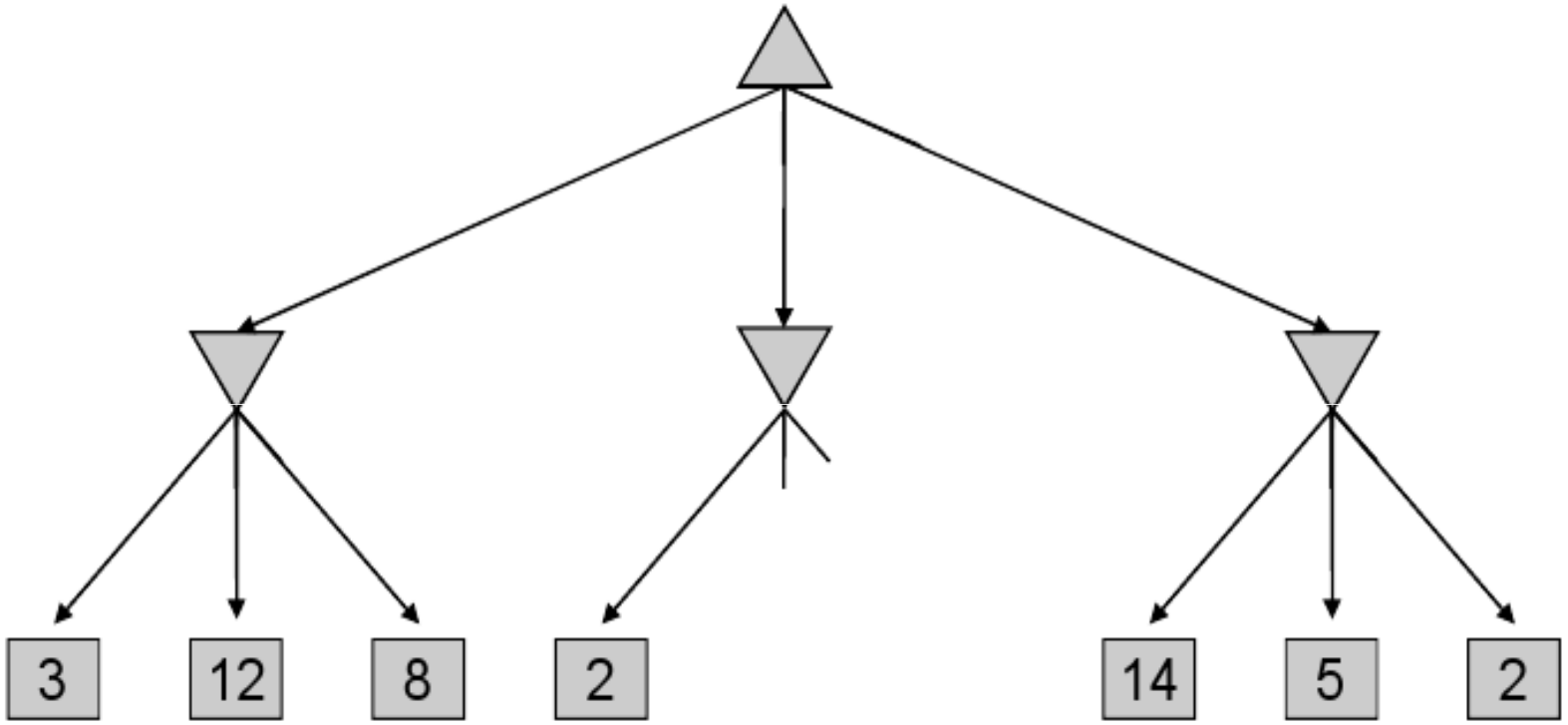
Podando minimax (ej 1)

- ¿Podemos ahorrarnos expandir o evaluar algún nodo?



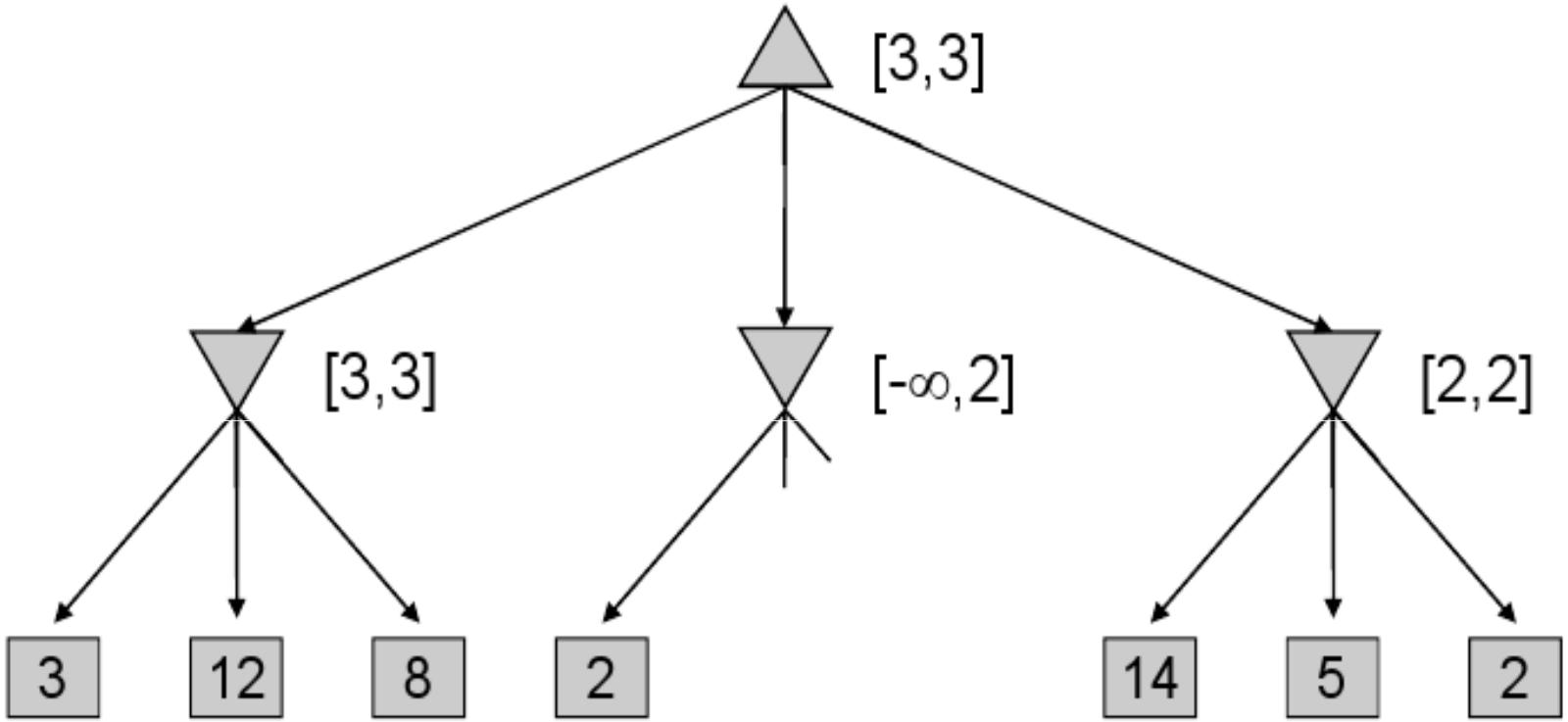
Podando minimax (ej 1)

- ¿Podemos ahorrarnos expandir o evaluar algún nodo?



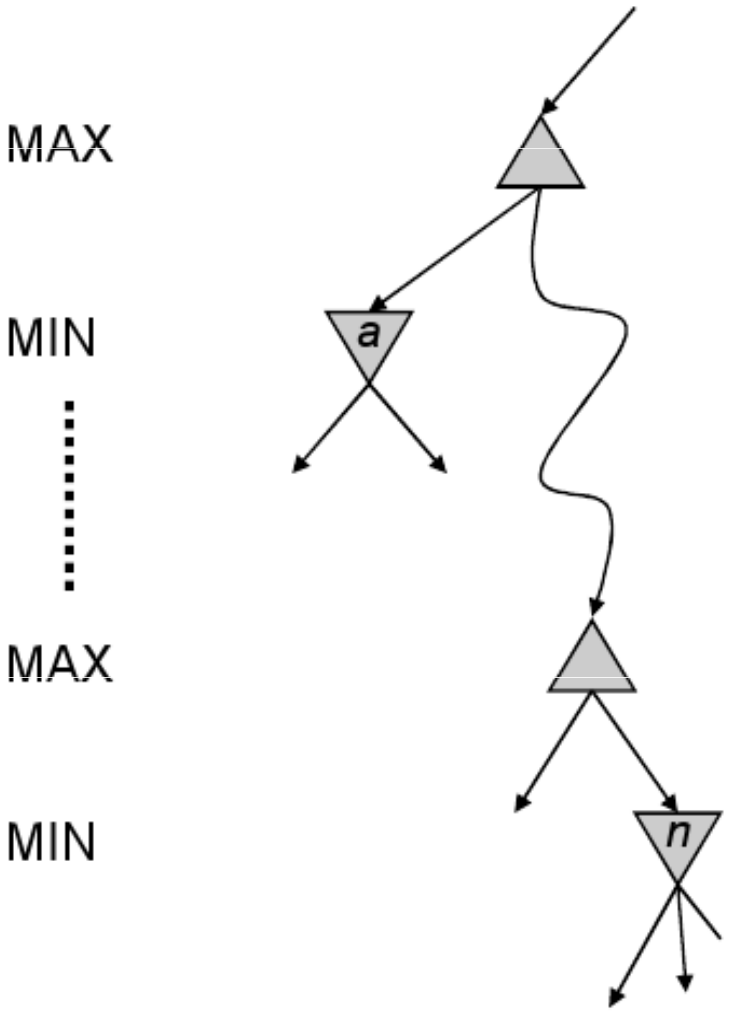
Podando minimax (ej 1)

- ¿Podemos ahorrarnos expandir o evaluar algún nodo?



Poda alfa-beta

- Calculando el mínimo en n (recorriendo los hijos de n).
- La estimación del valor de n va bajando.
- Sea a el valor máximo que MAX puede obtener en cualquier elección a lo largo del camino actual.
- Si n < a podemos dejar de expandir los hijos de n, ya que tenemos una alternativa mejor.
- Podemos definir de la misma forma b para MIN.



Poda alfa-beta

UNIVERSITAT DE BARCELONA

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

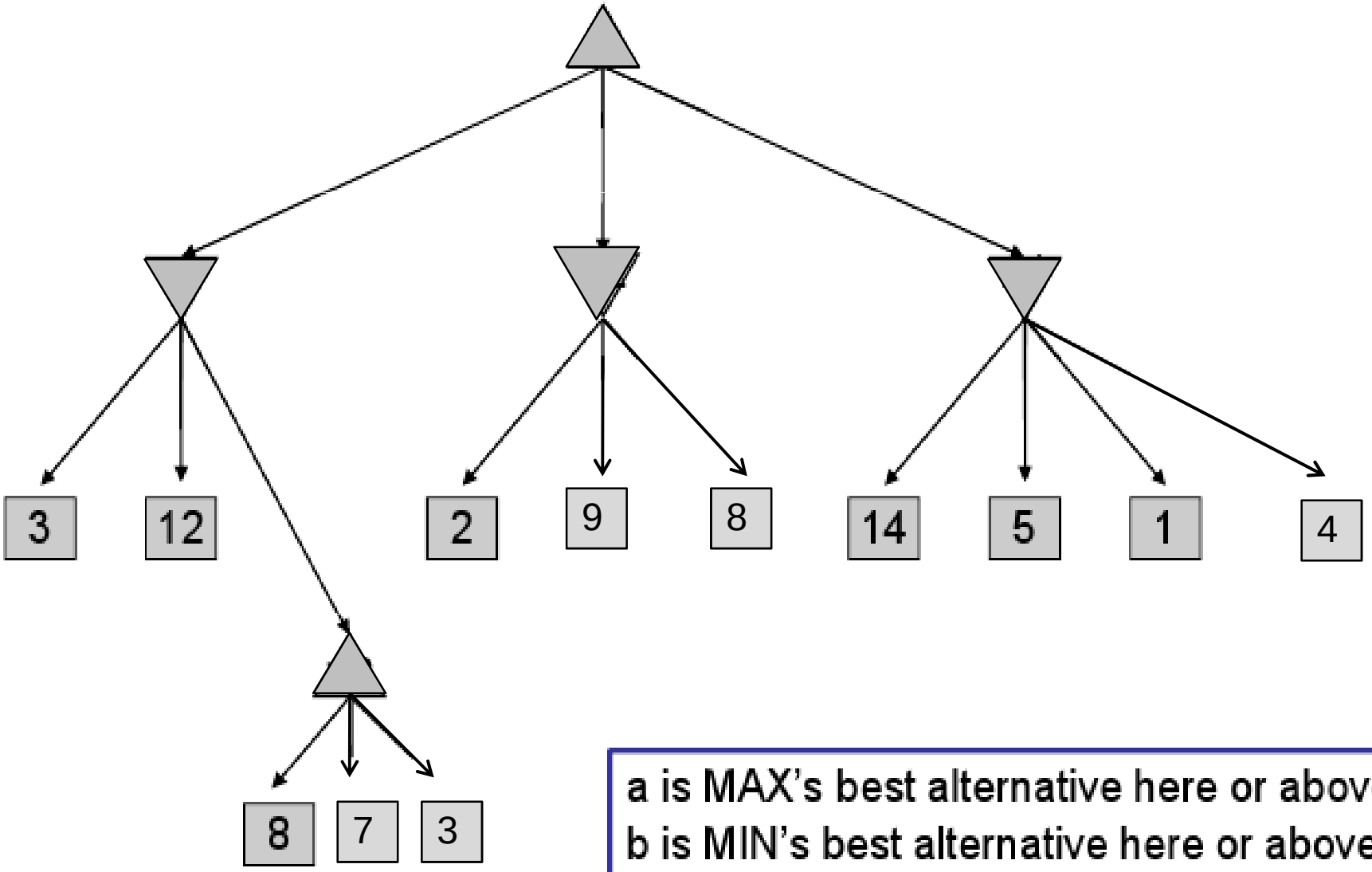
---

function MAX-VALUE($state, \alpha, \beta$) **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*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

MIN-VALUE (state, α, β):
if terminal(state) then return utility(state)
v =+∞
for a,s in Successors(state) do
    v← min(v, MAX-VALUE(s, α, β))
    if v ≤α then return v
    β=min(β, v)
return v

Ejemplo poda alfa-beta (ej 2)



a is MAX's best alternative here or above
b is MIN's best alternative here or above