DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Juegos

© Fernando Berzal, berzal@acm.org

Tipos de juegos

	Juegos deterministas	Juegos de azar
Con información perfecta	Ajedrez, damas, Go, Othello	Backgammon, Monopoly
Con información imperfecta	barquitos	Bridge, poker, scrabble

Juegos

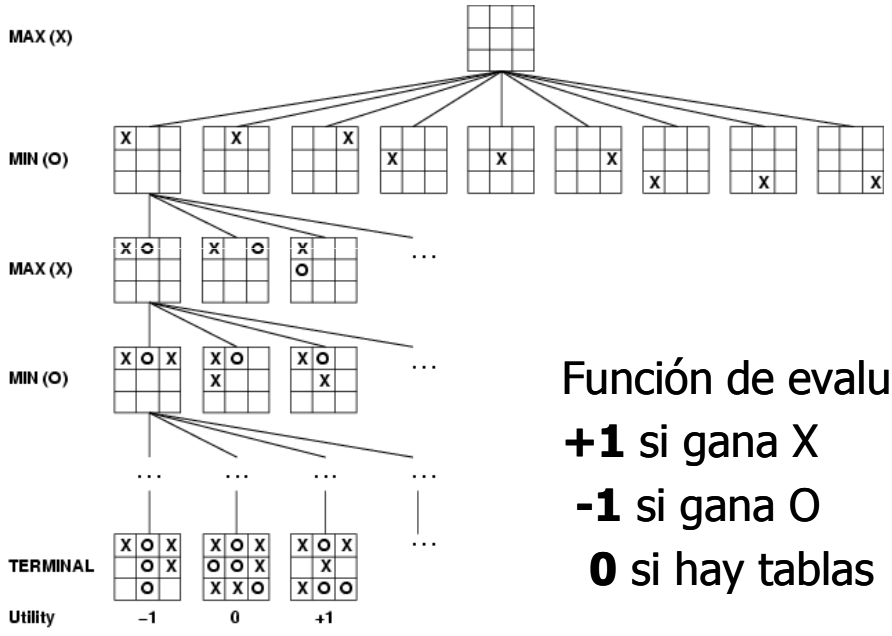
Juego perfecto

- Dos jugadores
- Movimientos intercalados
- Suma cero (la ganancia de uno es la pérdida del otro).
- Información perfecta (ambos jugadores tienen acceso a toda la información sobre el estado del juego: no se ocultan información el uno al otro).
- No interviene el azar (p.ej. dados).

Ejemplos:

Nim, Grundy, 3 en raya, conecta-4, damas, ajedrez…

Juegos



Función de evaluación:

+1 si gana X

-1 si gana O

0 si hay tablas

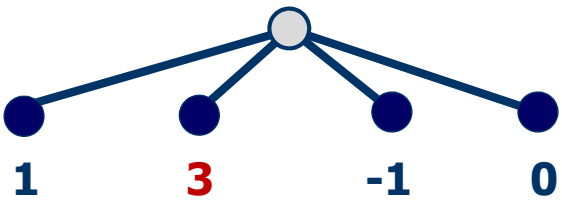
Juegos

Complejidad de algunos juegos con adversario

Juego	Estados
3 en raya	9! = 362280
Conecta-4	10^{13}
Damas	10^{18}
Ajedrez	10^{47}
Go	10^{170}

Minimax

Considerando sólo nuestros posibles movimientos…
… elegiríamos el movimiento más prometedor (MAX):

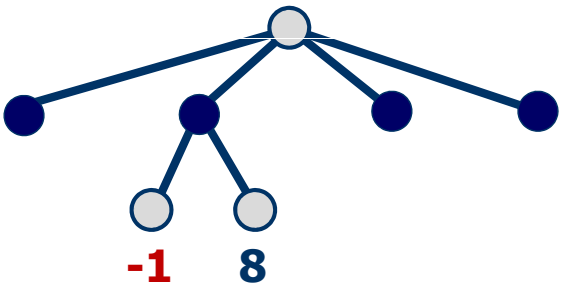


Nuestro turno

Turno de nuestro oponente

Minimax

Nuestro oponente, a continuación…
… elegiría lo que más le conviene a él (MIN):

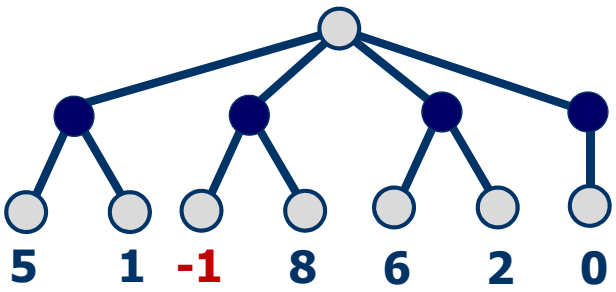


Nuestro turno

Turno de nuestro oponente

Minimax

Si antes de realizar el movimiento,
hubiésemos explorado un nivel más de profundidad, nos habría dado cuenta de que el movimiento no era bueno:

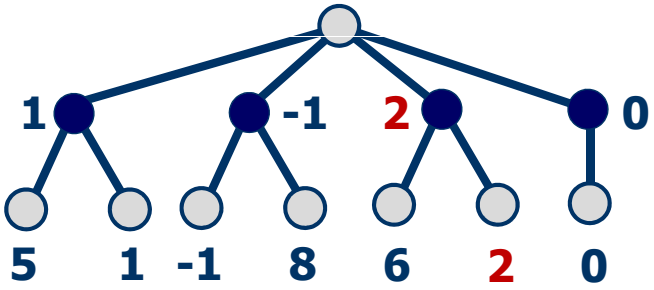


Nuestro turno

Turno de nuestro oponente

Minimax

Si suponemos que nuestro oponente es racional, deberíamos elegir otro movimiento inicial:



Nuestro turno

Turno de nuestro oponente

Minimax

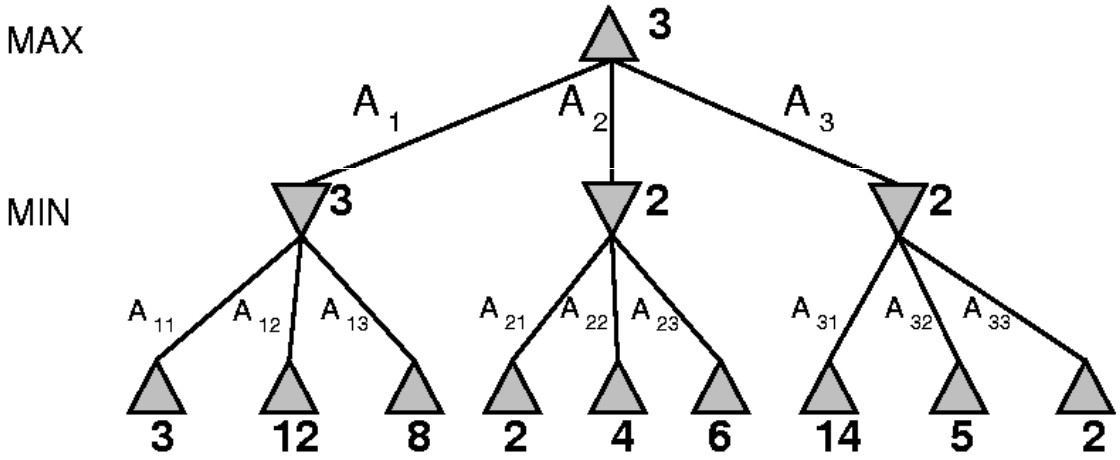
Estrategia perfecta para juegos deterministas.

IDEA: Elegir el movimiento que nos lleva a la posición que nos asegura una recompensa máxima en el peor caso (valor minimax).

- MAX: Cuando movemos nosotros, elegimos el nodo de máximo valor.
- MIN: Cuando mueve nuestro oponente, elige el nodo de menor valor (para nosotros).

Minimax

Árbol con 2 niveles (2-ply):



Minimax

function MINIMAX-DECISION(*state*) **returns** *an action*

$v \leftarrow$ 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*)
$v \leftarrow -\infty$
for a, s in SUCCESSORS(*state*) **do**
$v \leftarrow$ MAX(v, MIN-VALUE(s))
return v

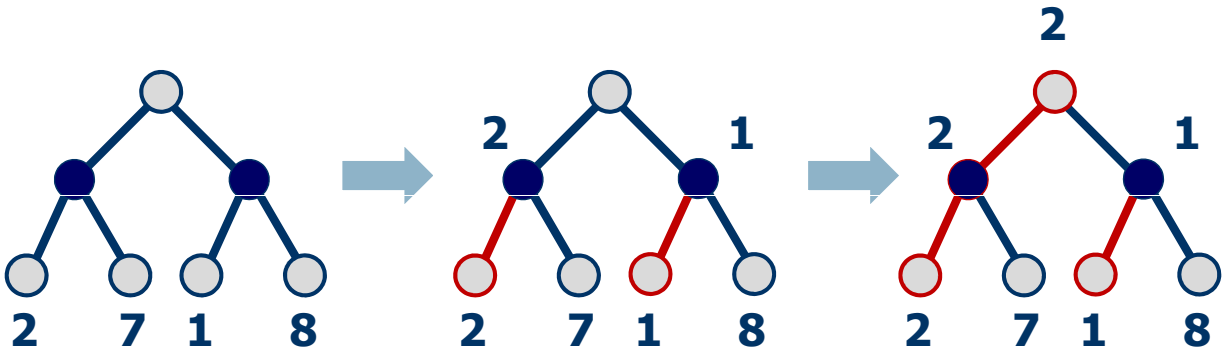
---

function MIN-VALUE(*state*) **returns** *a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
$v \leftarrow \infty$
for a, s in SUCCESSORS(*state*) **do**
$v \leftarrow$ MIN(v, MAX-VALUE(s))
return v

Minimax

Búsqueda minimax (primero en profundidad):



MAX
MIN

Minimax

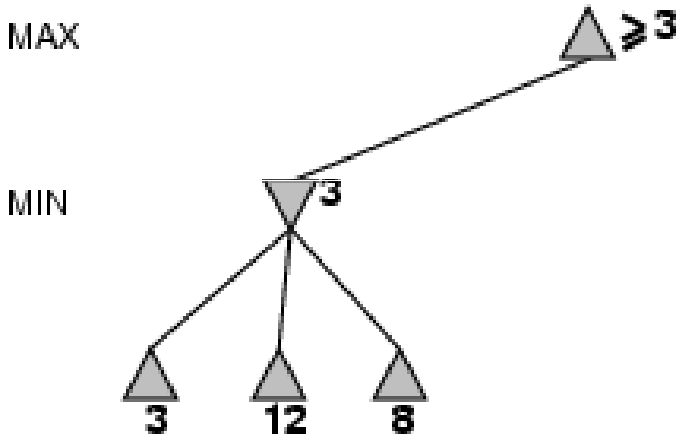
Complejidad

b = Factor de ramificación del árbol
d = Profundidad del árbol de juego

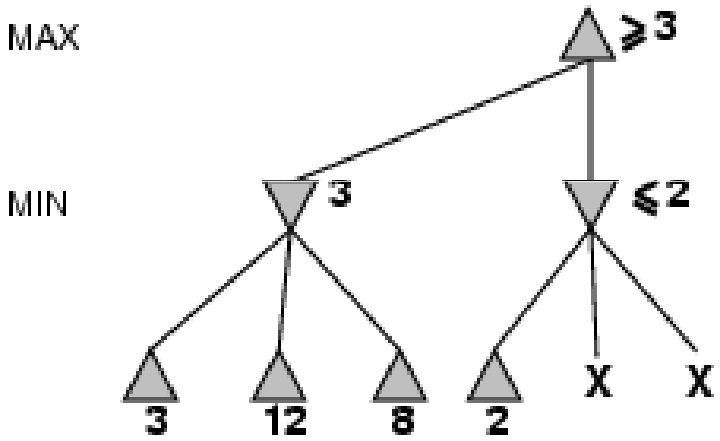
- Tiempo: **O(b^d)**.
- Espacio: **O(bd)**.

Ejemplo
En el ajedrez, b≈35 y d≈100, por lo que **no** podemos explorar el árbol completo del juego.

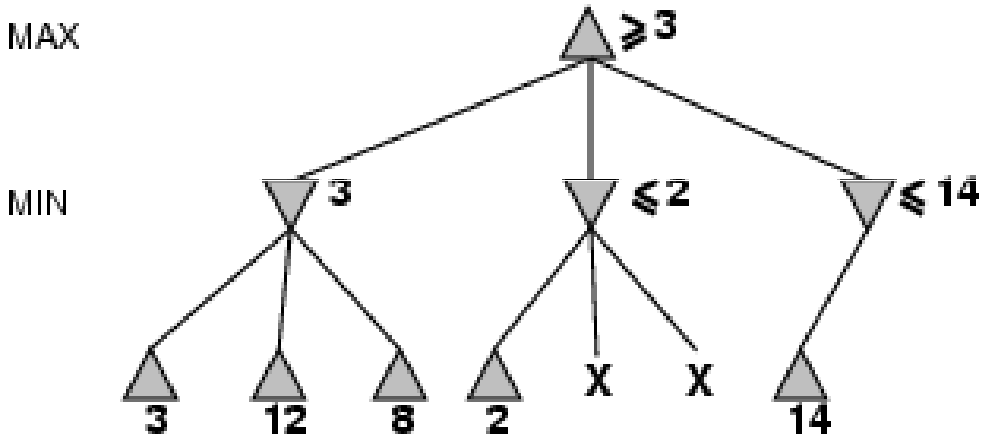
Poda α-β



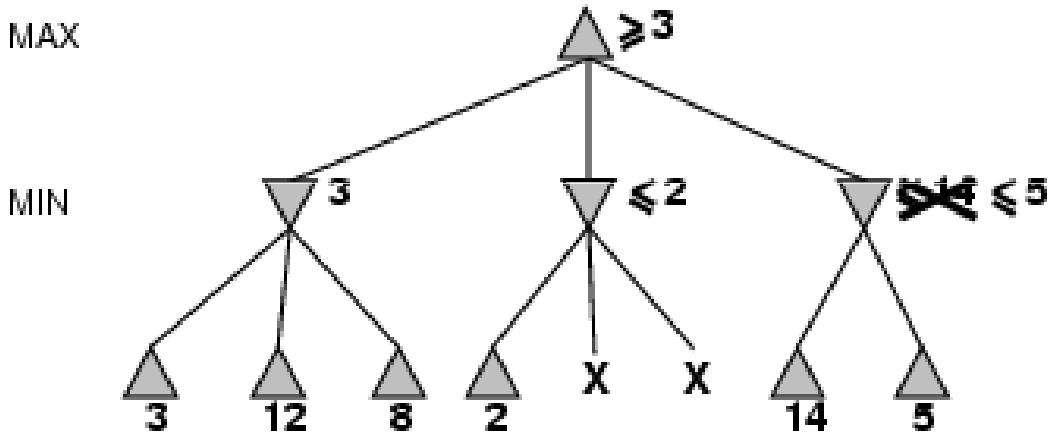
Poda α-β



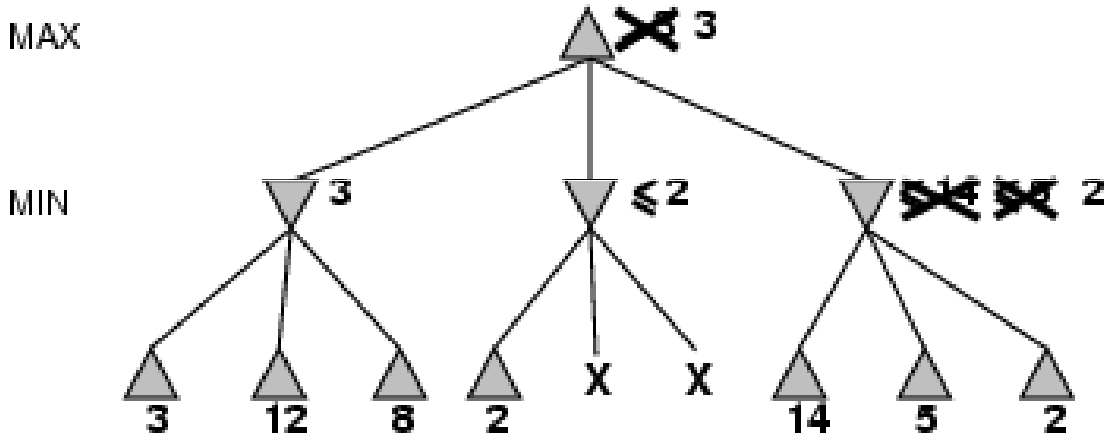
Poda α-β



Poda α-β



Poda α-β



Poda α-β

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

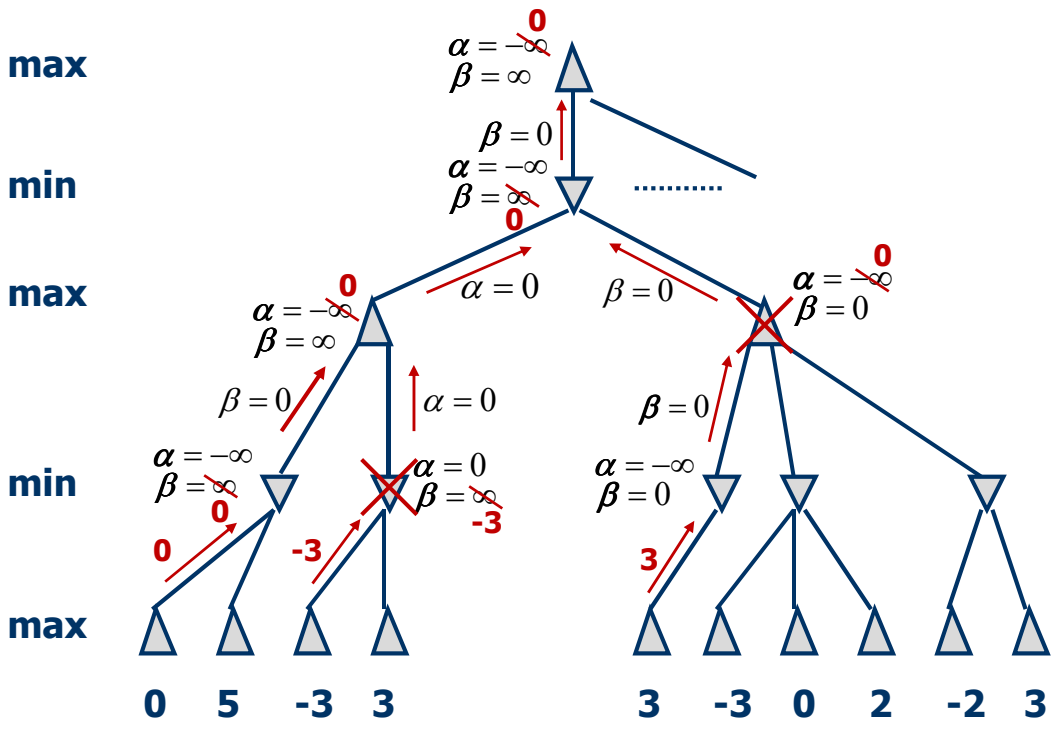
---

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

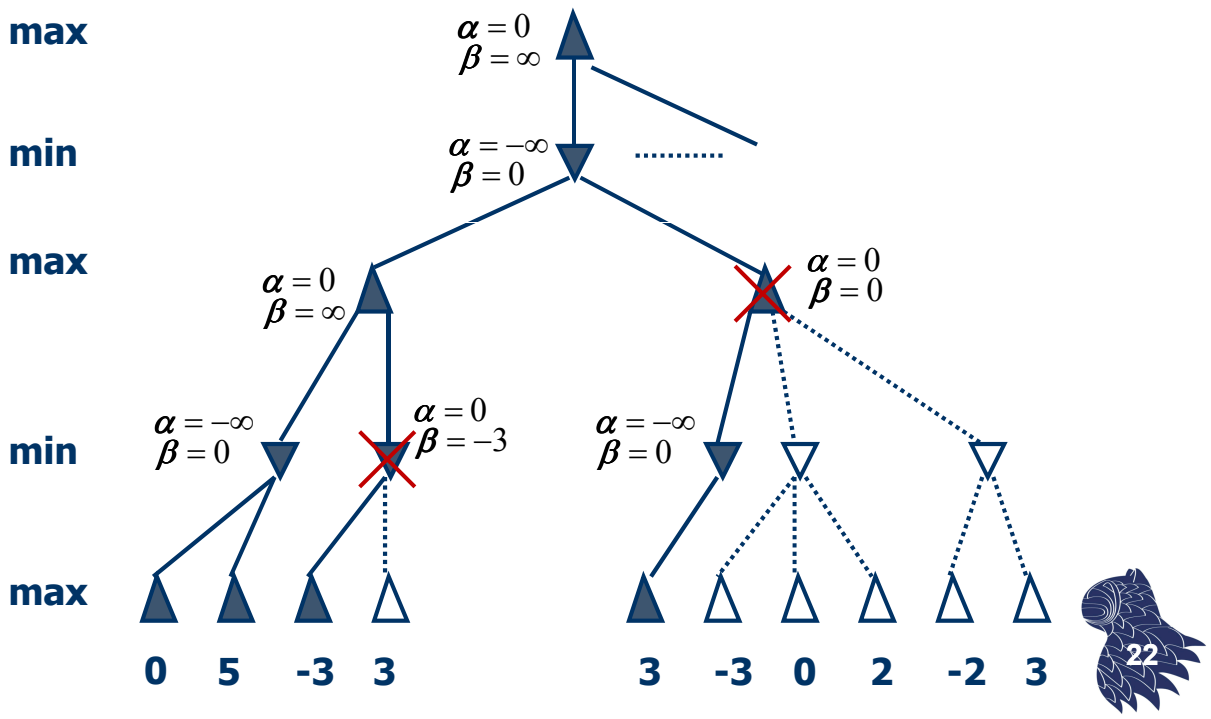
Poda α-β

```
function MIN-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 ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```


Poda α-β



Poda α-β



Poda α-β

- La poda α-β no afecta al resultado del juego.
- Cuanto mejor ordenemos los movimientos, más efectiva será la poda.
- Con una ordenación "perfecta", la complejidad del algoritmo es **$O(b^{d/2})$**.

  En otras palabras, con el mismo esfuerzo podremos explorar un árbol del doble de profundidad.

Poda α-β

¿Por qué se llama poda α-β?

- α es el valor de la mejor opción encontrada para el jugador MAX:

  MAX evitará cualquier movimiento que tenga un valor v peor que α (poda si v<α).

- β es el valor de la mejor opción encontrada para MIN (mínimo encontrado hasta ahora):

  MIN evitará cualquier movimiento que tenga, para él, un valor v peor que β (poda si v> β)

En la práctica…

Si disponemos de 100 segundos por movimiento y podemos explorar 10^4 nodos por segundo, sólo podremos analizar 10^6 nodos por movimiento:

Solución habitual: Exploración del árbol

- Cota de profundidad
- IDS [Iterative Deepening Search]:
  Se realiza una búsqueda en profundidad con una cota de profundidad creciente, hasta que se agota el tiempo.

En la práctica…

Si disponemos de 100 segundos por movimiento y podemos explorar 10^4 nodos por segundo, sólo podremos analizar 10^6 nodos por movimiento:

Solución habitual: Evaluación de los nodos

- Se utiliza una función de evaluación heurística que evalúa la bondad de un nodo dependiendo de ciertos criterios (en el ajedrez: número de piezas amenazadas, control del centro del tablero…)

  p.ej.
  $eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

En la práctica…

Aplicación: Ajedrez

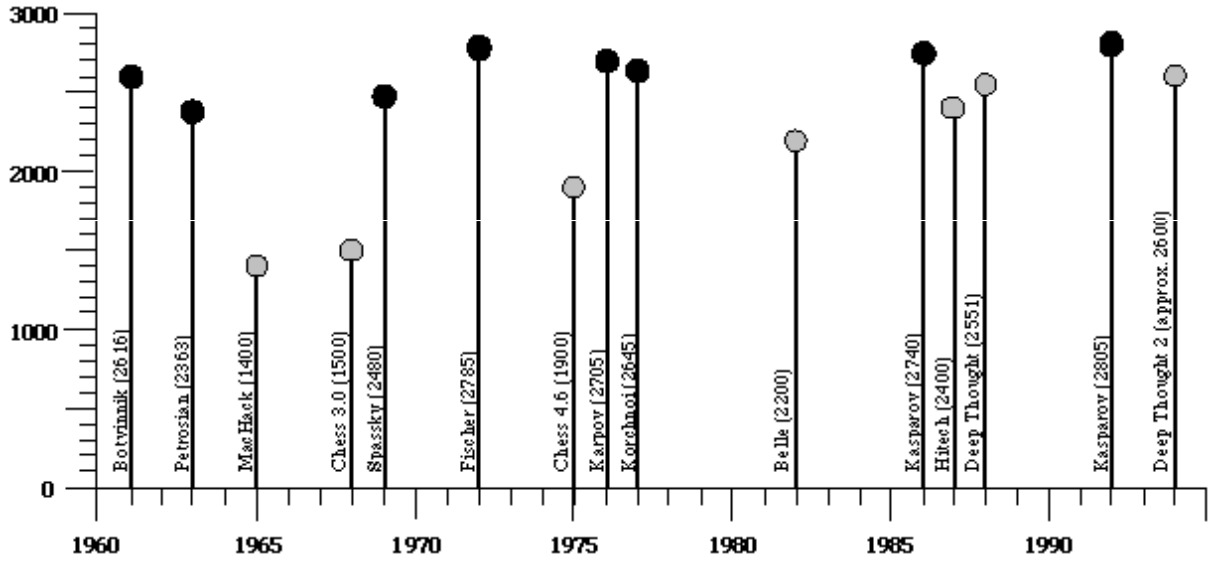
b=35

- 4-ply (novato)
  $d = 4 \rightarrow b^d = 1.5 \times 10^6$
- 8-ply (maestro): Programa típico para PC
  $d = 8 \rightarrow b^d = 2.25 \times 10^{12}$
- 12-ply (¿Kasparov?):
  $d = 12 \rightarrow b^d = 3.4 \times 10^{18}$

  En Deep Blue, el valor medio de b se reducía de 35 a 6 utilizando la poda α-β.

En la práctica…

Aplicación: Ajedrez

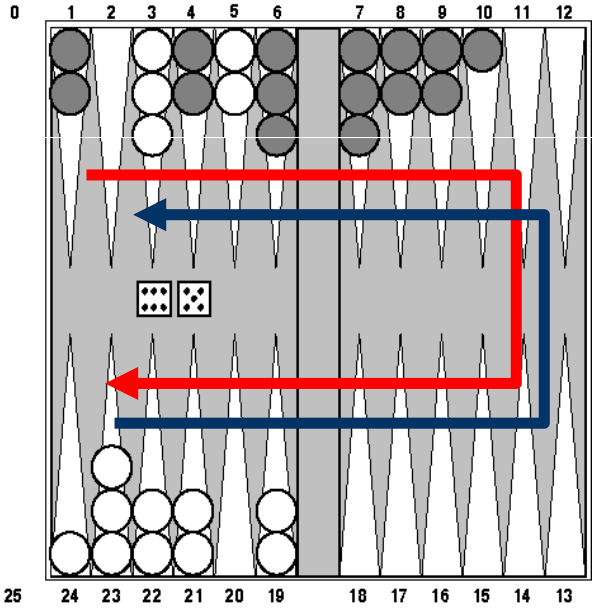


En la práctica…

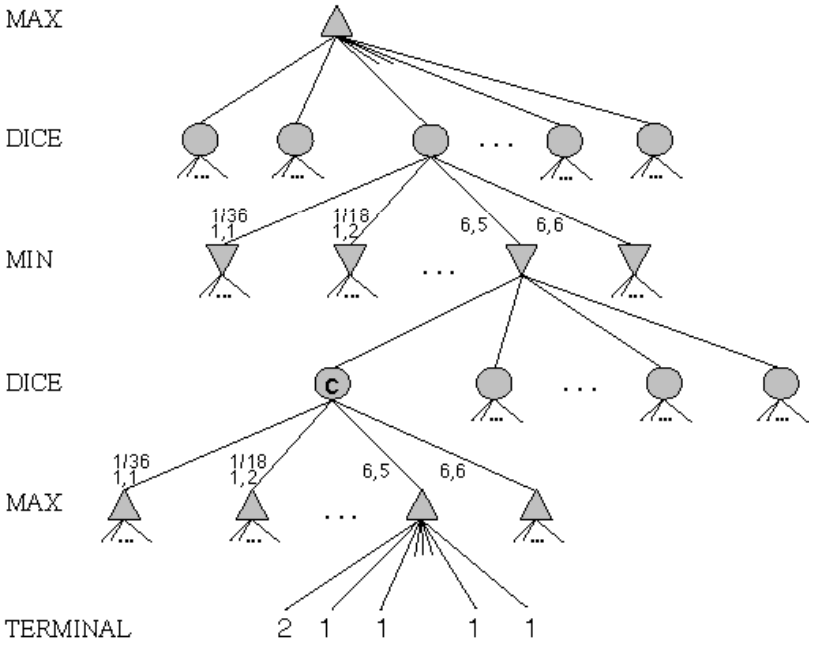
- **Damas**: Chinook venció al campeón del mundo, Marion Tinsley, en 1994 usando una base de datos que definía el juego perfecto para todas los finales de partida con 8 o menos piezas (443748 millones de posiciones).
- **Ajedrez**: Deep Blue venció a Gary Kasparov en 1997 analizando 200 millones de posiciones por segundo y usando 8000 características y heurísticas que le permitían analizar algunas secuencias de hasta 40 ply.
- **Othello**: Los campeones humanos se niegan a jugar contra ordenadores porque son demasiado buenos.
- **Go**: Los campeones humanos se niegan a jugar contra ordenadores porque son demasiado malos (b>300).

Juegos de azar

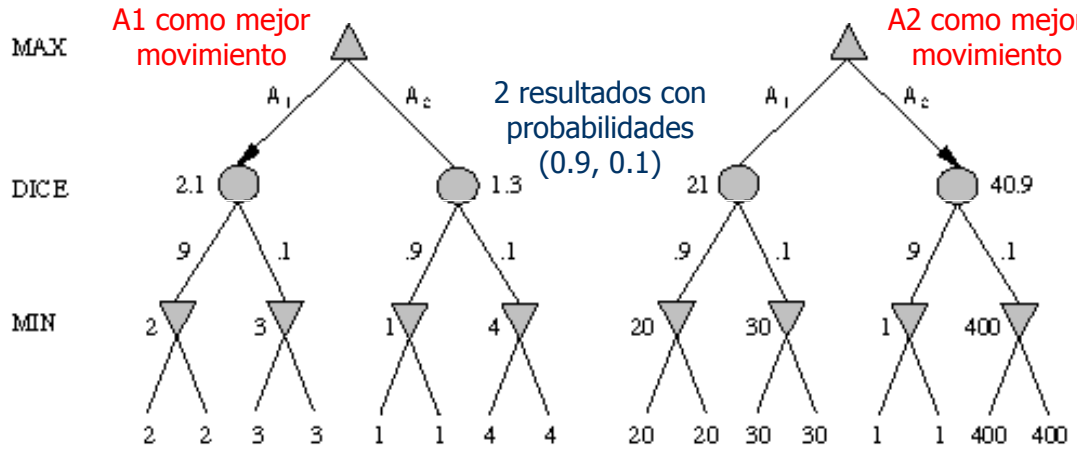
Hay juegos, como el Backgammon, en los que interviene el azar (en forma de dados):



Juegos de azar

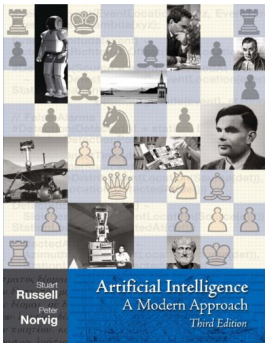


Juegos de azar



Bibliografía

- Stuart Russell & Peter Norvig:
  Artificial Intelligence:
  A Modern Approach
  [Chapter 5: Adversarial Search]
  Prentice-Hall, 3rd edition, 2009
  ISBN 0136042597



- Nils J. Nilsson
  The Quest for Artificial Intelligence
  [Sections 5.4 & 32.1]
  Cambridge University Press, 2009
  ISBN 0521122937