Problemas de búsqueda entre adversarios

Juegos

Introducción, I

- Juegos
  - » Origen, 1928: John Von Newmann
    - – Teorema fundamental de los juegos bipersonales de suma nula.
  - » Desarrollo, 1944:
    - – Von Newmann, J. Morgernsten, O. “Theory of Games and Economic Behaviour”.
- Aplicaciones
  - » Antropología, psicología, economía, política, negocios, biología, IA, etc.
- Elementos:
  - » Jugadores: personas, empresas, naciones, entes biológicos, etc.
  - » Conjunto de estrategias: operadores o acciones
  - » Resultado o Valor del juego: estado/s objetivos/s
  - » Conjuntos de Pagos para cada jugador: función de utilidad sobre las estrategias

Introducción, II

- Clasificación desde diferentes perspectivas:
  - » Cooperación
    - – Cooperativos/no cooperativos
  - » Número de jugadores
    - – n=2, bipersonales:
      - por naturaleza no cooperativos
    - – n>2, n-personales:
      - Pueden ser cooperativos. Dan lugar a coaliciones
  - » Beneficios
    - – Suma nula (habituales en IA): la suma de beneficios y pérdidas de los jugadores debe ser 0 (puede ser constante → reescalado).
    - – Suma no nula.
  - » Duración
    - – Finitos:
      - tienen final programado (nº jugadas, ruinas, etc.)
    - – Infinitos: sin final programado

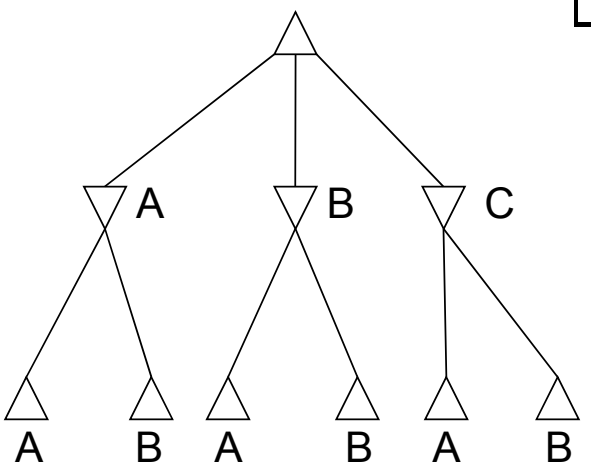
Introducción, III

- Formas de representación de un juego
  - » Forma matricial
    - – *matriz de balances finales* o *matriz del juego*: proporciona la utilidad de cada estrategia de cada jugador para cada acción del resto de jugadores.

		J2	
		A	B
J1	A	2	-3
	B	0	2
	C	-5	10

Pagos de J2 a J1

»Forma de árbol



Juegos bipersonales, I

- Los juegos bipersonales en la IA
  - » Son problemas con contingencias
  - » En ocasiones pueden tener una ramificación alta
    - – por ejemplo en ajedrez, b ≈ 35
  - » Puede haber limitaciones de tiempo
    - – Entorno semidinámico
- En la resolución se utilizan:
  - » Funciones de evaluación
    - – Evalúan los operadores utilizados por cada jugador.
    - – Ayudan a decidir el resultado del juego y las mejores estrategias para cada jugador.
  - » Métodos de poda
    - – Simplificación de la búsqueda.

Juegos bipersonales, II

- Planteamiento general:
  - » 2 jugadores: MAX y MIN (MAX mueve primero):
  - » Estado inicial
    - – Posición del tablero e identificación del primer jugador a mover
  - » Función sucesora:
    - – Lista de pares (movimiento, estado) indica cada movimiento legal y su estado resultante
  - » Función objetivo:
    - – Determina cuándo se acaba el juego (en nodos objetivo o terminales)
  - » Función de utilidad (función ***u***):
    - – Definida en nodos terminales (valores numéricos)
    - – Resultado del juego. Por ejemplo:
      - +1 si gana MAX
      - -1 si gana MIN
      - 0 si empate (tablas)

Juegos bipersonales, III

- Juegos que incorporan AZAR
  - » En ocasiones el *azar* interviene en los juegos
    - – Lanzamientos de monedas, dados, generación de números aleatorios, cartas, etc.
  - » El árbol del juego tiene que reflejar dicha contingencia, introduciendo al azar como si de un jugador más se tratase
  - » La toma de decisiones se puede ver influenciada por la distribución de probabilidad existente sobre las acciones del jugador *azar.*

Juegos bipersonales, IV

» En aquellos juegos en que existe azar, es posible recoger la información procedente de nodos en los que interviene una distribución de probabilidad para determinar los nodos generados a partir del nodo actual.

» Para determinar el valor esperado de la utilidad se procede según la siguiente definición:

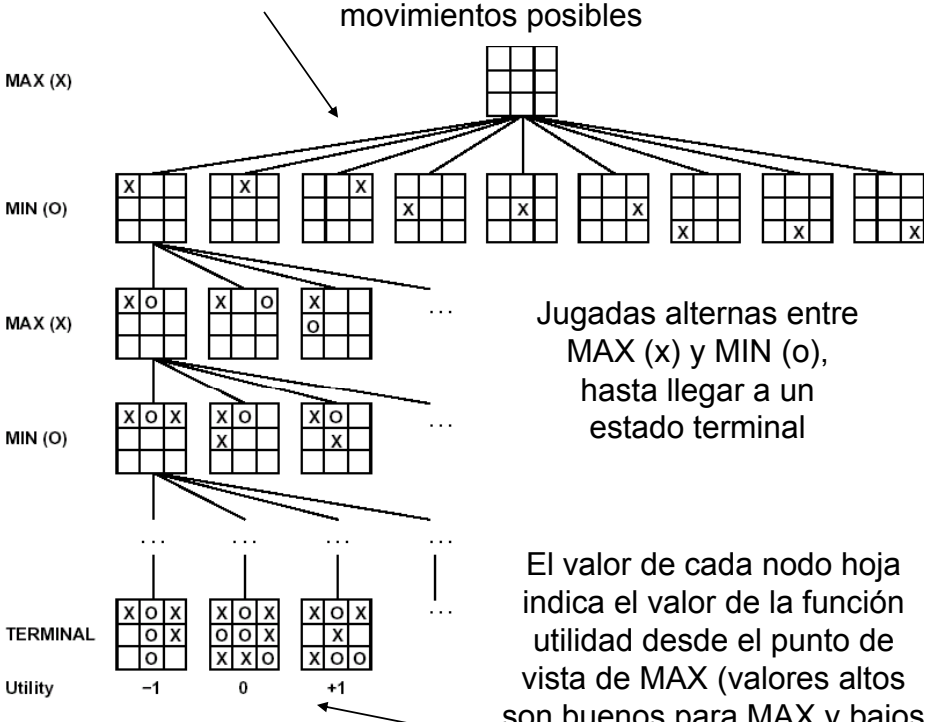
$$EXPECTMINIMAX(n) = \begin{cases} UTILITY(n) & \text{si } n \text{ es terminal} \\ \max_{s \in successors(n)} EXPECTMINIMAX(s) & \text{si } n \text{ es un nodo MAX} \\ \min_{s \in successors(n)} EXPECTMINIMAX(s) & \text{si } n \text{ es un nodo MIN} \\ \sum_{s \in successors(n)} P(s) \cdot EXPECTMINIMAX(s) & \text{si } n \text{ es un nodo ALEATORIO} \end{cases}$$

– Ejemplo: un nodo con lanzamiento de un dado

- Si Equilibrado → $P_i = \frac{1}{6} \quad \forall i = 1, \ldots, 6$
- Si No equilibrado → distintas probabilidades para cada valor del dado

Juegos bipersonales, IV

- Ejemplo: tres en raya



El estado inicial y los movimiento legales de cada jugador definen el *árbol del juego.*

Decisiones óptimas en juegos de dos adversarios, I

- Algoritmo *minimax*
  - » Tiene por objetivo decidir un movimiento para MAX.
  - » HIPÓTESIS
    - Jugador MAX trata de maximizar su beneficio (función de utilidad).
    - Jugador MIN trata de minimizar su pérdida.
  - » Aplicación algoritmo:
    - 1) Generar árbol entero hasta nodos terminales
    - 2) Aplicar función ***u*** a nodos terminales
    - 3) Propagar hacia arriba para generar nuevos valores de ***u*** para todos los nodos
      - minimizando para MIN
      - maximizando para MAX
    - 4) Elección jugada con máximo valor de ***u***
  - » MINIMAX-VALUE(n) =
    - UTILITY(n) Si n es un nodo terminal
    - $\max_{s \rightarrow \text{Sucesor(n)}}$ MINIMAX-VALUE(s) Si n es un nodo MAX
    - $\min_{s \rightarrow \text{Sucesor(n)}}$ MINIMAX-VALUE(s) Si n es un nodo MIN

Decisiones óptimas en juegos de dos adversarios, II

- Algoritmo:

```
function MINIMAX-DECISION(state) return una acción
    inputs: state, estado actual en el juego
    v ← MAX-VALUE(state)
    return una acción de SUCCESSORS(state) con valor v
```

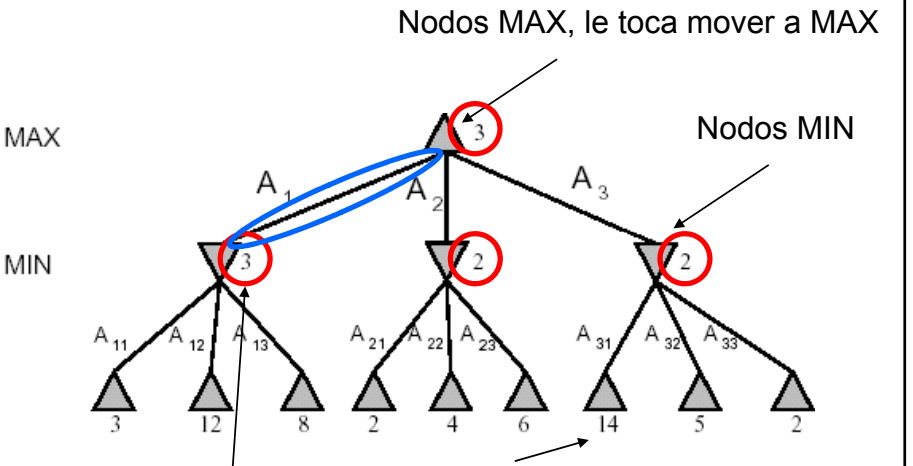
```
function MAX-VALUE(state) returns valor utilidad
    if TERMINAL-TEST(state) then return UTILITY(n)
    v ← - ∞
    for  s en SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns valor utilidad
    if TERMINAL-TEST(state) then return UTILITY(n)
    v ← ∞
    for  s en SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s))
    return v
```

- La complejidad (m = máxima profundidad), como es una búsqueda en profundidad, es:
  - Temporal: $O(b^m)$
  - Espacial: $O(bm)$
- Para juegos reales la complejidad temporal hace que sea impracticable. Es válido para *casos de libro.*

Decisiones óptimas en juegos de dos adversarios, III

- Ejemplo



• La mejor jugada de MAX es A_1 porque genera el mayor valor minimax entre sus nodos sucesores: **ÓPTIMA**
• La mejor jugada entonces de MIN es A_{11} porque genera el menor valor minimax entre sus nodos sucesores.

Ejemplo de juego con azar, I

- Sean dos jugadores, MAX y MIN. Para poder jugar han de depositar una fianza de 1€ en el *pot* (bote en el centro de la mesa). Se reparte una carta a cada jugador de un mazo que contiene, a partes iguales, Ases (A) y Reyes (K). Una vez repartidas las cartas el jugador MAX escoge su jugada (según muestra la figura adjunta).
  - » MAX siempre está obligado a ***apostar* 2**, **4** ó **6 €**.
  - » Después de anunciar su jugada, efectúa lo propio el jugador MIN. En su caso tiene dos opciones:
    - – ***ver la apuesta*** (en cuyo caso iguala la cantidad apostada por MAX) o
    - – ***no ver la apuesta*** (pasar).

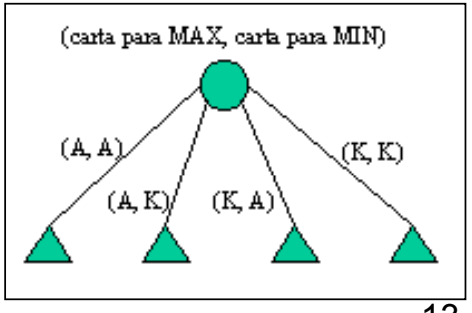
PAGOS:

1) Si MIN ***no ve la apuesta*** pierde el dinero que puso en el bote.

2) Si MIN ***ve la apuesta*** se vuelven las cartas:

i) Si las cartas son diferentes gana la mejor, con el criterio: A es preferida a K

ii) Si ambas cartas son iguales se reparte el bote equitativamente.



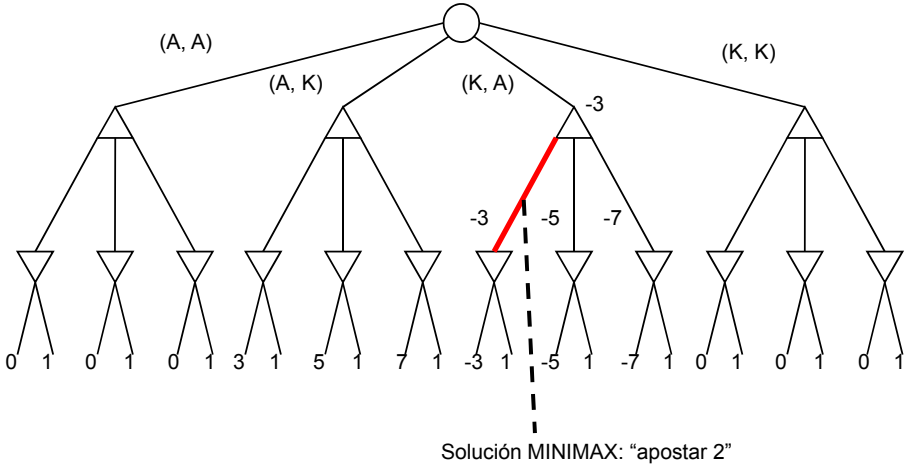
Ejemplo de juego con azar, II

Representar el árbol del juego indicando en los nodos hoja los pagos según la función de utilidad de MAX "incremento de capital obtenido en la jugada".

Indicar cuál sería la estrategia preferida para el jugador MAX en la jugada (K, A) según el criterio MINIMAX

△ 3 posibles acciones: +2, +4, +6

▽ 2 posibles acciones: ver, no-ver



Ej. de pago si secuencia de jugada es [(K,A), +4, ver]
Ver → comparar(K,A) → gana(MAX)
Pago-a-MAX = 4 + 1 = 5 (4 de ver la apuesta, 1 del bote)
(¡ojo!, el pago representa **incremento** de capital)

Decisiones imperfectas en juegos de dos adversarios

- Decisión imperfecta ←→ Estrategia Mixta
- El algoritmo *minimax* asume una expansión hasta el final (en realidad es imposible).
  - » Se usa una función de evaluación (***f***), que sea una estimación de ***u***.
- Función de evaluación:
  - » El papel que hace ***f*** es el de ***u*** en nodos terminales, ordenando de manera correcta los nodos en los que se calcula
  - » Ejemplos:
    1. Estrategia con:
       - 50% posibilidades de ganar,
       - 25% de perder,
       - 25% de empate.

       F = 1*0.50+(-1)*0.25+0*0.25=0.25
    2. En ajedrez, valorando piezas: peón = 1, alfil = 3, ...
       - MAX = fichas-blancas

       F = (num-peones-negros)*1 +
       (num-alfiles-negros)*3 -
       (num-peones-blancos)*1 –
       (num-alfiles-blancos)*3 …

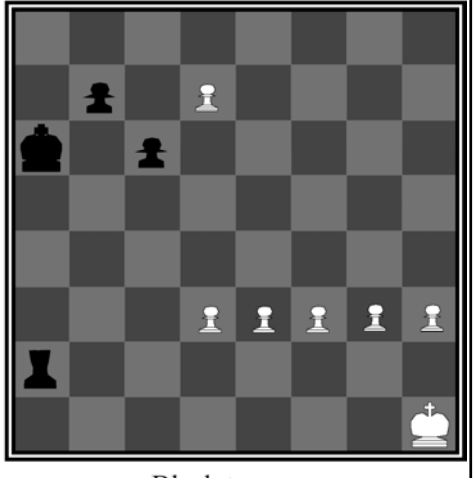
Decisiones imperfectas en juegos de dos adversarios

- Dada una función de evaluación ***f***, se puede aplicar una búsqueda *minimax* con límite de profundidad:
  - » Se elige un límite de profundidad
    - – Observación: el límite puede tener una posición desventajosa en un nivel más abajo.
  - » Se pueden elegir sucesivos límites de profundidad.
  - » El límite de profundidad se debería aplicar sólo a posiciones “inactivas”.
    - – Por ejemplo, en ajedrez, serían posiciones en las que es poco probable que existan capturas

Problema del horizonte

Surge cuando el programa se enfrenta a una acción del oponente, inevitable y que causa serios perjuicios.

Ejemplo: en la figura anexa, peón blanco amenaza convertirse en dama. Torre negra amenaza con jaque. La ventaja actual es negra y la inmediata futura es blanca (evaluación calidad piezas).



Black to move

Decisiones imperfectas en juegos de dos adversarios

- Ejemplo: Tres en raya
  - » Definimos la función de utilidad:

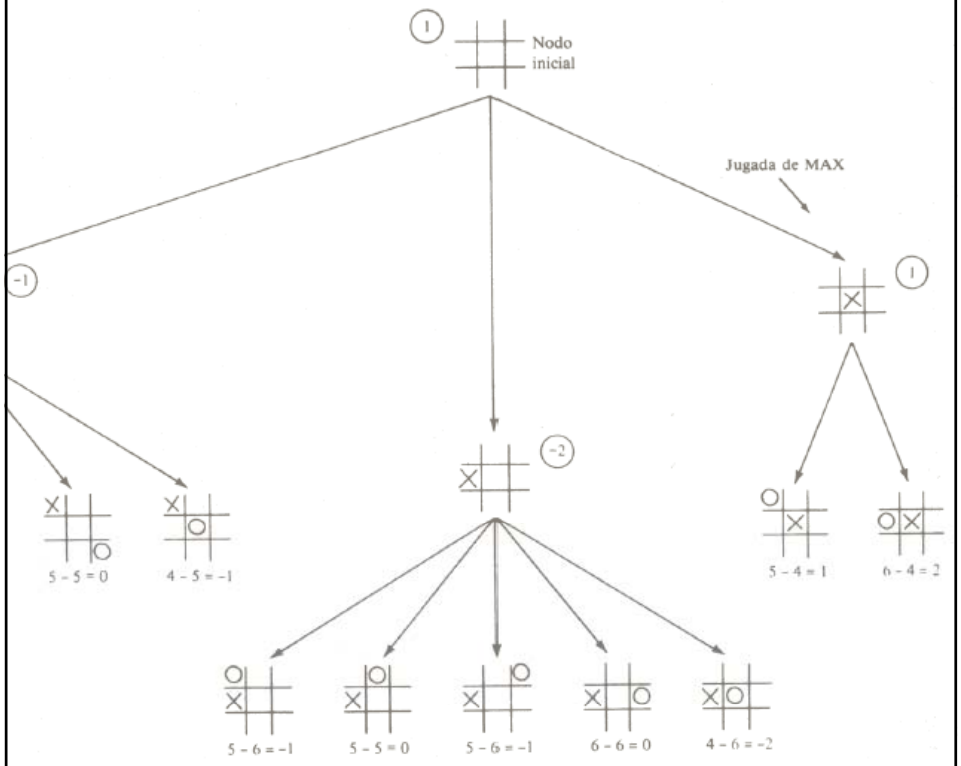
f(n)	Fcd-max - Fcd-min	si *n* no es una solución en que gane alguno de los jugadores
	infinito	Si gana MAX
	- infinito	Si gana MIN

Fcd-max: número de filas, columnas o diagonales libres para MAX
Fcd-min: número de filas, columnas o diagonales libres para MIN

- Exploración y evaluación:
  - » El procedimiento de exploración visto separa por completo el proceso de generación del árbol de exploración y la evaluación de posiciones.
  - » Se puede reducir el esfuerzo requerido si se hace evaluación de los nodos finales y se llevan hacia atrás esas evaluaciones con la generación el árbol

Decisiones imperfectas en juegos de dos adversarios

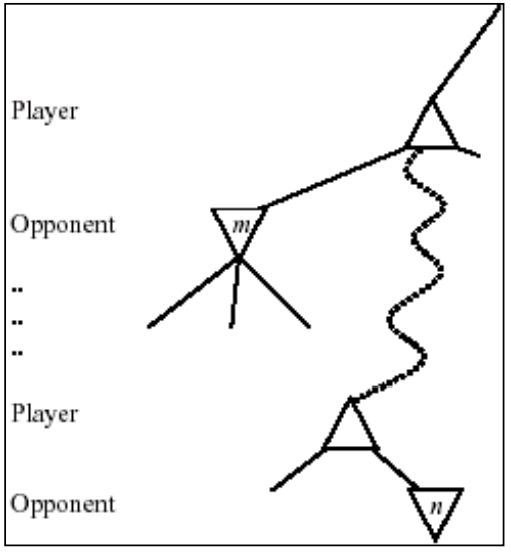
- Ejemplo: Tres en raya



ver Nilsson

Poda $\alpha - \beta$

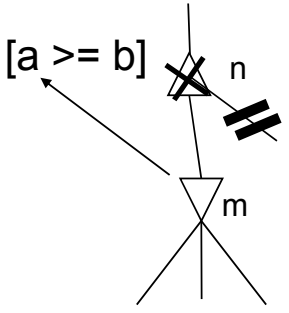
- Consiste en tratar de localizar la decisión óptima *minimax* sin tener que explorar todos los nodos del árbol.
- Aplicable a árboles de cualquier profundidad
- Puede podar subárboles enteros
- Principio general

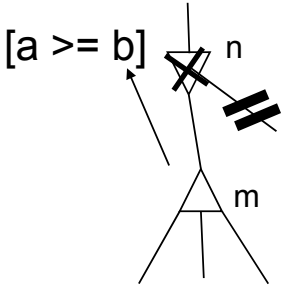


El algoritmo efectúa una búsqueda en profundidad. Si durante la misma se produce que ***m*** es mejor que ***n*** para un jugador, entonces nunca se llegará a ***n*** en el juego

Poda $\alpha-\beta$

- Fundamentos del algoritmo de poda
  - » Definimos
    - – El valor α es una cota inferior para el valor obtenido por propagación desde los sucesores de un nodo.
    - – El valor β es una cota superior para el valor obtenido por propagación desde los sucesores de un nodo.
  - » Poda
    - – Si *n* es ascendente (padre) de *m* y se verifica alguna de las condiciones siguientes, no hace falta seguir examinando por debajo de *n* (se producen podas). El nodo *n* no afecta al resultado final (valor de la acción que lleva al nodo *n*) y es *prescindible*





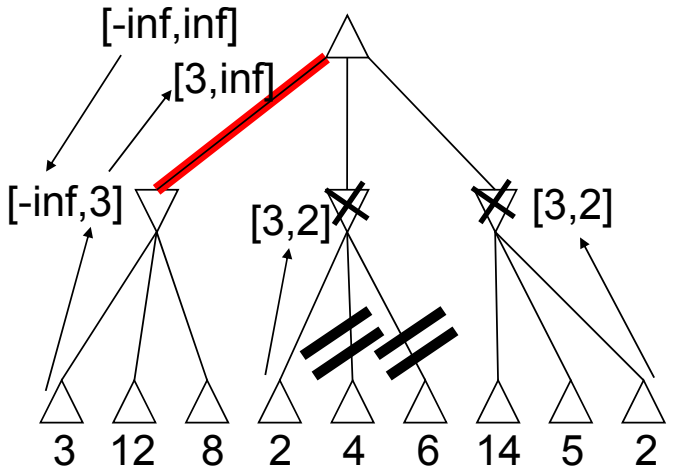
Algoritmo de poda $\alpha-\beta$

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

  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v

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
  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
```

Es similar al *minimax* salvo sendas líneas en las rutinas MIN-VALUE y MAX-VALUE que mantienen los valores de *alpha* y *beta*

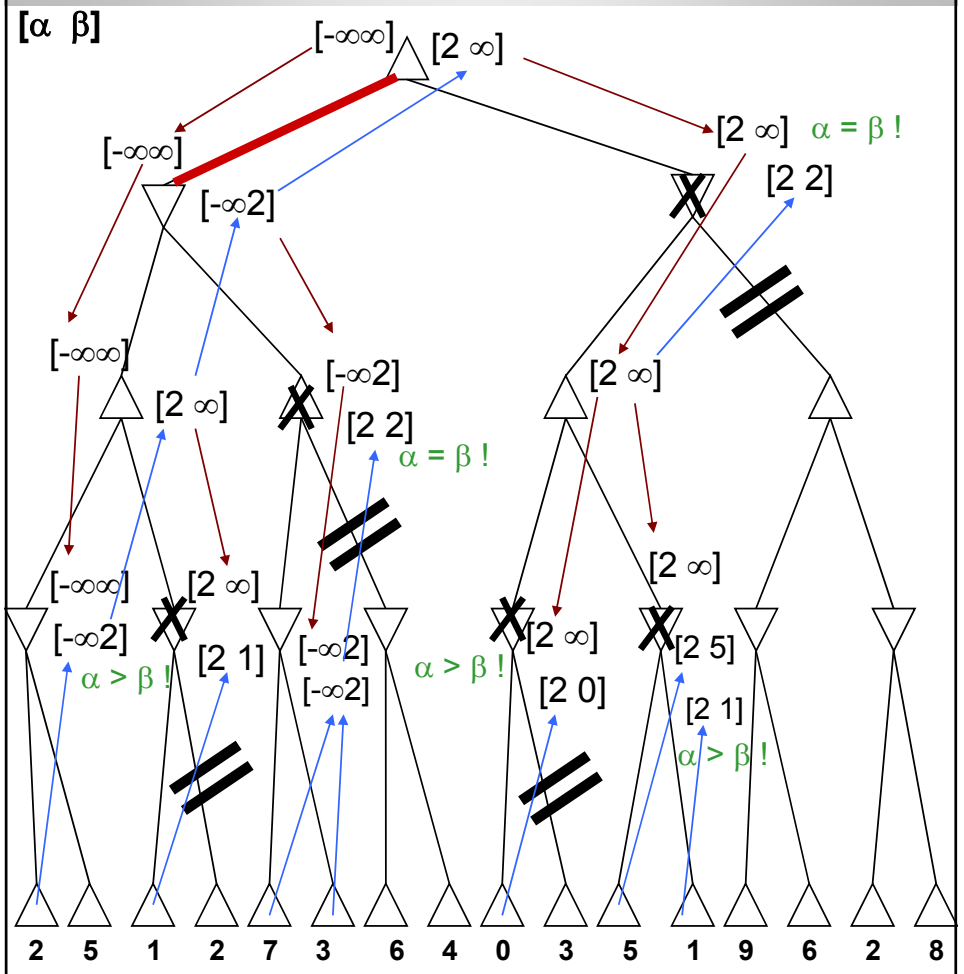
Ejemplos, I

Ejemplo sencillo de poda



- Estimación en el ajedrez
  - » Un agente puede examinar unas 1000 posiciones/segundo.
    - – Si tenemos 150 segundos para pensar un movimiento, entonces, como b es aproximadamente 35, podemos descender 3 ó 4 niveles en el árbol.
      - La poda $\alpha - \beta$ va a permitir bajar hasta más niveles.

Ejemplos, II



No mejoran β = 2

Efectividad de la poda $\alpha-\beta$

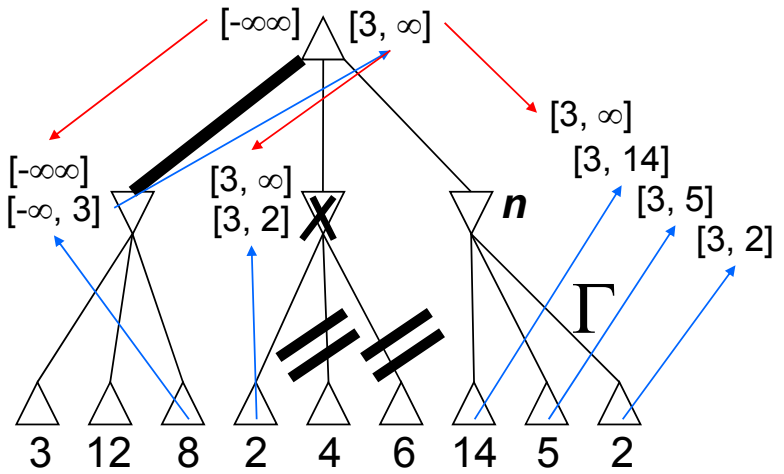
- La poda depende del orden en que se examinan los nodos
  - » En el ejemplo de la página siguiente, no se producen podas por debajo del nodo *n* porque la rama Γ se expande la última.
- Si se pudiera elegir el nodo más conveniente (el nodo de *f* mínima en el caso de MIN):
  - » Knuth y Moore [1], demostraron que la complejidad temporal es:

$$O(b^{d/2})$$

  - » Por tanto, el factor de ramificación efectivo sería $\sqrt{b}$ en lugar de *b.*
    - En el ajedrez tendríamos $b = \sqrt{35} \approx 6$
      - Podríamos bajar hasta el nivel 8.
  - » Es una situación ideal (supondría expandir los nodos para calcular el de menor *f*).

[1] Donald E. Knuth; Ronald W. Moore; An analysis of alpha-beta pruning. Artificial Intelligence 6(4); 293-326 (1975)

Efectividad de la poda $\alpha-\beta$



- Knuth y Moore demostraron también que si examinan los sucesores de forma aleatoria para valores moderados de *b,* la complejidad temporal es aproximadamente:

$$O(b^{3d/4})$$