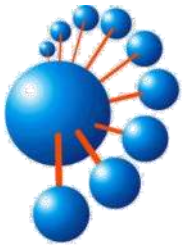
Búsqueda con adversario (parte 1)

Julio Godoy

DIICC



- Ambiente multiagente
  - Comportamiento de otros agentes muchas veces impredecible
  - Dificultan la toma de decisiones de un agente
  - Asumimos ambientes competitivos
    - Objetivos de los agentes están en conflicto
  - Teoría de juegos
    - Rama de la economía que considera ambientes multiagente como juegos

Ambiente

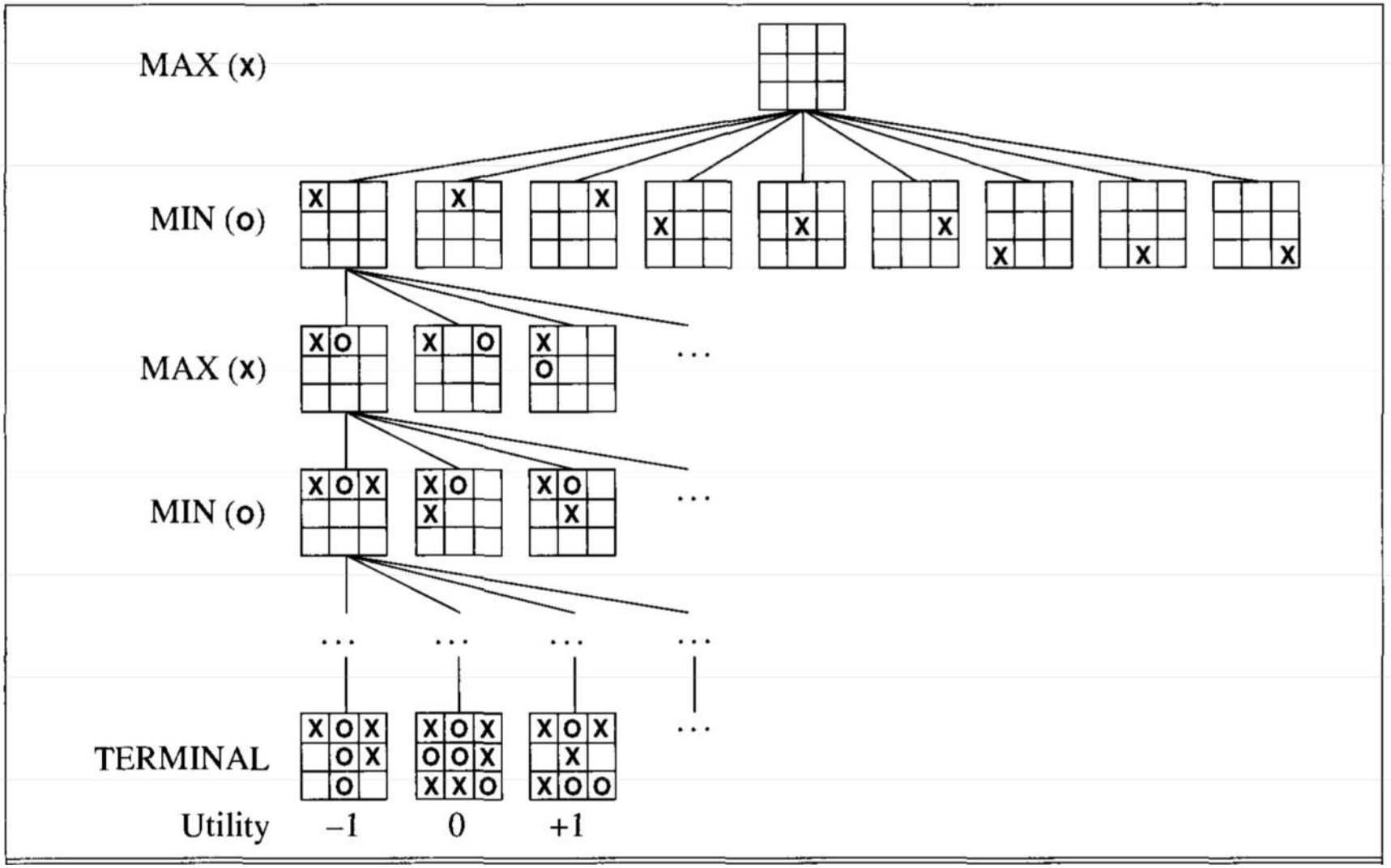
- Determinista
- Observable
- Decisiones se toman **en turnos**
- Dos agentes/jugadores
- La utilidad de un estado final es opuesta con la misma magnitud
  - Juego de suma cero/constante o zero sum game

Formulación de Problemas de Búsqueda

- estado inicial
  - tablero, posiciones de piezas
  - ¿de quién es el turno?
- operadores
  - definen los movimientos legales
- test objetivo
  - determina cuándo termina el juego
  - calcula el resultado
    - ganado, perdido, empate
- función de utilidad o resultado
  - valor numérico que mide la ganancia del juego

Juegos para dos Personas

- En juegos con dos jugadores
  - llamados MIN y MAX
  - generalmente MAX mueve primero, luego se turnan
- MAX debe encontrar una estrategia para llegar a un estado ganador
  - no importa lo que haga MIN
- MIN hace lo mismo
  - o al menos trata de evitar que MAX gane
- información completa
  - ambos jugadores conocen el estado completo del ambiente


MAX (x)
MIN (o)
MAX (x)
MIN (o)
TERMINAL
Utility
−1
0
+1

Decisiones Perfectas

- Basadas en una estrategia racional (óptima) para MAX
  - recorren todas las partes relevantes del árbol de búsqueda
    - esto debe incluir posibles movidas de MIN
  - identifican un camino que lleva a MAX a un estado ganador

- A menudo no es muy práctico
  - limitaciones de tiempo y espacio

Estrategia Minimax

- Estrategia óptima para MAX:

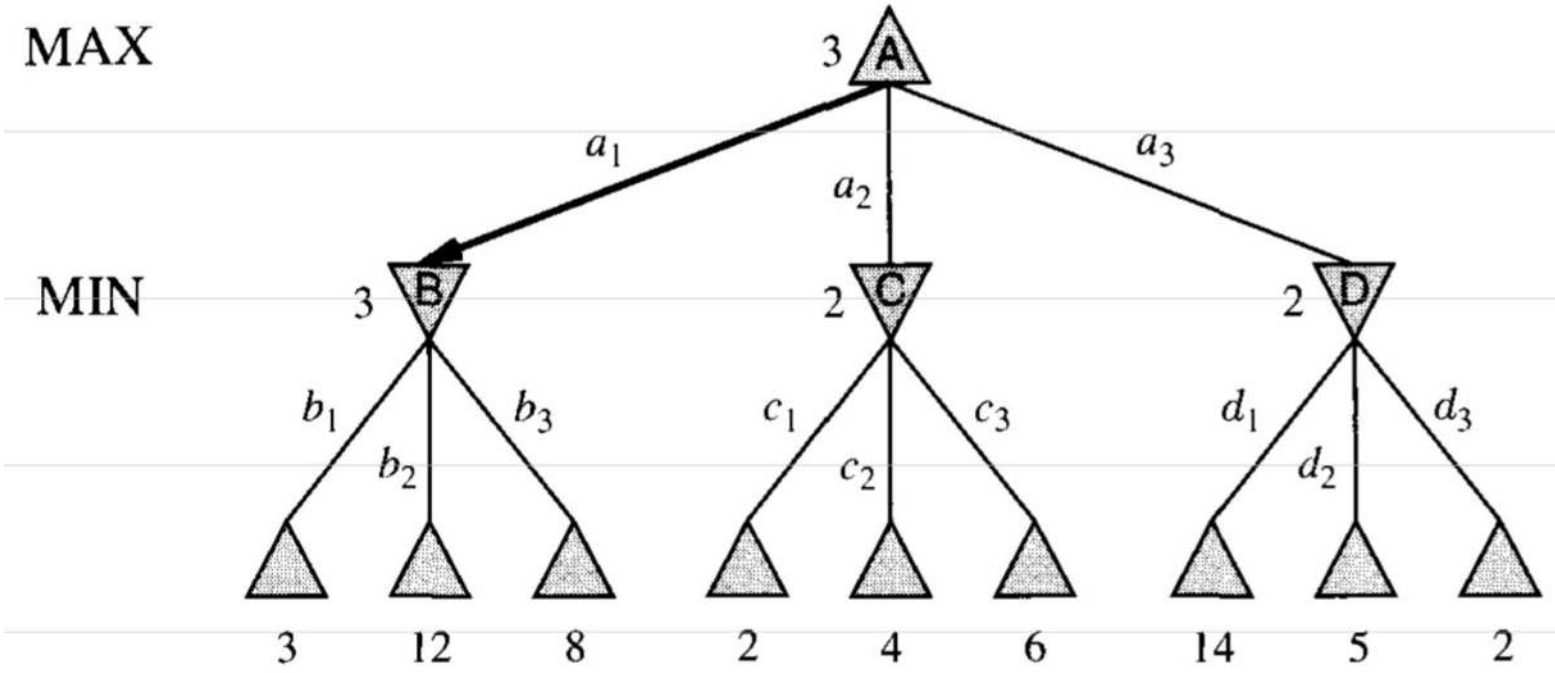
•Genera el árbol de juego completo

•calcula el valor de cada estado terminal basado en la función de utilidad

•calcula la utilidad de los nodos de mayor nivel, partiendo de los nodos hoja hacia la raíz

•MAX selecciona el nodo con valor más alto

•MAX supone que MIN en su movida seleccionará el nodo que minimiza el valor



$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

Algoritmo MiniMax

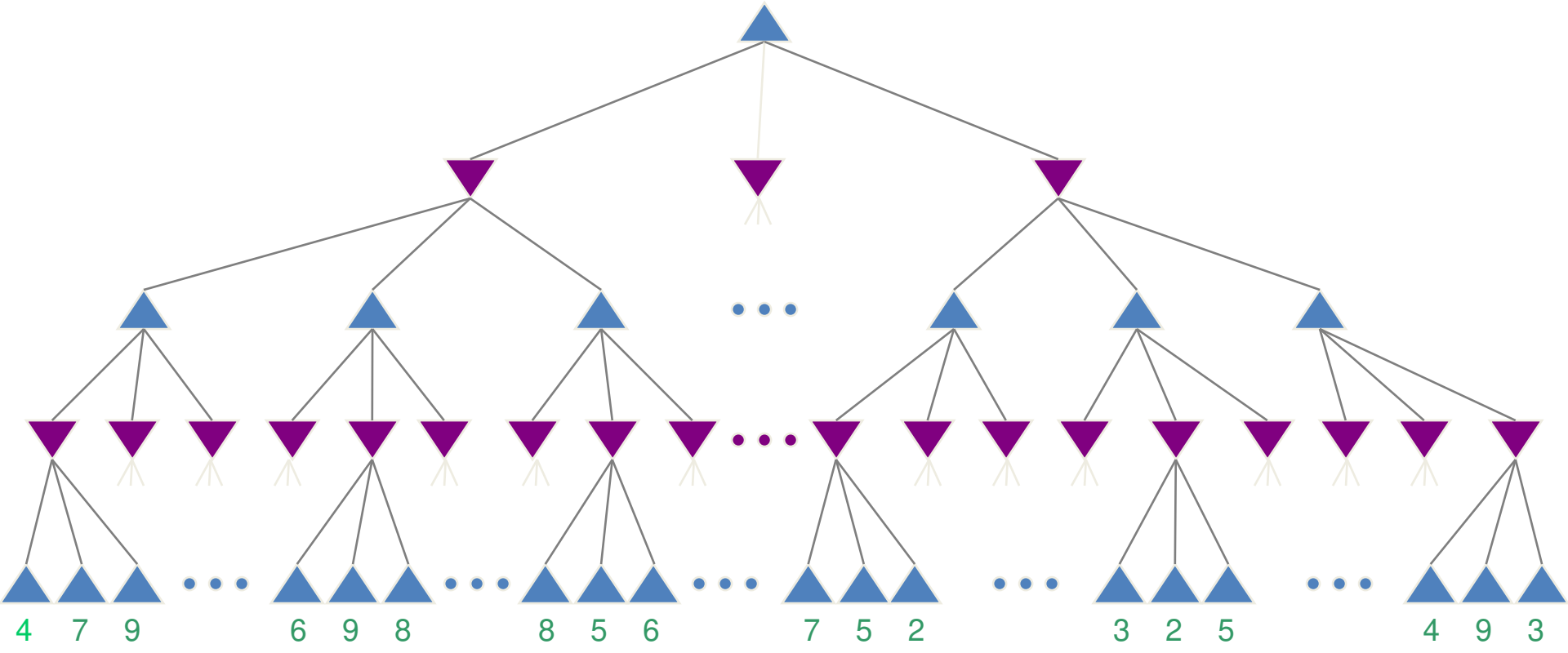
function Minimax-Decision(*juego*) **returns** *operador*
  for each *op* **in** Operadores[*juego*] **do**
    Valor[*op*] := Minimax-Valor(Aplica(*op,juego*), juego)
  end
  return *op* con el mayor valor[*op*]

function Minimax-Valor(*estado,juego*) **returns** *valor de utilidad*
  if Test-Objetivo [*juego*](*estado*) **then**
    return Utilidad[*juego*](*estado*)
  else if mueve Max **then**
    return el mayor Minimax-Valor de exitosos(*estado*)
  else
    return el menor Minimax-Valor de exitosos(*estado*)

Propiedades Minimax

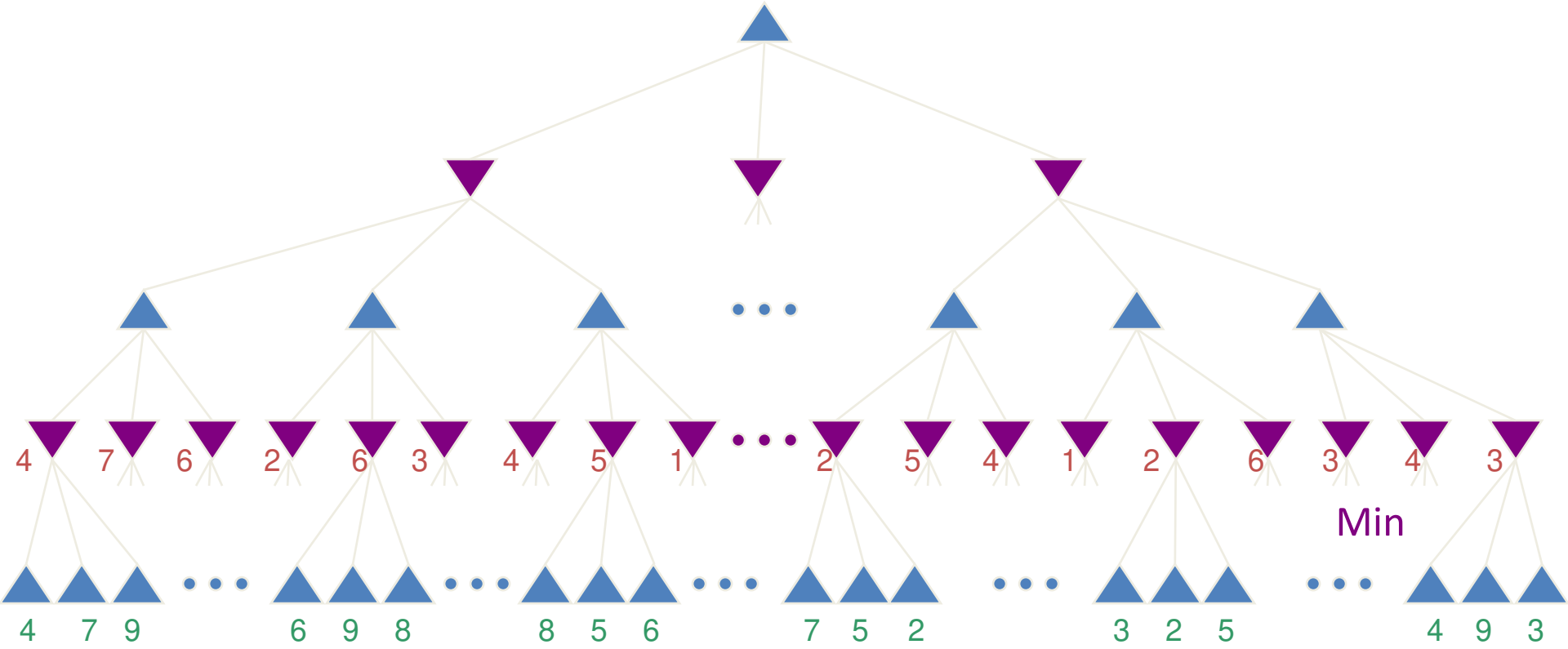
- Basado en búsqueda en profundidad
  - implementación recursiva
- complejidad temporal: $O(b^m)$
- complejidad espacial: $O(bm)$
  - donde b es el factor de ramificación y m la profundidad máxima del árbol de búsqueda

Ejemplo Minimax



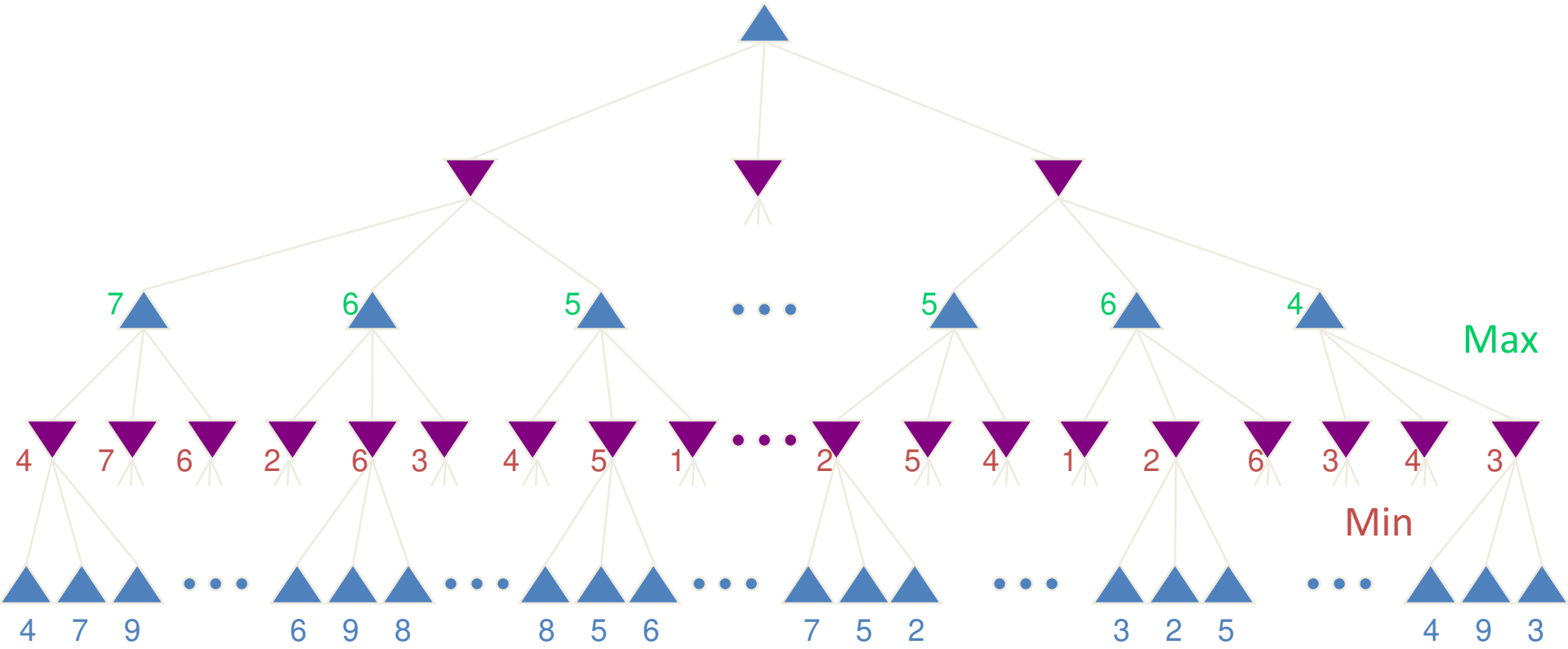
Nodos terminales: valores calculados a partir de la función de utilidad

Ejemplo Minimax

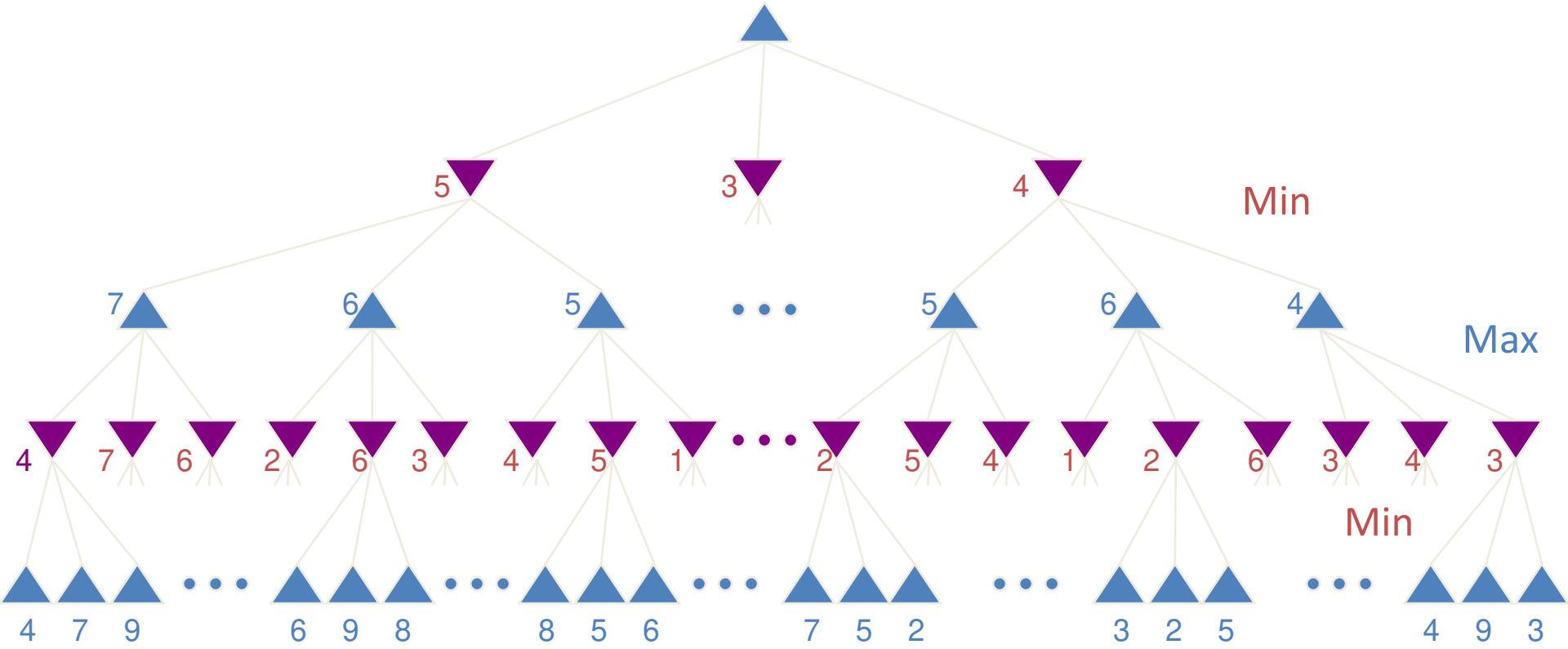


Otros nodos: valores calculados vía algoritmo minimax

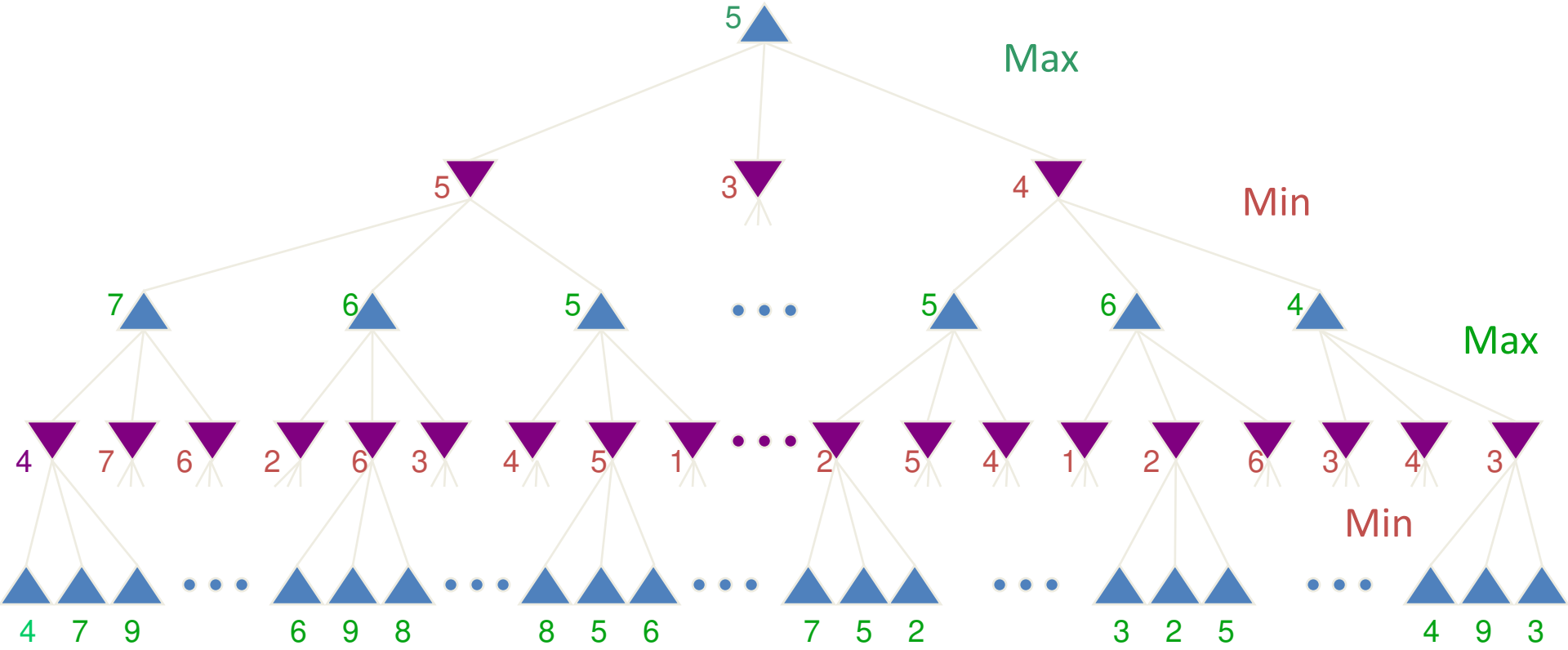
Ejemplo Minimax



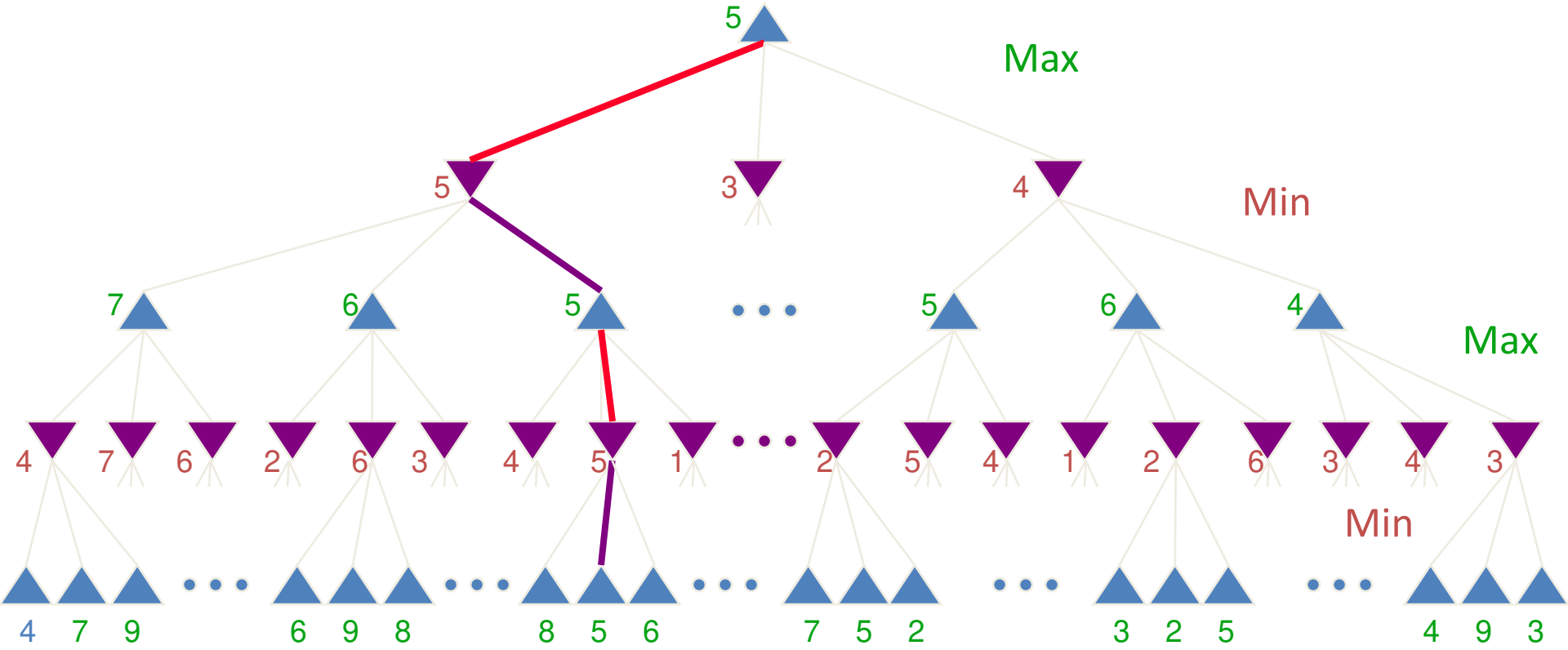
Ejemplo Minimax



Ejemplo Minimax



Ejemplo Minimax



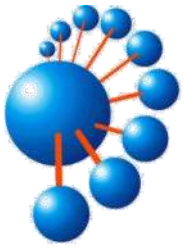
Movidas por Max y contramovidas por Min

Fin parte 1

Búsqueda con adversario (parte 2)

Julio Godoy

DIICC



Poda

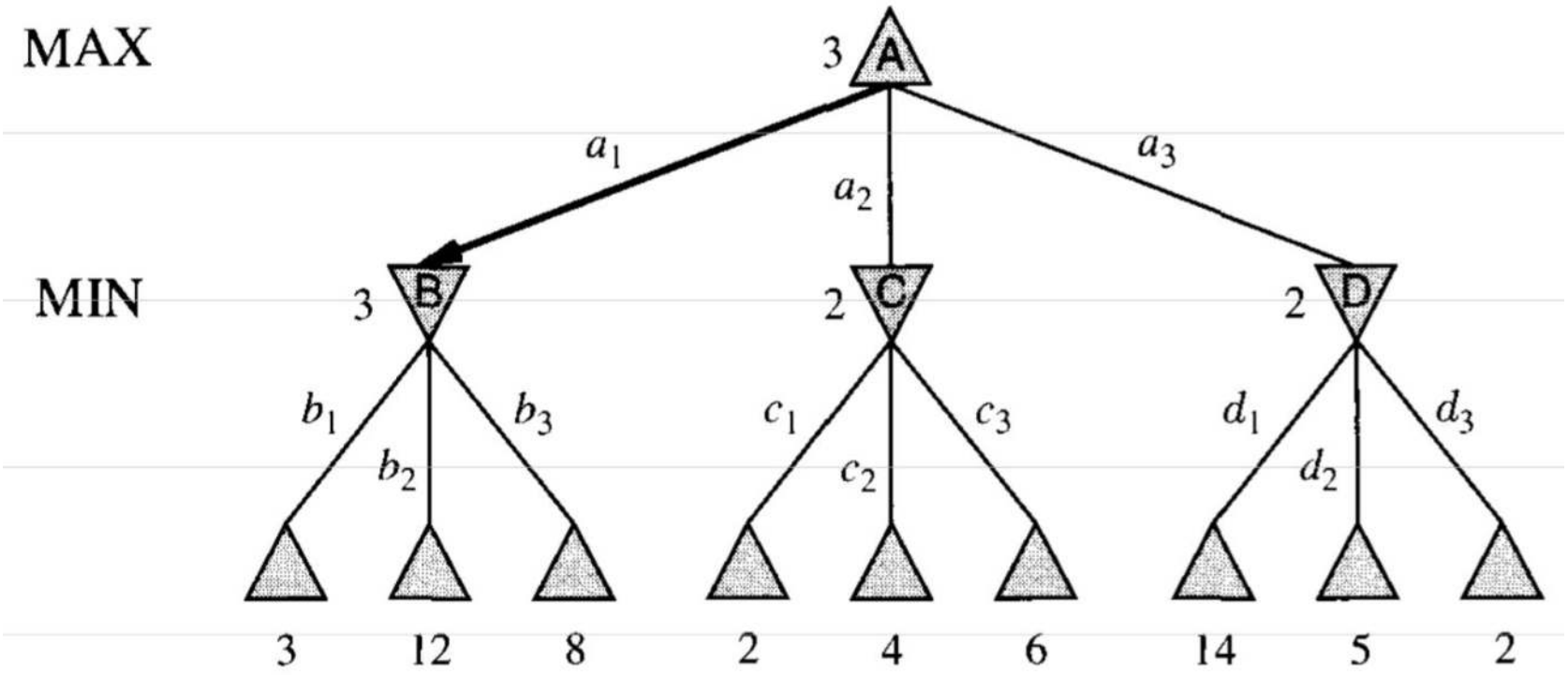
- Descarta partes del árbol de búsqueda
  - que garantizadamente no serán buenos movimientos
- Resulta un ahorro substancial tanto en tiempo como en espacio
  - sin embargo, la parte de la tarea que queda puede ser exponencial

Poda Alfa-Beta

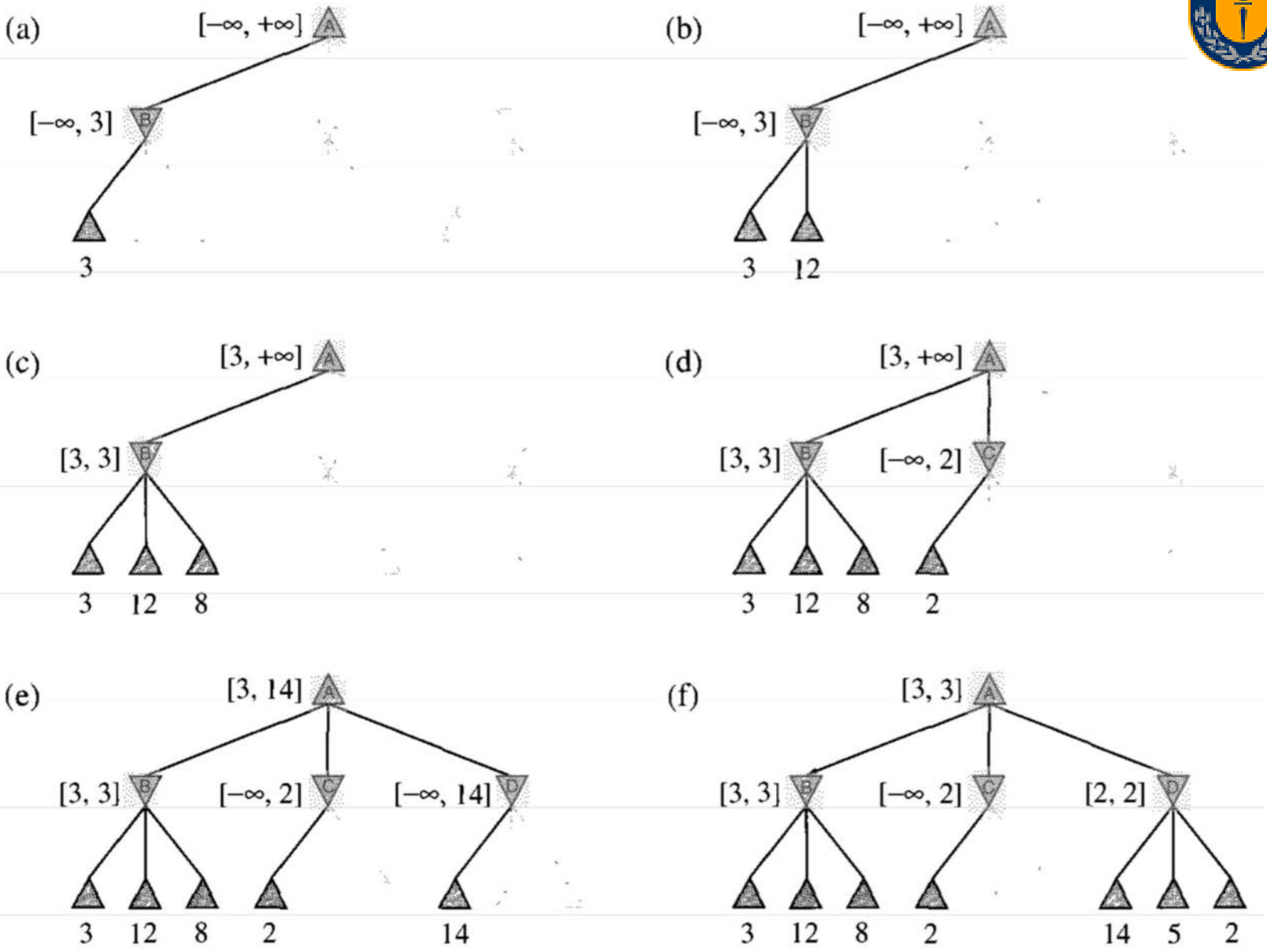
- Descarta movimientos que con certeza no tendrán una buena evaluación
  - nodos superiores en el árbol tienen una mejor opción
- Se aplica a movimientos de ambos jugadores
  - α indica la mejor elección para Max así que nunca disminuirá
  - β indica la mejor elección para Min así que nunca aumentará
- Es una extensión del minimax
  - como resultado entrega el mismo movimiento que minimax, pero menos costoso

Algorítmo Alfa-Beta

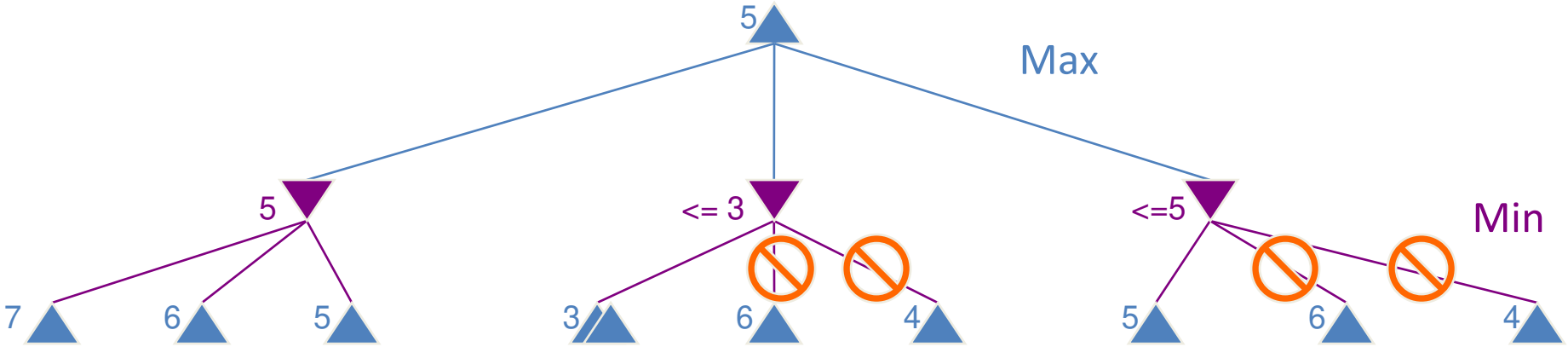
```
function Max-Value(estado, juego, alfa, beta) returns el valor minimax de estado
  if Cutoff-Test (estado) then return Eval(estado)
    for each s in  Sucesores(estado) do
        alfa := Max (alfa, Min-Value(s, juego, alfa, beta))
      if alfa  >= beta then return beta
    end
    return alfa

function Min-Value(estado, juego, alfa, beta) returns el valor  minimax de estado
  if Cutoff-Test (estado) then return Eval(estado)
    for each s in  Sucesores(estado) do
       beta := Min (beta, Max-Value(s, juego, alfa, beta))
      if beta <= alfa then return alfa
    end
    return beta
```



$$
\begin{aligned}
\textsc{Minimax-Value}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3,z,2) \qquad \text{where } z \leq 2 \\
&= 3.
\end{aligned}
$$


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

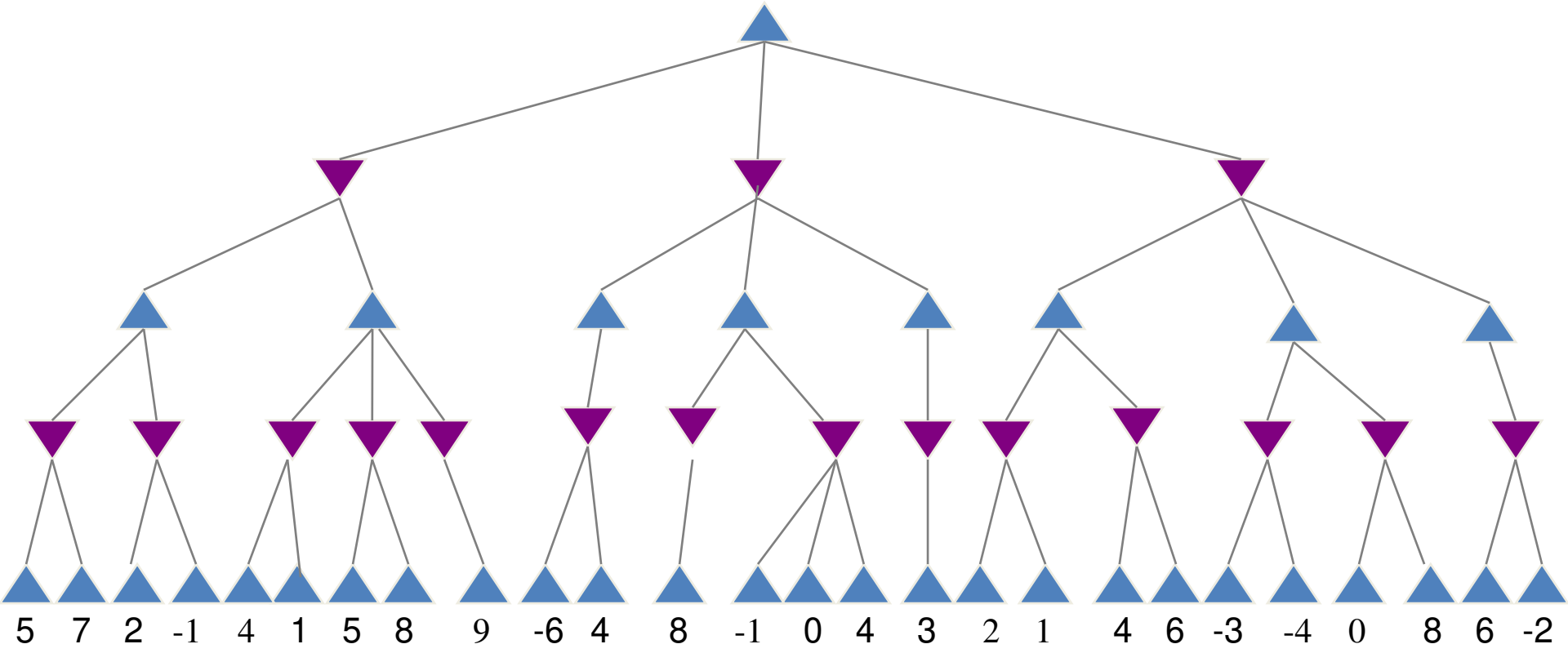
Ejemplo 1 Alfa-Beta



α mejor elección para Max 5
β mejor elección para Min 7 -> 6 -> 5 -> 3

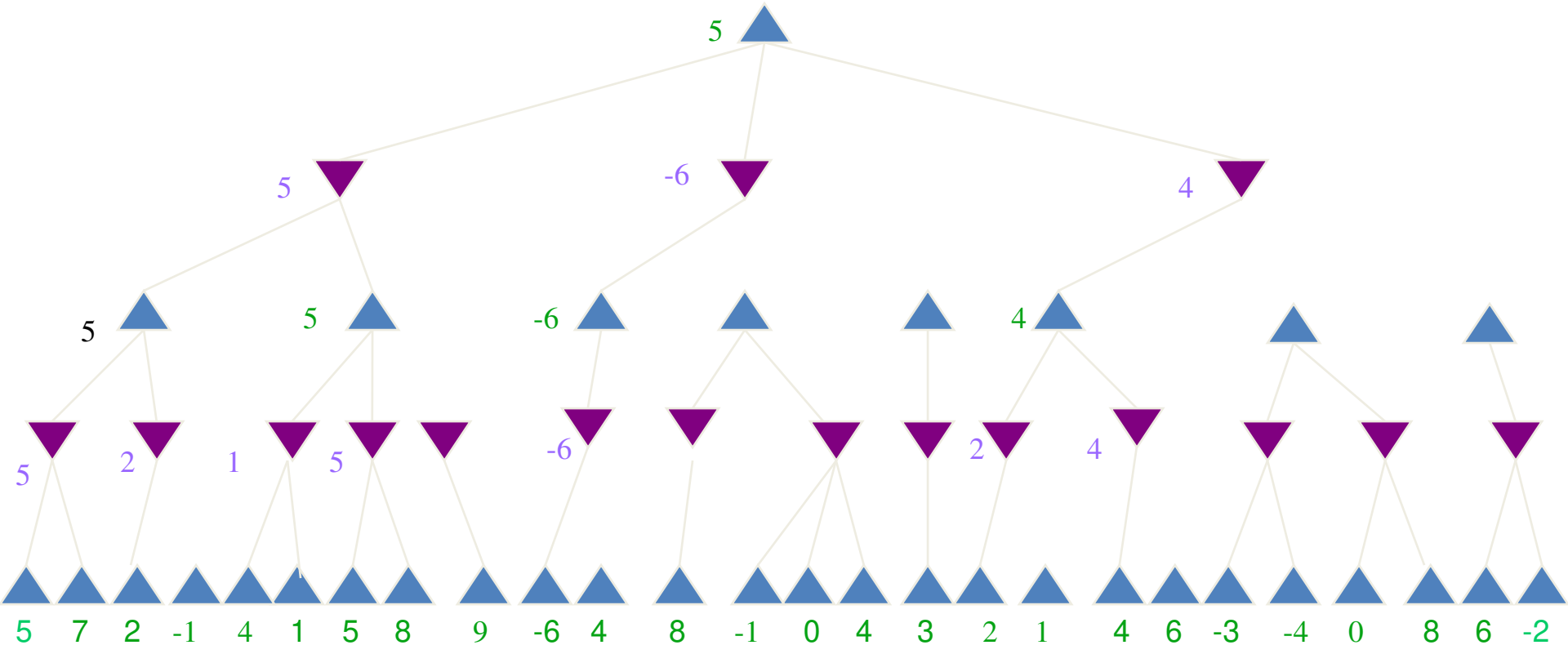
- Algunas ramas del árbol pueden ser podadas ya que ellas nunca serán consideradas
  - después de examinar una rama, Max ya sabe si ella no le interesará, Min eligiría un valor menor del que Max ya tiene a su disposición

Ejemplo 2 Alfa-Beta

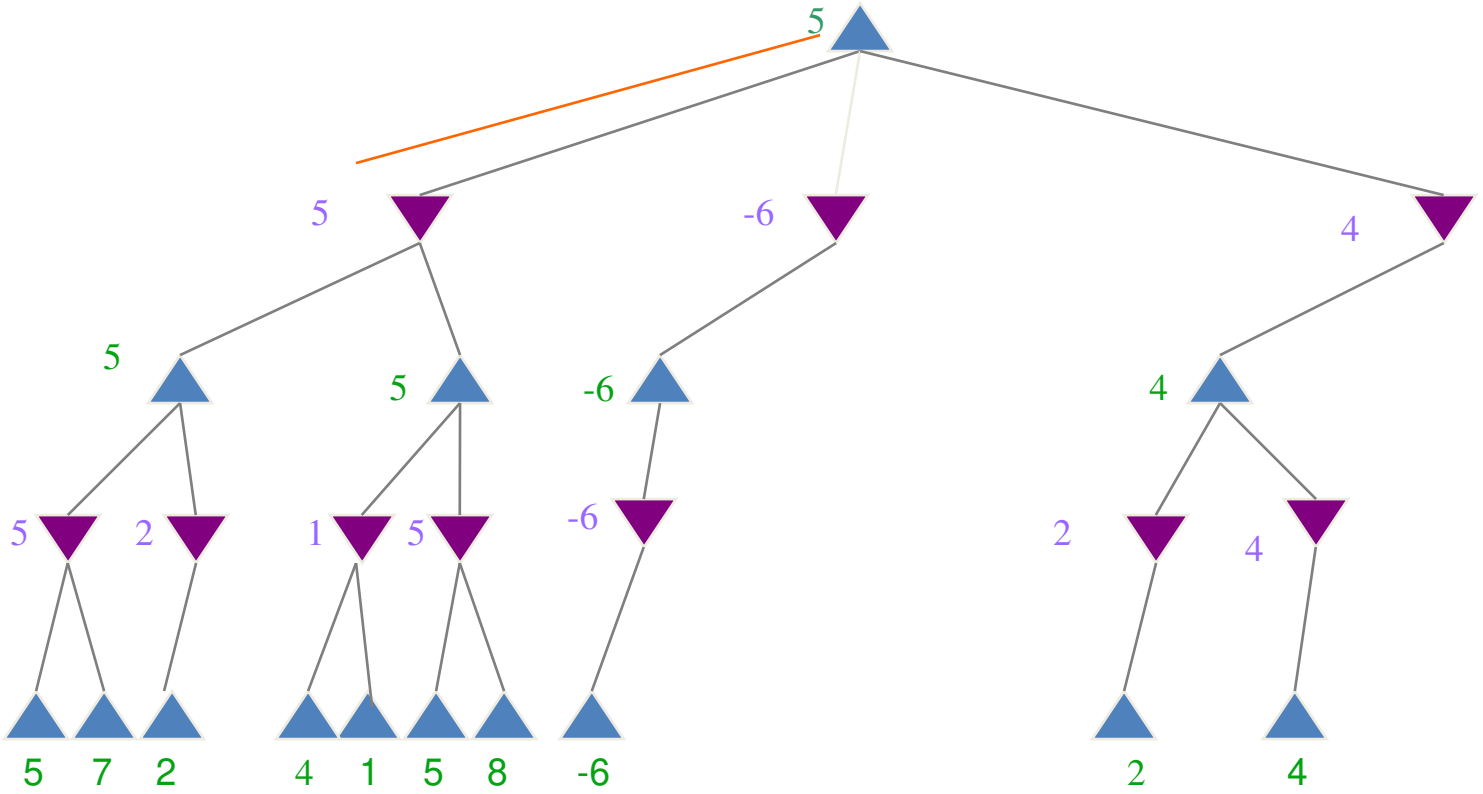


Nodos terminales: valores calculados a partir de la función de utilidad

Ejemplo 2 Alfa-Beta

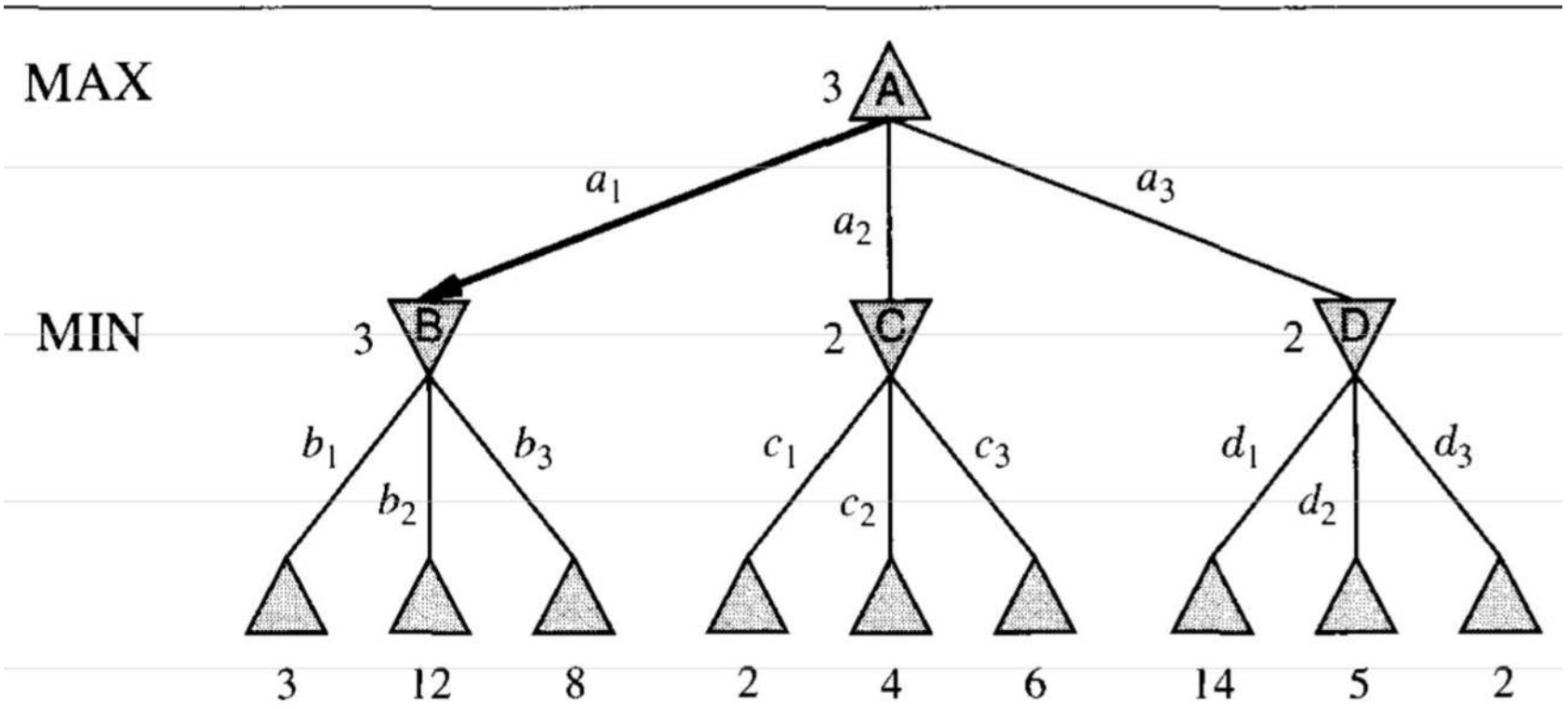


Ejemplo 2 Alfa-Beta



Propiedades Poda Alfa - Beta

- El orden en que se generan los nodos terminales a evaluar importa



Propiedades Poda Alfa - Beta

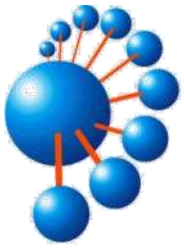
- El orden en que se generan los nodos terminales a evaluar importa
- Con ordenamiento 'perfecto':
  - C. Temporal: $O(b^m) \rightarrow O(b^{m/2})!$
  - En ajedrez, $35^{100} \rightarrow 35^{50}$
  - Sin embargo, 35^{50} sigue siendo un número muy grande.

Fin parte 2

Búsqueda con adversario (parte 3)

Julio Godoy

DIICC



Decisiones Imperfectas

- La búsqueda completa es impracticable para la mayoría de los juegos

- Ajedrez: 35^{100} (grafo de búsqueda aprox. 10^{40} nodos)

- Go: $250^{??}$

Decisiones Imperfectas

- Alternativa:
  - buscar sólo parte del árbol
    - requiere un test de corte para determinar cuándo parar
  - Usa función de evaluación
    - basada en heurísticas para estimar la utilidad esperada del juego a partir de una posición dada

Función de Evaluación

- Determina la eficiencia de un programa de juegos
- debe ser consistente con la función de utilidad
  - los valores para los nodos terminales deben ser los mismos
- compromiso entre precisión y costo en tiempo
  - sin límites de tiempo, puede usarse minimax

Función de Evaluación

- debería reflejar las chances reales de ganar

- con frecuencia usa funciones lineales ponderadas $E = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$
  - combinación de características, ponderadas de acuerdo a su relevancia

Ejemplo: Juego del gato

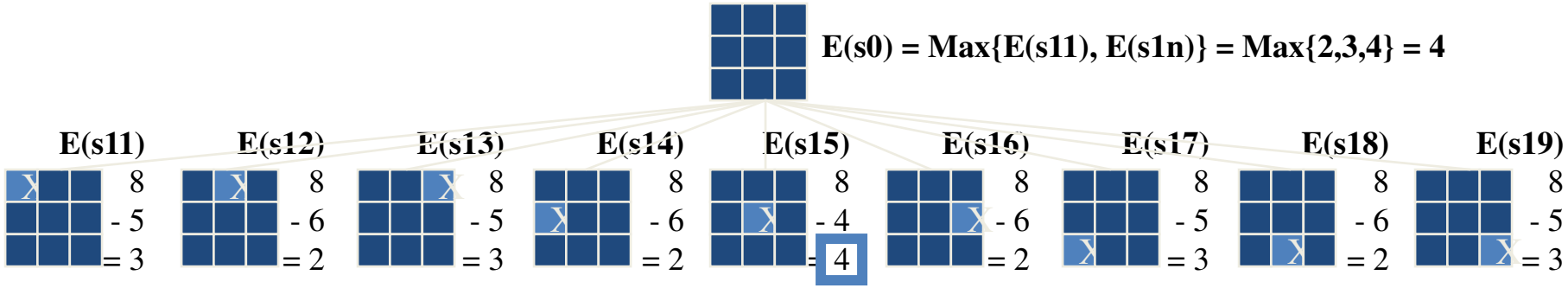
- Función de evaluación simple

```
E(s) = (fx + cx + dx) - (fo + co + do)
```

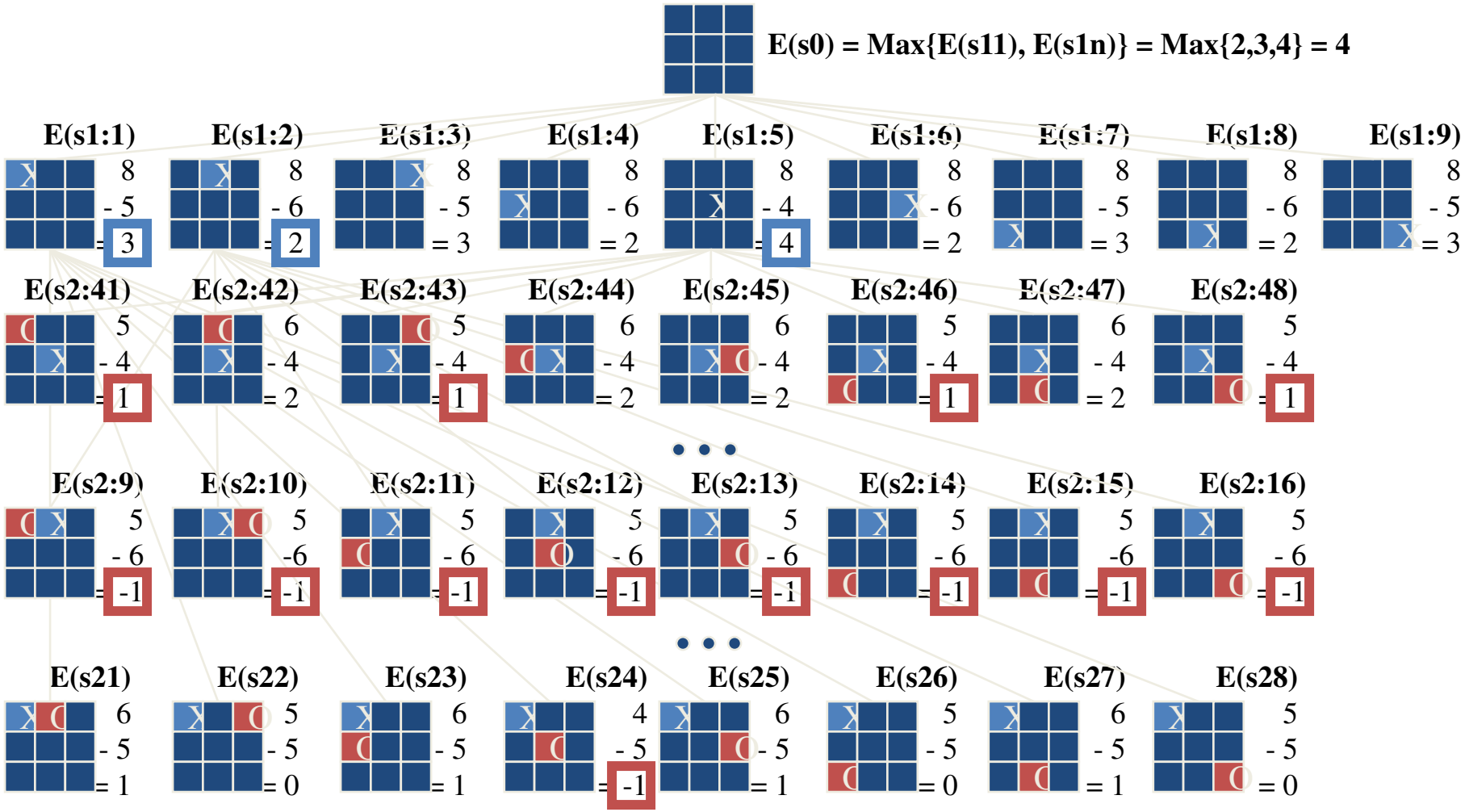
donde f,c,d son los números de las filas, columnas y diagonales aún disponibles; x y o son las piezas de los dos jugadores

- 1-jugador lookahead
  - inicia en el tope del árbol
  - evalúa las 9 elecciones para el jugador 1
  - elige el valor máximo de E
- 2-jugadores lookahead
  - también mira posibles movimientos del oponente
    - suponiendo que el oponente elige el mínimo valor para E

Juego del gato 1-Jugador



Juego del Gato 2-Jugadores



Límites de Búsqueda

- La búsqueda debe ser acotada a causa de las limitaciones de tiempo y espacio

- pueden usarse estrategias de búsqueda como profundidad limitada o profundización iterativa
  - no sacan partido del conocimiento sobre el problema

- estrategias más refinadas usan conocimiento de respaldo

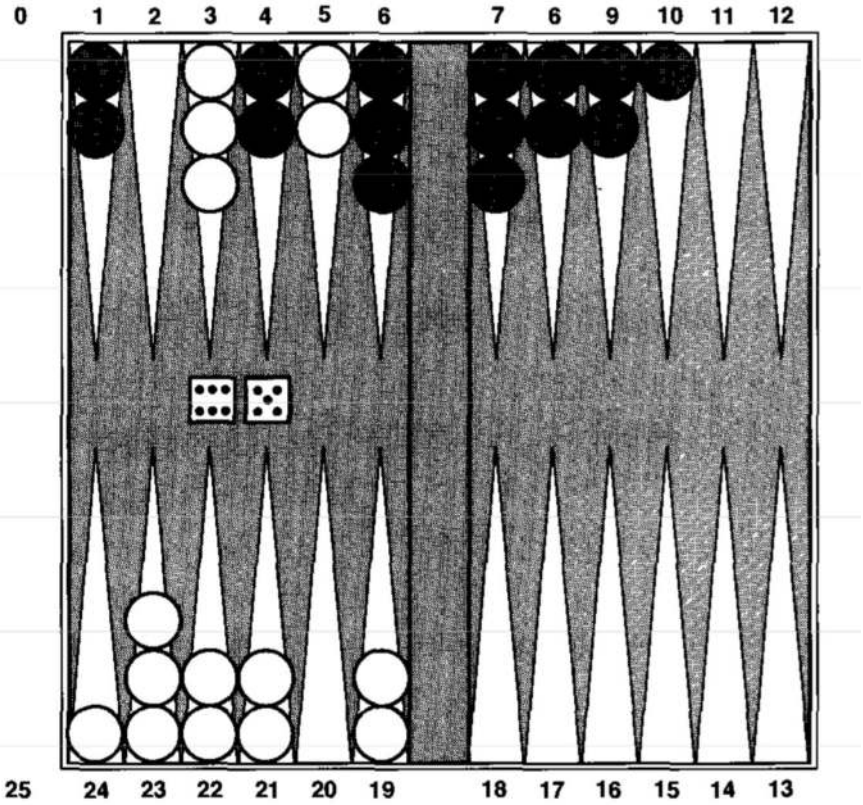
El Problema del Horizonte

- Movimientos pueden tener consecuencias desastrozas en el futuro, pero no son visibles
  - los cambios en la evaluación no serán evidentes hasta en niveles más profundos
    - ellos están "detrás del horizonte"

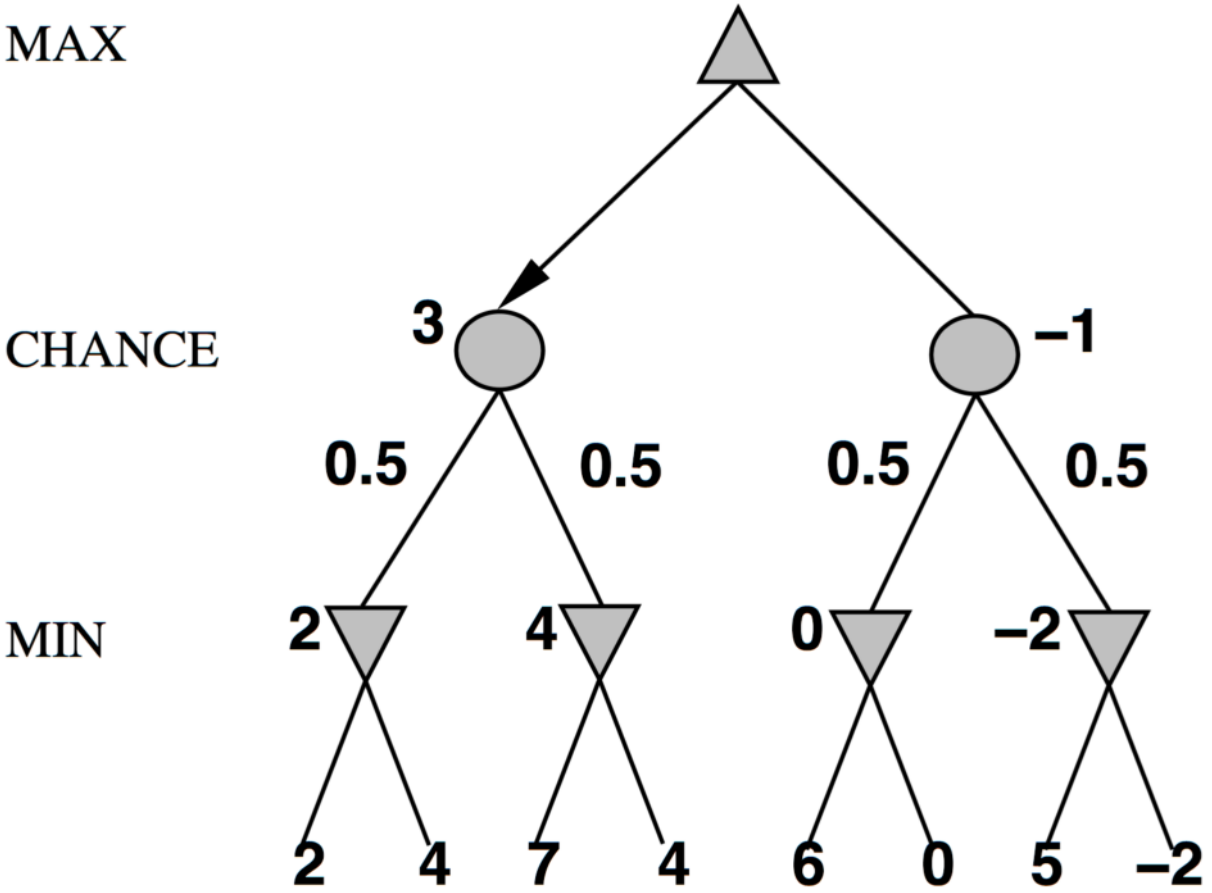
- Qué horizonte usar?
  - problema abierto aún, sin una solución general
  - Sólo aproximaciones pragmáticas restringidas a juegos o situaciones especificas

Juegos estocásticos

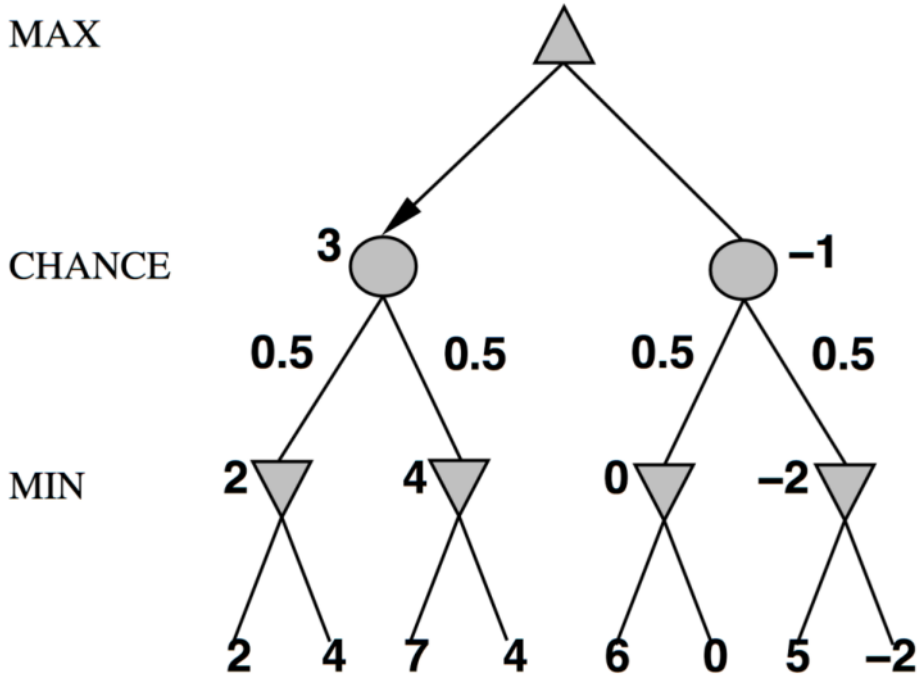
- Juegos donde hay elementos aleatorios
  - Juegos con dados
  - Buen desempeño: mezcla de habilidad y suerte.



Ejemplo



Expectiminimax



$$\text{EXPECTIMINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \text{if } n \text{ is a chance node} \end{cases}$$

- Elementos aleatorios afectan la complejidad
- Ej: Backgammon
  - b aprox. 20
  - Posibles combinaciones de 2 dados: 21
  - A profundidad 4: $1.2*10^9$!
  - TD-gammon d=2 + muy buena función de evaluación.
- A mayor profundidad, probabilidad de alcanzar un nodo disminuye
- Podemos utilizar poda alfa-beta?

Estado del arte en juegos

- Damas: Chinook (usando poda alfa-beta) venció al campeón humano en 1994.
  - Desde 2007, Chinook juega a la perfección
    - Además usa base de datos de 37 billones de jugadas finales.
- Ajedrez: Deep Blue (de IBM) venció a campeón humano en 1997
  - Analizaba 30 mil millones de jugadas por cada movimiento, alcanzando profundidad 14.
  - Función de evaluación de +8000 elementos

Estado del arte en juegos

- Backgammon: TD-Gammon, competitivo con mejores jugadores, usa:
  - Aprendizaje por refuerzo
  - Redes neuronales
- Go: AlphaGo venció a campeón humano en 2016. Usa:
  - Búsqueda en árbol basada en método Monte Carlo
  - Aprendizaje profundo

Estado del arte en juegos

- Starcraft?
- DOTA?