INTELIGENCIA ARTIFICIAL PARA JUEGOS

SESIÓN 4

Dr. Edwin Villanueva Talavera

Contenido

- Búsqueda Adversarial
  - Minimax
  - Alpha-Beta
  - Búsqueda Adversarial en Tiempo Real
  - Búsqueda Adversarial Estocástica

Bibliografía:

Capitulo 5.1-5.3 del libro:

Stuart Russell & Peter Norvig "Artificial Intelligence: A modern Approach", Prentice Hall, Third Edition, 2010

Búsqueda Adversarial

Decisiones Óptimas en Juegos (2 jugadores): MINIMAX

- Dado un árbol de juego, la estrategia óptima puede ser determinada a partir del **valor Minimax de cada nodo:**

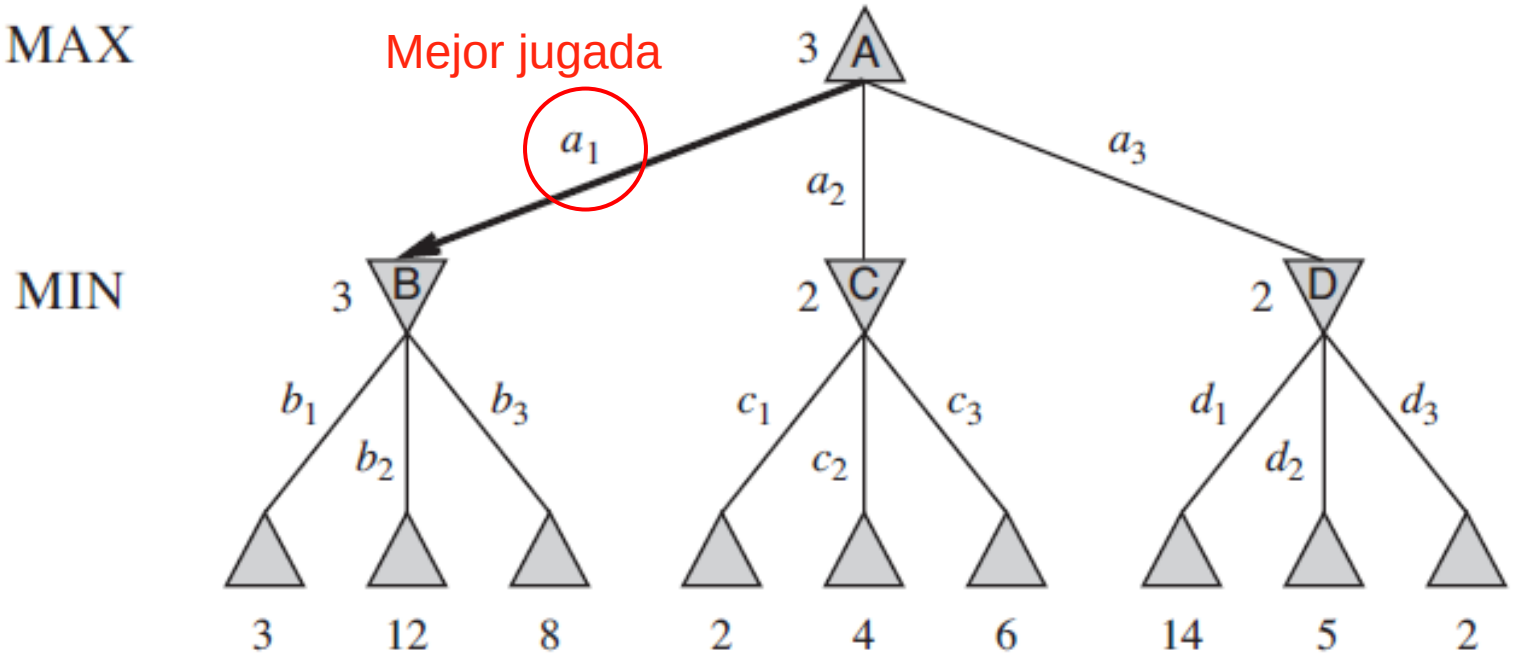
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

- El valor Minimax(s) (**para MAX**) es la utilidad de MAX de estar en estado s asumiendo que MIN escogerá los estados más ventajosos para el mismo hasta el final del juego (es decir, los estados con menor valor de utilidad para MAX)

Búsqueda Adversarial

Ejemplo de cálculo del valor de MINIMAX:

- Cada jugador hace un solo movimiento



Búsqueda Adversarial

Algoritmo MINIMAX:

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Búsqueda Adversarial

Propiedades del algoritmo MINIMAX:

- Equivale a una búsqueda completa en profundidad en el árbol del juego.
  - m: profundidad máxima del árbol
  - b: movimientos válidos en cada estado

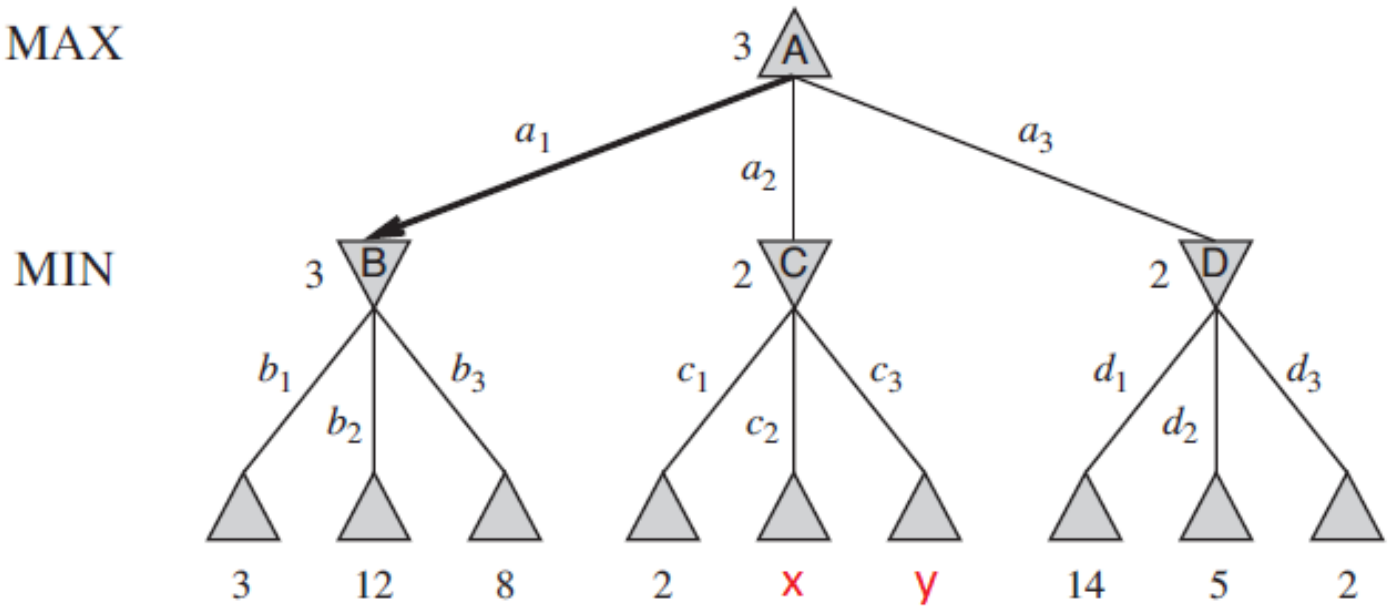
- Completo? Si (Si el árbol es finito)
- Óptimo? Si (contra un oponente óptimo)
- Complejidad de tiempo? $O(b^m)$
- Complejidad de espacio? O(bm)

Para ajedrez, b ≈ 35, m ≈100 para juegos "razonables"
→ solución exacta no es posible

Búsqueda Adversarial

Poda α-β :

- Surge como una forma de aliviar la complejidad temporal de MINIMAX, la cual es exponencial en la profundidad del árbol
- Se basa en el hecho de que decisiones optimas pueden ser tomadas evitando explorar ramas que no influirían en la decisión final
- Ejemplo:

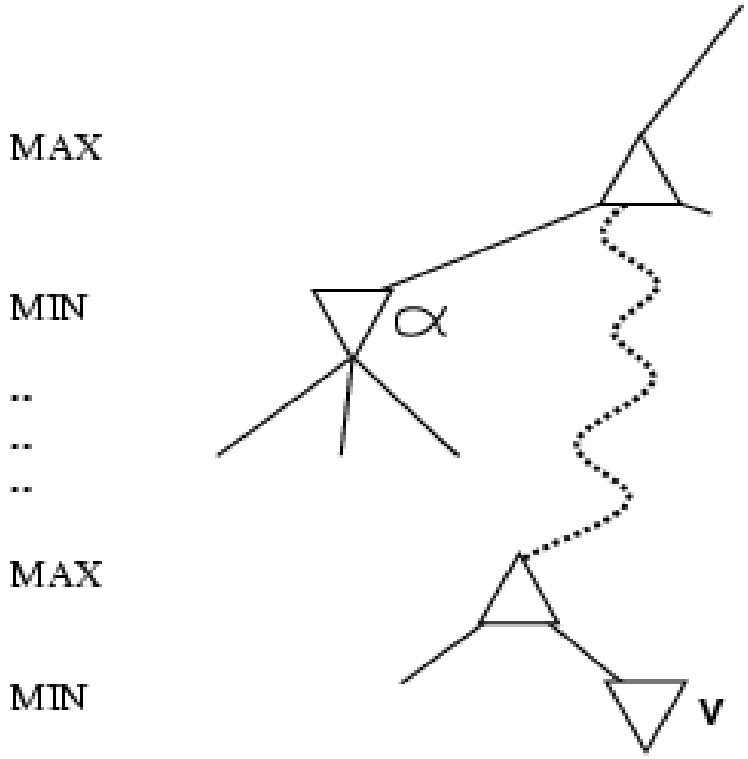


- No se necesita saber los valores de x, y para tomar la decisión en A

Búsqueda Adversarial

Implementación Poda α-β :

- α es el valor de la mejor opción (valor mas alto) encontrado hasta ahora para cualquier punto de elección de MAX;
- Si *v* es peor que α, MAX no recorrerá ese camino (podará esa rama de la búsqueda)
- β es definida de forma análoga



Búsqueda Adversarial

Algoritmo ALPHA-BETA:

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Búsqueda Adversarial

Orden de examinación de nodos en Poda α-β :

- La efectividad de la poda α-β depende del orden en que los sucesores son examinados
- Con el mejor orden posible la complejidad de tiempo es $O(b^{m/2})$ (puede resolver um arbol del doble de profundidad al mismo costo)
- Usando orden aleatoria se espera explorar $O(b^{3m/4})$
- Usando conocimiento a priori de juegos pasados puede ayudar a escoger un orden (ej. Primero capturar pieza, luego amenazar y luego avanzar)
- Esquemas con tablas de transposición pueden mejorar el costo en arboles con estados repetidos (en juegos donde diferentes secuencias de movidas pueden generar el mismo estado). Similar a Explored Set.

Búsqueda Adversarial en Tiempo Real

- Minimax y ALPHA-BETA desarrollan el árbol completo del juego, o una parte significativa del mismo, lo cual inviable en problemas grandes
- Una alternativa es cortar la búsqueda hasta una profundidad máxima y aplicar una función de evaluación (heurística)

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

Búsqueda Adversarial en Tiempo Real

Función de evaluación:

- Retorna la utilidad del estado del juego
- Debe ordenar los estados terminales en el mismo orden que la utilidad verdadera. En los estados no terminales debe ser proporcional a las chances verdaderas de ganar
- No debe implicar alto costo computacional
- Mayoria de funciones de utilidad se basan en features
  - Ej. en Ajedrez : numero de peones negros, blancos, alfiles, etc.
- Frecuentemente se usa funciones lineales de los features (*fi*) como funciones de evaluación:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

Búsqueda Adversarial en Tiempo Real

Función de evaluación:

- En muchos juegos las funciones lineales son limitadas. Se hace necesario funciones no lineales y dinámicas
  - Ej. en Ajedrez: el alfil vale mas al final del juego, cuando hay menos piezas. Dos alfiles valen mas que la suma de cada uno por separado
- La función de evaluación puede ser aprendida de un conjunto de datos de juegos usando aprendizaje de máquina

Búsqueda Adversarial en Tiempo Real

Poda com función de evaluación:

- ALPHA-BETA puede ser adaptado para trabajar con función de evaluación. Se substituye las líneas que mencionan *Terminal-Test*(state) por las siguientes:

$$\textbf{if } \textsc{Cutoff-Test}(state, depth) \textbf{ then return } \textsc{Eval}(state)$$

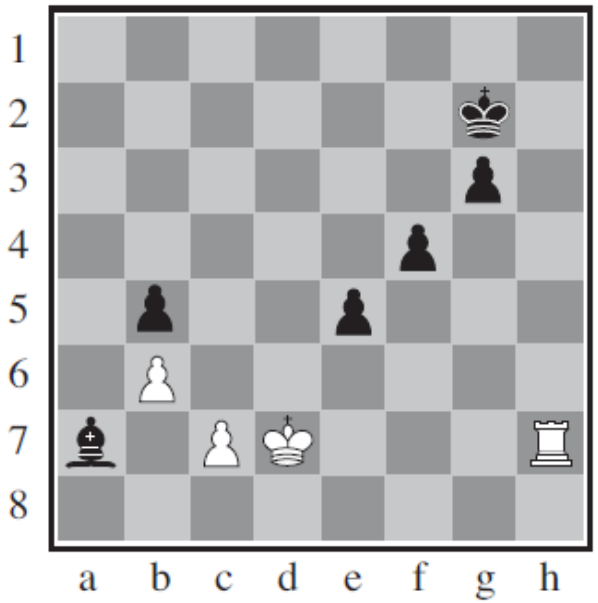
  - *Cutoff-Test* retorna true para todas las profundidades > depth y para estados terminales

- Uma forma robusta de usar poda con funcion de evaluacion es usar Busqueda em Profundidad com profundizacion iterativa (IDS)
- La profundidad máxima es seleccionada de acuerdo al tiempo de respuesta máximo permitido

Búsqueda Adversarial en Tiempo Real

Poda con función de evaluación: Problema del horizonte

- El problema del horizonte aparece cuando el agente se encara a una movida de oponente que causa serio daño y que es inevitable pero que el agente puede realizar movidas dilatorias para evitarla, causando un mayor daño al final. Ej.

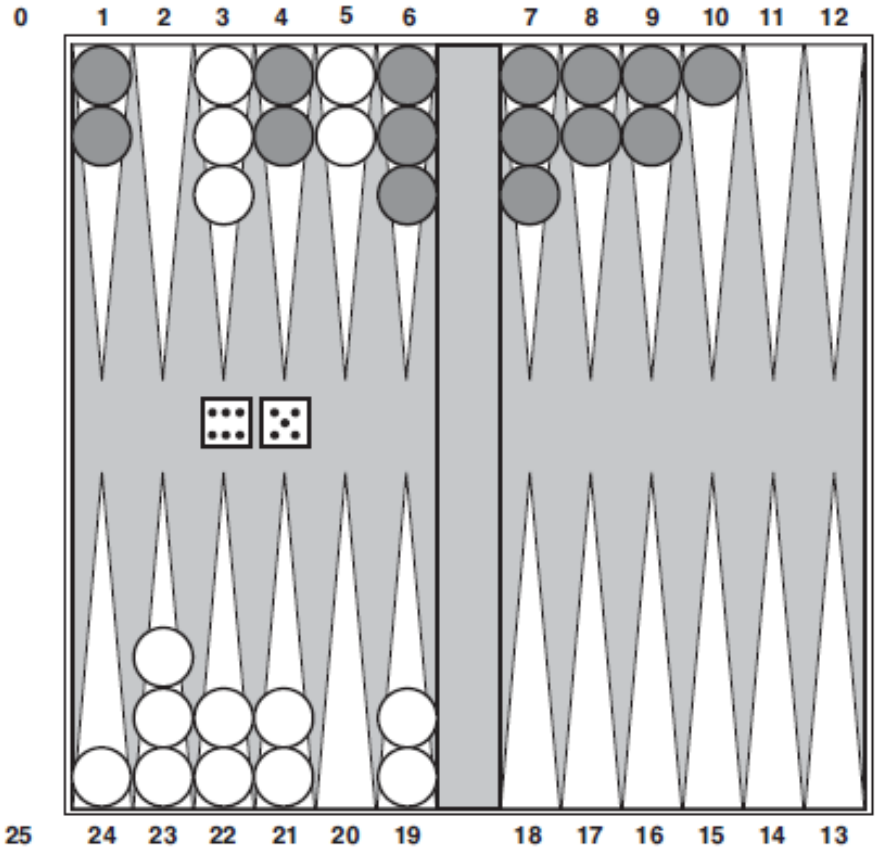


- Si el horizonte es 7 plys, el agente que controla las negras pensaria que haciendo dos veces jaque con los peones e5 y f4 es un buen negocio para salvar el alfil, ya que la perdida de este desaparece del horizonte

- Una forma de aliviar es usar **extensión singula**r: avanzar el arbol para aquellas movidas prometedoras

Búsqueda Adversarial Estocástica

- Varios juegos incluyen elementos estocásticos, como el hecho de tirar dados para saber que movidas se dispone. Estos entornos se llaman juegos estocásticos



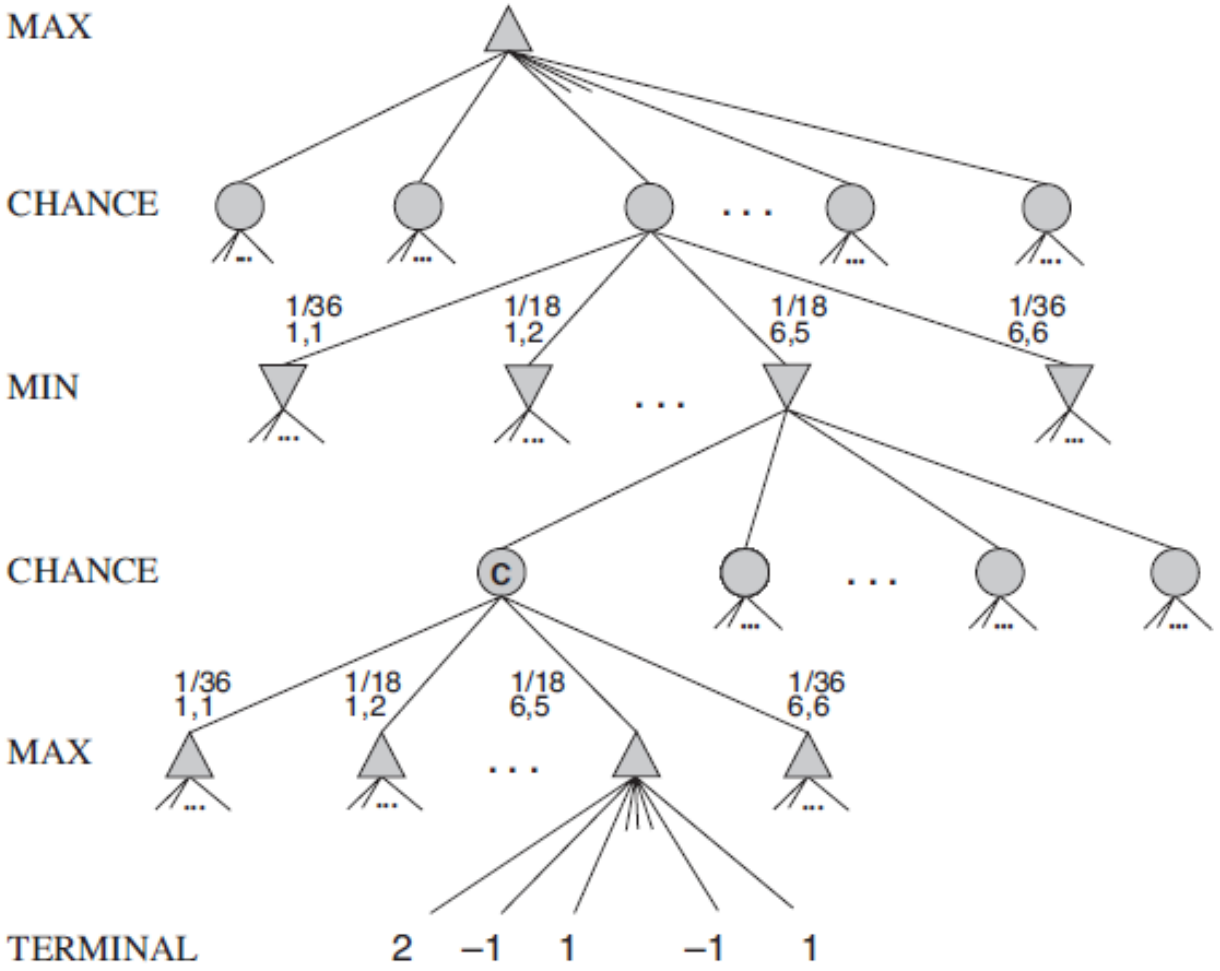
- Backgammon

Búsqueda Adversarial Estocástica

- El agente que controla un jugador (blanco por ejemplo) no sabe cuales son las movidas validas del oponente y por lo tanto no puede desarrollar un árbol de juego estandad
- Em juegos estocásticos se incluye nodos chance, en adición a los nodos MAX y MIN
- Los arcos que salen de cada nodo chance denota los posibles eventos aleatório (ej. numeros de dados) y sus probabilidades
  - Para el caso de dos dados serian 21 eventos distintos. Los eventos con numeros iguales tienen probabilidad 1/36. Los otros eventos (15) tienen probabilidad 1/18

Búsqueda Adversarial Estocástica

- Ejemplo de arbol de juego estocástico em Backgammon



Búsqueda Adversarial Estocástica

EXPECTIMINIMAX

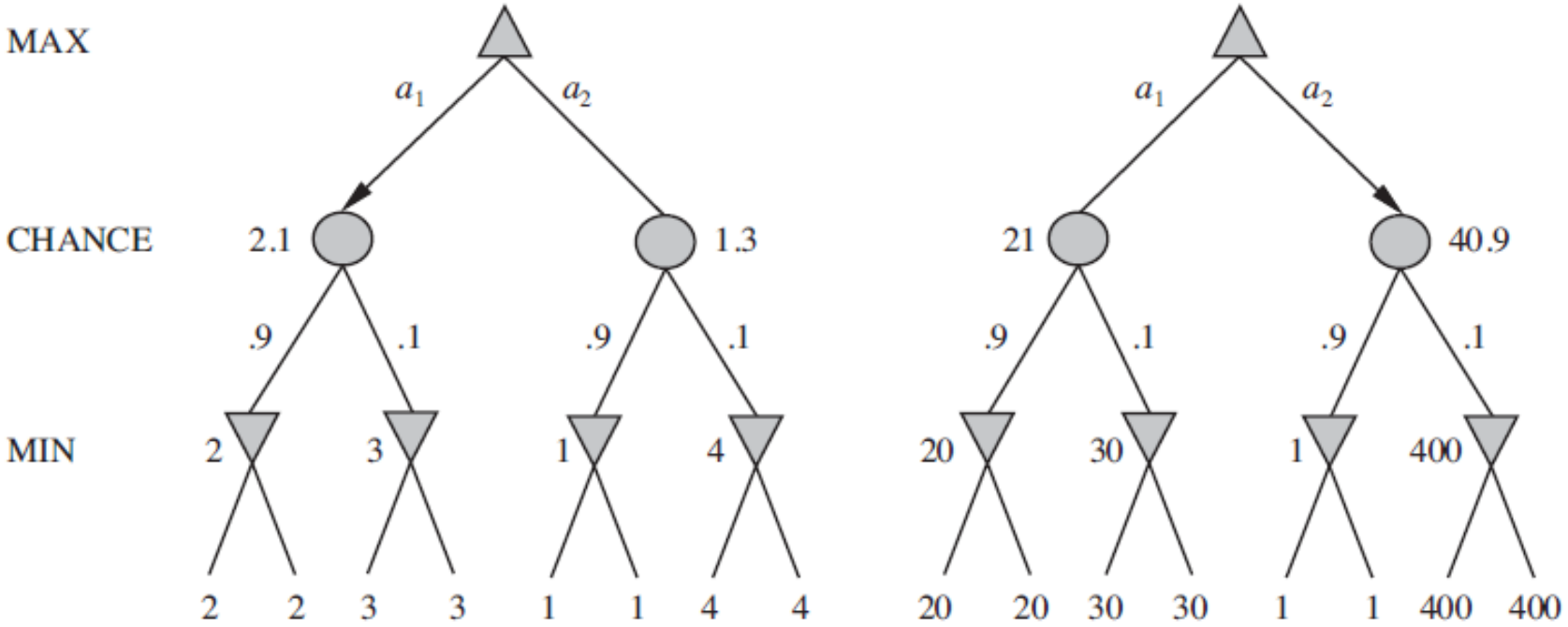
- Dado que los estados no tienen valores Minimax definidos. En lugar se calcula el valor esperado de la utilidad de un estado, esto es, el valor medio de todos los posibles resultados de los nodos chance.
- Así, la búsqueda Expectiminimax es una generalización de Minimax para lidiar con nodos chance. Expectiminimax calcula la utilidad de cada estado s como :

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

Búsqueda Adversarial Estocástica

Funcion de evaluacion en juegos estocásticos

- Dado que los valores de utilidades son promedios de valores, se tiene que tener cuidado con la escala de valores de la funcion de evaluación. Se puede tener un comportamiento bien diferente si se cambia la escala de los valores de la funcion



- La funcion de evaluación debe dar valores proporcionales a la probabilidad de ganar (utilidad)

Búsqueda Adversarial Estocástica

Poda en juegos estocásticos

- En un juego sin nodos chance el costo computacional en tiempo de la busqueda Minimax es $O(b^m)$, donde b es el factor de ramificacion y m la maxima profundidad del abol del juego.
- En um juego con nodos chance, el costo es: $O(b^m n^m)$, donde n es el numero de distintos eventos aleatorios
- En backgammon n= 21, b= 20. Probablemente solo sea manejable llegar a 3 plys
- La estocasticidad hace crecer las posibilidades enormemente. No es facil podar nodos chance: Se puede imponer limites a los valores de utilidad y asi hacer la poda de posibles nodos hijos que no afectarian los limites:
- Alternativas: Montecarlo search, Aprendizaje por refuerzo

Preguntas?