Búsqueda Entre Adversarios

Sistemas Inteligentes

Dr. Víctor de la Cueva
vcueva@itesm.mx

Entornos multiagentes

- En los entornos multiagentes, cualquier agente debe considerar las acciones de los otros agentes y cómo afectan a su propio bienestar.
- La imprevisibilidad de estos agentes puede introducir muchas posibles contingencias en el proceso de resolución de problemas del agente.
- Los entornos multiagente pueden ser cooperativos o competitivos.

Entornos competitivos

- Los entornos competitivos, en los cuales los objetivos de los agentes están en conflicto, dan lugar a un tipo especial de problemas de búsqueda llamados búsqueda entre adversarios, a menudo conocidos como juegos.
- NOTA: Los entornos con muchos agentes se estudian mejor como entornos económicos más que como juegos. La teoría de juegos es una rama de la Economía.

Juegos de suma cero

- En la IA los juegos son, por lo general, una clase más especializada, que los teóricos de juegos llaman juegos de suma cero:
  - Dos jugadores
  - Tiran por turnos
  - Determinista
  - De información completa

Nivel de juego de un programa

- El nivel ha ido en ascenso desde el inicio:
  - Damas, Otelo, han superado a la gente.
  - Texas Hold'em Poker, juegan a un nivel de maestro.
  - Ajedrez (10-feb-1996) y Backgammon (1992), ambos de IBM, han derrotado campeones.
  - Go, estuvo a nivel de aficionado por años y finalmente en Enero de 2016 logró derrotar a un campeón).

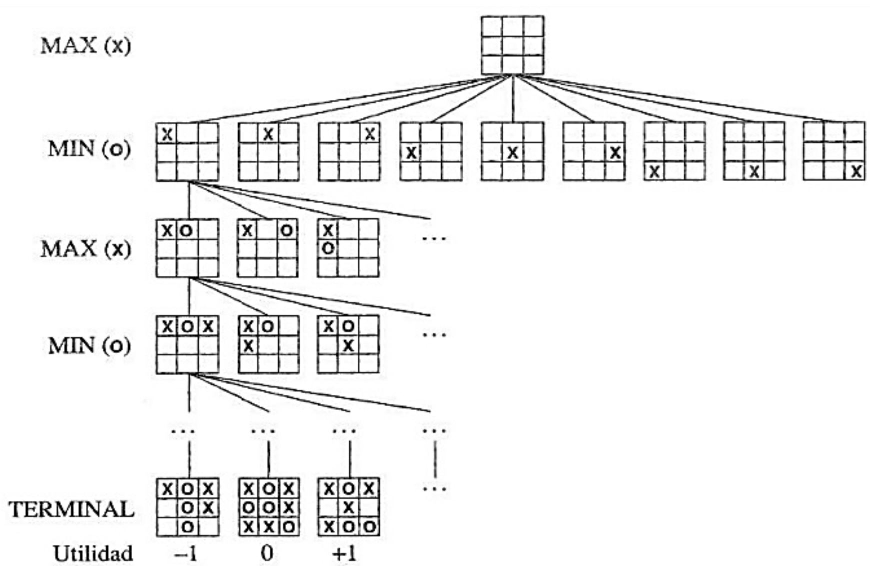
Problemas difíciles

- Los juegos son interesantes porque son problemas difíciles de resolver y requieren inteligencia.
- Al mismo tiempo, se pueden modelar muy bien en los programas de computadora.
- Sin embargo, con los métodos conocidos hasta ahora (2016) la ineficiencia se castiga con severidad:
  - Ajedrez: factor de ramificación ≈ 35, movimientos por jugador ≈ 50. Su árbol de búsqueda tiene 35^{100} o 10^{154} nodos (10^{78} – 10^{80} átomos en el universo conocido).
  - Go: factor de ramificación = 250, movimientos por jugador ≈ 200.

Decisiones óptimas en juegos

- Consideremos juegos con 2 jugadores que llamaremos MAX y MIN.
- MAX tira primero y luego se mueven por turnos hasta que el juego termina.
- Un juego es una clase de problema de búsqueda con los siguientes elementos:
  - Estado inicial (e.g. posición del tablero y jugador que mueve)
  - Función sucesor
  - Test terminal (estados terminales)
  - Función de utilidad: valor numérico a los estados terminales (e.g. ajedrez = +1, 0, -1, Backgammon = de +192 a -192).

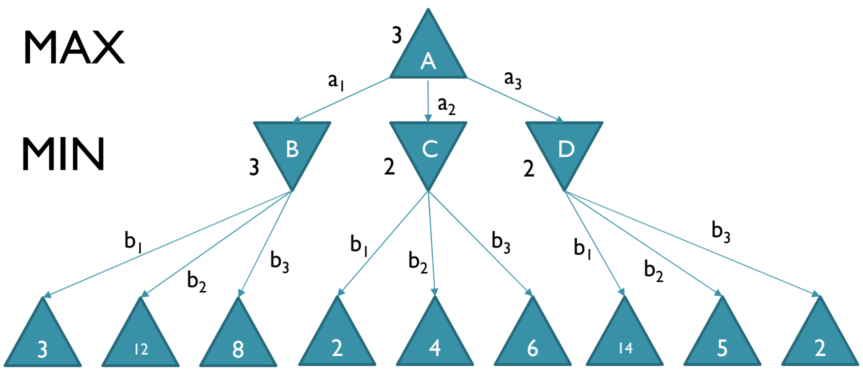
Árbol para el Gato (Tic-Tac-Toe)



Grandes árboles

- El juego más simple tiene un árbol muy complidado:

MAX: a_1, a_2, a_3 MIN: b_1, b_2, b_3



Valor Minimax

- Considerando un árbol del juegos, la estrategia óptima puede determinarse examinando el valor minimax de cada nodo con la función: VALORMINIMAX(N).
- El valor minimax de un nodo es la utilidad (para MAX) de estar en el estado correspondiente, asumiendo que ambos jugadores juegan <u>óptimamente</u> desde allí al final del juego.
- El valor minimax de un estado terminal es sólo la utilidad.
- MAX preferirá moverse a un nodo de valor máximo, mientras que MIN a uno de valor mínimo (de ahí su nombre).

La función VALORMINIMAX(N)

$$ValorMinimax(n) = \begin{cases} Utilidad(n) & si\ n\ es\ un\ nodo\ terminal \\ \max_{s \in Sucesores(n)} \{ValorMinimax(s)\} & si\ n\ es\ un\ estado\ MAX \\ \min_{s \in Sucesores(n)} \{ValorMinimax(s)\} & si\ n\ es\ un\ estado\ MIN \end{cases}$$

Minimax

function MINIMAX-DECISION(*state*) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(*state*, *a*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ 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*)
  $v \leftarrow \infty$
  for each *a* **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

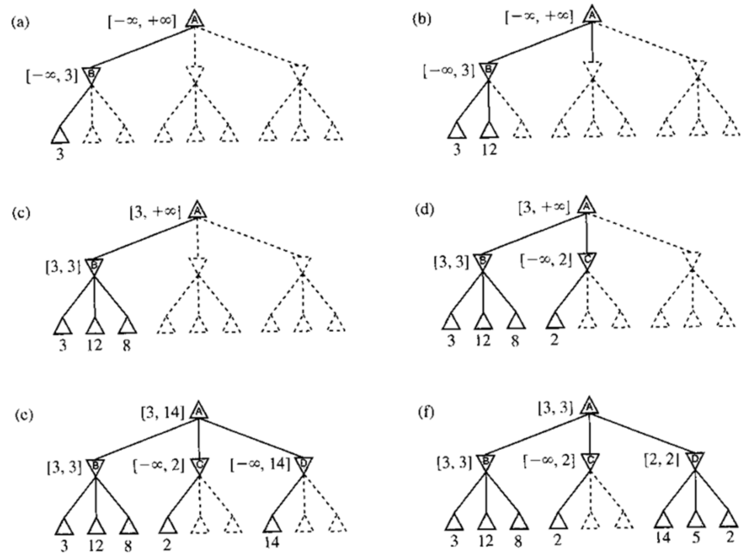
Minimax con poda alfa-beta

- Como muchos de los algoritmo de búsqueda en arboles, el Minimax puede ser muy costoso, por lo que si se puede ahorrar algo se debe hacer.
- Una forma de ahorro es llevar un control de las evaluaciones y parar la búsqueda en todos los hijos del nodo actual cuando se esté seguro de que no se puede mejorar.
- Esta versión del algoritmo Minimax se llama con poda alfa-beta.
- Poda alfa-beta regresa el mismo valor que Minimax.

Alfa y beta

- Alfa: el valor de la mejor (es decir, el valor más alto) opción que hemos encontrado hasta el momento en cualquier punto a lo largo del camino para MAX.
- Beta: El valor de la mejor (es decir, el valor más bajo) opción que hemos encontrado hasta el momento en cualquier punto a lo largo del camino MIN.
- La búsqueda alfa-beta actualiza los valores de alfa y beta conforme se avanza y poda las ramas restantes en cuanto sepa que el valor actual es peor que el valor actual de alfa o beta para MAX o MIN, respectivamente.

Ejemplo árbol minimax con poda alfa-beta



Minimax con poda alfa-beta

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  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*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Referencia

- S. Russel and P. Norvig. Inteligencia Artificial un enfoque moderno. 2ª edición, Pearson, España (2004).