Resolución de problemas

Búsquedas con adversarios

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

3 de julio de 2021

Búsquedas con adversarios

Búsquedas con adversarios

- Se consideran ahora ambientes **competitivos** donde las metas de los agentes están en conflicto.
- A los problemas donde agentes adversarios buscan la mejor solución para cada uno se les conoce como *juegos*.
- Consideraremos juegos **deterministas**, **completamente observables**, donde los agentes actúan por **turnos**. Por ejemplo:
  - Gato
  - Damas chinas
  - Damas españolas
  - Ajedrez
- A la definición de problema de planeación se agregan dos elementos:

  Jugador(s) Define el **turno**, a qué jugador le toca realizar una acción.

  Utilidad(s,p) Es una **función de utilidad** que indica el valor final del juego en el estado terminal s para el jugador p.

Minimax

Minimax

- El objetivo del algoritmo es encontrar una *estrategia óptima*, tal que obtenga el mejor resultado posible cuando se juega contra un **oponente infalible**.
- Se define para un juego con dos participantes:

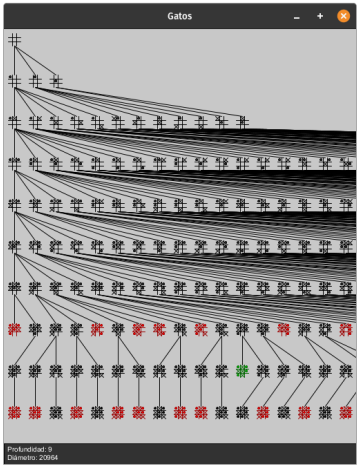
  Max Es el primero en jugar y su objetivo es obtener valores de utilidad lo más **altos** posible.

  Min Juega en el turno siguiente y su objetivo es obtener valores lo más **bajos** que pueda.

Minimax

Algoritmo 1 Minimax

1: Cada hoja en el árbol del juego recibe su valor de utilidad.
2: **for all** nodo padre, desde las hojas hasta la raíz **do**
3: **if** nodo es MAX **then**
4: Asignar al nodo el **mayor** valor de entre los valores de sus hijos.
5: **else if** nodo es MIN **then**
6: Asignar al nodo el **menor** valor de entre los valores de sus hijos.



Ejercicio minimax

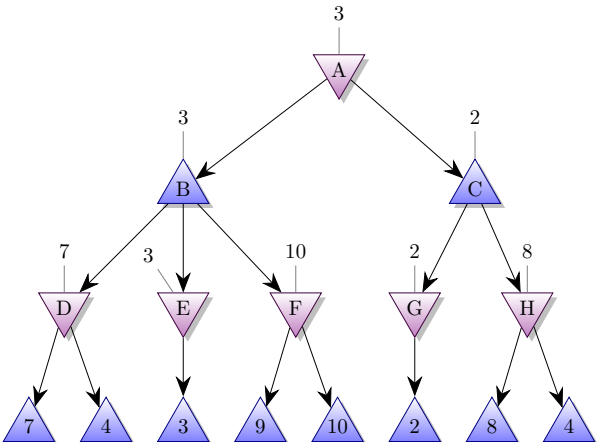


Figura: Ejecución del algoritmo Minimax

Poda $\alpha - \beta$

Principios

- Es posible *podar* ciertas ramas del árbol de búsqueda si asumimos que el otro jugador realizará tiradas óptimas.
- Solamente se explorará lo suficiente hasta saber qué tan buena puede ser la jugada para el oponente, **si es demasiado buena, no se elegirá la acción que lleva a ese subárbol** a esto se le llama *podar* el subárbol, por lo que ya no es necesario explorar las ramas siguientes.

Algoritmo 2 Poda $\alpha - \beta$

```
1: function PODA α − β(estado)
2:     v ← MAX-VALOR(estado, −∞, +∞)
3:     return v.acción()
```

Poda $\alpha - \beta$

Algoritmo 3 Las acciones de Max

```
1: function MAX-VALOR(estado, α, β)
2:     if estado.esMeta() then return estado.utilidad()
3:     v ← −∞
4:     for a ∈ accionesAplicables(estado) do
5:         estado′ ← aplica(estado, a)
6:         v ← MÁXIMO(v,MIN-VALOR(estado′, α, β))
7:         if v ⩾ β then return v
8:         α ← MÁXIMO(α, v)
9:     return v
```

Poda $\alpha - \beta$

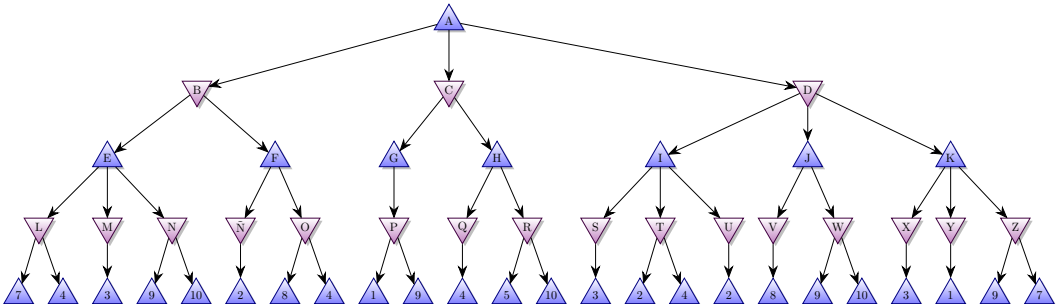
Algoritmo 4 Las acciones de Min

```
1: function MIN-VALOR(estado, α, β)
2:     if estado.esMeta() then return estado.utilidad()
3:     v ← ∞
4:     for a ∈ accionesAplicables(estado) do
5:         estado′ ← aplica(estado, a)
6:         v ← MÍNIMO(v,MAX-VALOR(estado′, α, β))
7:         if v ⩽ α then return v
8:         β ← MÍNIMO(β, v)
9:     return v
```

Ejecución de poda $\alpha - \beta$

- α es una cota inferior, la asigna **Max**.
  - Todos los hijos de **Max** reciben el mismo valor de β.
  - β se usa para saber cuándo **Max** ya no debe seguir revisando a sus hijos.
- β es una cota superior, la asigna **Min**.
  - Todos los hijos de **Min** reciben el mismo valor de α.
  - α se usa para saber cuándo **Min** ya no debe seguir revisando a sus hijos.
- ν contiene el valor del hijo óptimo, de entre los que han sido visitados.
  - Éste es el valor de regreso.

Ejercicio $\alpha - \beta$



Referencias I


Russell, Stuart y Peter Norving (2010). *Artificial Intelligence, A Modern Approach*. Ed. por Michael Hirsch. 2a. Pearson Prentice Hall.

Licencia