Búsqueda con adversarios

Ya habíamos hablado de los entornos multiagente, en los cuales cualquier agente tendrá que considerar las acciones de otros agentes y cómo afectan a su propio bienestar. Los entornos multiagentes pueden ser cooperativos o competitivos. Los entornos competitivos, en los cuales los objetivos del agente están en conflicto, dan ocasión a problemas de búsqueda entre adversarios, a menudo conocidos como juegos.

La teoría matemática de juegos, ve a cualquier entorno multiagente como un juego a condición de que el impacto de cada agente sobre los demás sea significativo, sin tener en cuenta si los agentes cooperativos o competitivos. En IA los juegos son, por lo general, una clase más especializada (que los teóricos de juegos llaman juegos de suma cero, de dos jugadores, por turnos, determinista, o de información perfecta). En nuestra terminología, significan entornos deterministas, totalmente observables en los cuales hay dos agentes cuyas acciones deben alternar y en los que los valores de utilidad, al final del juego, son siempre iguales y opuestos. Por ejemplo, si un jugador gana un juego de ajedrez (+1), el otro jugador necesariamente pierde (-1).

Los juegos, a diferencia de los problemas anteriores, son interesantes porque son muy difíciles de resolver. Por ejemplo el ajedrez tiene un factor de ramificación promedio de 35, y los juegos a menudo requieren 50 movimientos por cada jugador, entonces el árbol de búsqueda tiene aproximadamente 35^{100} nodos nodos. Por lo tanto, los juegos, como el mundo real, requieren la capacidad de tomar decisiones cuando no es factible calcular la decisión óptima.

Consideraremos juegos con dos jugadores, que llamaremos MAX y MIN por motivos que pronto se harán evidentes. MAX mueve primero, y luego mueven por turno hasta que el juego se termina. Al final de juego, se conceden puntos al jugador ganador y penalizaciones al perdedor. Un juego puede definirse formalmente como una clase de problemas de búsqueda con los componentes siguientes:

- El **estado inicial**, que incluye la posición del tablero e identifica al jugador que mueve.
- Una **función sucesor**, que devuelve una lista de pares (movimiento, estado), indicando un movimiento legal y el estado que resulta.
- Un **test terminal**, que determina cuándo se termina el juego. A los estados donde el juego se ha terminado se les llaman estados terminales.
- Una **función utilidad** (también llamada función objetivo o función de rentabilidad), que da un valor num6rico a los estados terminales. En el ajedrez, el resultado es un triunfo, pérdida, o empate, con valores +1,-1 o O. Algunos juegos tienen una variedad más amplia de resultados posibles: las rentabilidades en el backgammon se extienden desde +192a -192.

El estado inicial y los movimientos legales para dacad lado definen el árbol de juegos. La Figura 6.1 muestra la parte del árbol de juegos para el tic-tac-toa (tres en raya o gato). Desde el estado inicial, MAX tiene nueve mocimientos posibles. El juego alterna entre la colocación de una C para MAX y la colocación de un O para MIN, hasta que alcancemos nodos hojas correspondientes a estados terminales, de modo que un jugador tenda tres en raya o todos los cuadrados esten llenos. El número sobre cada hoja indica el valor de utilidad del estado terminal desde el punto de vista de MAX; se supone que los valores altos son buenos para MAX y malos para MIN.

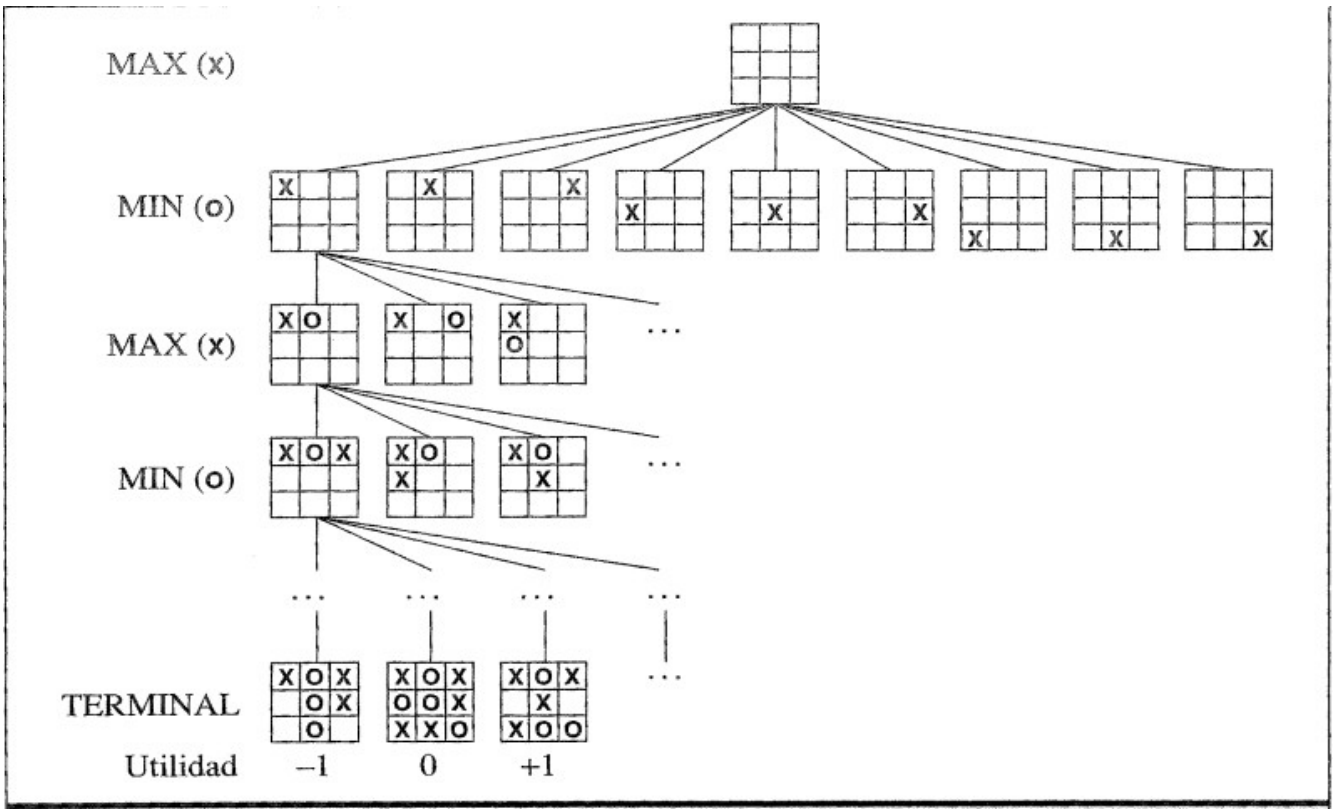


Figura 6.1 Un árbol (parcial) de búsqueda para el juego de tic-tac-toe. El nodo de arriba es el estado inicial, y MAX mueve primero, colocando un X en un cuadrado vacío. Mostramos la parte del árbol de búsqueda, con movimientos alternados por MIN (O) y MAX, hasta que finalmente alcancemos estados terminales, en los cuales podemos asignar utilidades según las reglas del juego.

Algoritmo minimax

En los juegos bipersonales de suma cero, donde cada jugador conoce de antemano la estrategia de su oponente y sus consecuencias, existe una estrategia que permite a ambos jugadores minimizar la pérdida máxima esperada. En particular, cuando se examina cada posible estrategia, un jugador debe considerar todas las respuestas posibles del jugador adversario y la pérdida máxima que puede acarrear. El jugador juega, entonces, con la estrategia que resulta en la minimización de su máxima pérdida. Tal estrategia es llamada óptima para ambos jugadores sólo en caso de que sus minimaxes sean iguales (en valor absoluto) y contrarios (en signo). Si el valor común es cero el juego se convierte en un sinsentido.

En los juegos de suma no nula, existe tanto la estrategia minimax como la maximin. La primera intenta minimizar la ganancia del rival, o sea busca que el rival tenga el peor resultado. La segunda intenta maximizar la ganancia propia, o sea busca que el jugador obtenga el mejor resultado.

Pasos del algoritmo minimax:

1. Generación del árbol de juego. Se generarán todos los nodos hasta llegar a un estado terminal.
2. Cálculo de los valores de la función de utilidad para cada nodo terminal.
3. Calcular el valor de los nodos superiores a partir del valor de los inferiores. Según nivel si es MAX o MIN se elegirán los valores mínimos y máximos representando los movimientos del jugador y del oponente, de ahí el nombre de minimax.
4. Elegir la jugada valorando los valores que han llegado al nivel superior.

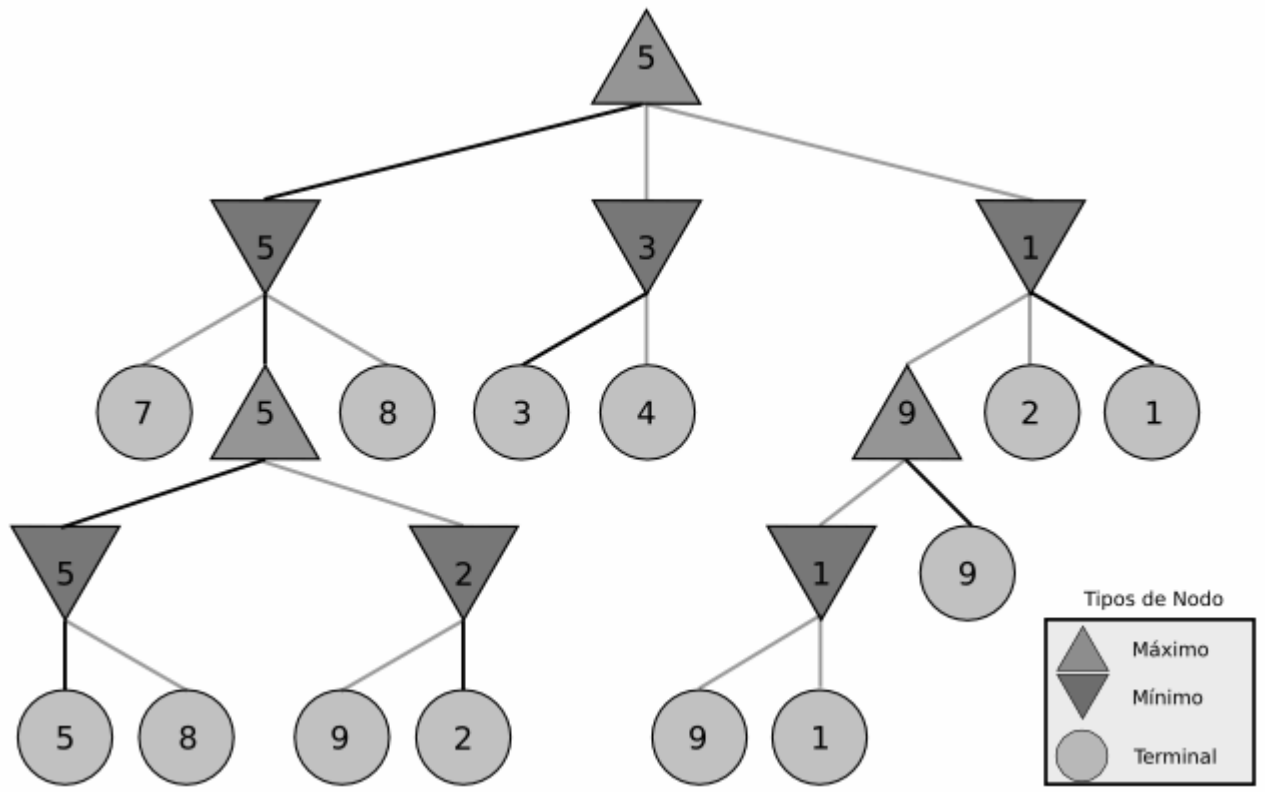
El algoritmo explorará los nodos del árbol asignándoles un valor numérico mediante una función de evaluación, empezando por los nodos terminales y subiendo hacia la raíz. La función de utilidad

definirá lo buena que es la posición para un jugador cuando la alcanza. En el caso del ajedrez los posibles valores son (+1,0,-1) que se corresponden con ganar, empatar y perder respectivamente. En el caso del backgammon los posibles valores tendrán un rango de [+192,-192], correspondiéndose con el valor de las fichas. Para cada juego pueden ser diferentes.

Si minimax se enfrenta con el dilema del prisionero escogerá siempre la opción con la cual maximiza su resultado suponiendo que el contrincante intenta minimizarlo y hacernos perder.

EJEMPLO

- Funcionamiento de Minimax en un árbol generado para un juego imaginario.
- Los posibles valores de la función de utilidad tienen un rango de [1-9].
- En los movimientos del contrincante suponemos que escogerá los movimientos que minimicen nuestra utilidad
- En nuestros movimientos suponemos que escogeremos los movimientos que maximizan nuestra utilidad.
- 1er. Paso: Calcular los nodos terminales, en verde.
- 2º. Paso: Calcular el cuarto nivel, movimiento MIN, minimizando lo elegido (5, 2 y 1).
- 3er. Paso: Calcular el tercer nivel, movimiento MAX, maximizando la utilidad (5, 9).
- El segundo nivel es un movimiento MIN(5, 3 y 1).
- Finalmente llegamos al primer nivel, el movimiento actual, elegiremos el nodo que maximize nuestra utilidad (5).



Aplicación: El Juego del Gato

- Dos jugadores **MIN** y **MAX**
- Los jugadores colocan fichas en un tablero de 3 X 3
- **MAX** usa las fichas X
- **MIN** usa las fichas O

X	O	X
O	X	O
X	O	X

¡ MAX gana !

- **Reglas:**
- Inicialmente el tablero está vacío
- MAX empieza y se alternan los movimientos

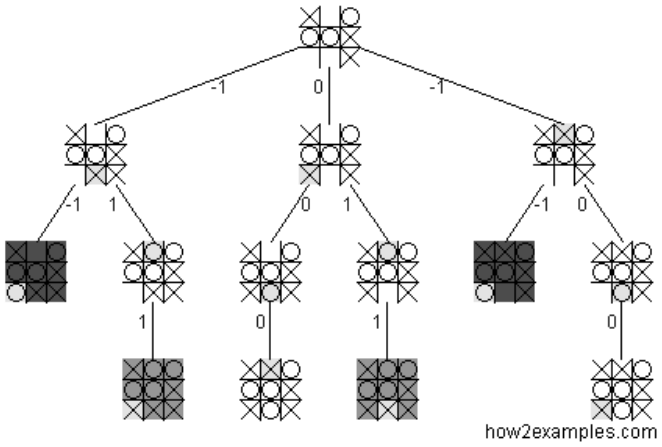
O	O	X
O	X	O
O	X	X

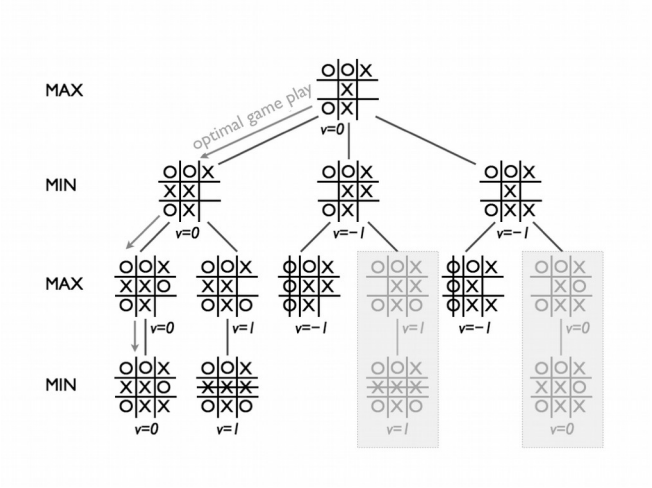
¡ MIN gana !

- MAX gana si obtiene una línea de 3 X's
- MIN gana si obtiene una línea de 3 O's
- Existe la posibilidad de empate

X	O	X
O	X	O
O	X	O

¡ Empate !





Algoritmo MINIMAX

---

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(state)
  return the *action* in SUCCESSORS(*state*) with value v

---

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

---

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