Algoritmo MiniMax en juegos multijugador

Muchos juegos permiten más de dos jugadores. Examinemos como obtener una extensión de la idea MiniMax a juegos multijugador.

Primero, tenemos que sustituir el valor para cada nodo con un vector de valores. Por ejemplo, en un juego con tres jugadores, A, B y C, un vector (V_A, V_B, V_C) asociado con cada nodo. Para los estados terminales, este vector dará la utilidad del estado desde el punto de vista de cada jugador.

Ahora tenemos que considerar los estados no terminales. Consideremos el nodo marcado con X en el árbol de juegos de la Figura 6.4. En ese estado, el jugador C elige que hacer. Las dos opciones conducen a estados terminales con los vectores de utilidad (V_A=1, V_B=2, V_C=6) y (V_A=4, V_B=2, V_C=3). Como 6 es más grande que 3, C debería elegirlo como siguiente movimiento. Esto significa que si se alcanza el estado C, el movimiento siguiente conducirá a un estado terminal con utilidades (V_A=1, V_B=2, V_C=6). De ahí, que el valor que le llega a X es este vector. En general, el valor hacia atrás de un nodo n es el vector de utilidad de cualquier sucesor que tiene el valor más alto para el jugador que elige en n.

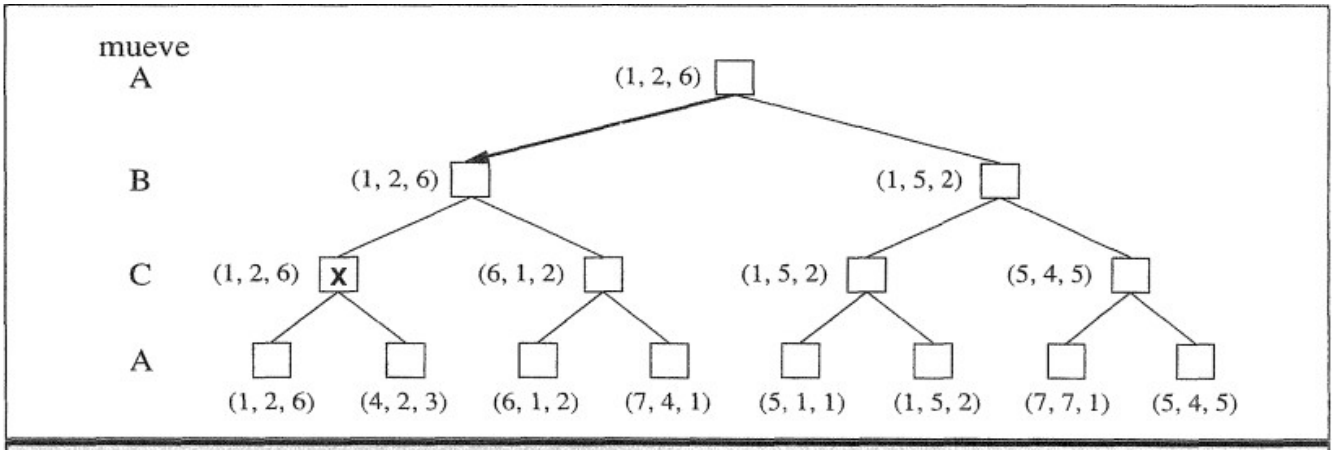


Figura 6.4 Las tres primeras capas de un árbol de juegos con tres jugadores (*A, B, C*). Cada nodo se etiqueta con valores desde el punto de vista de cada jugador. En la raíz se marca el mejor movimiento.

Poda alfa-beta

El problema de la búsqueda minimax es que el número de estados que tiene que examinar es exponencial al número de movimientos. Lamentablemente no podemos eliminar el exponente, pero podemos reducirlo, con eficacia, en la mitad. La jugada es que es posible calcular la decisión minimax correcta sin mirar todos los nodos en el árbol de juegos. La técnica que examinaremos se le llama poda alfa-beta. Cuando lo aplicamos a un árbol minimax estándar, devuelve el mismo movimiento que devolvería minimax, ya que podar las ramas no puede influir, posiblemente, en la decisión final.

Consideremos otra vez el árbol de juegos de la Figura 6.5. Vamos a realizar un calculo de decisión óptima una vez más, esta vez prestando atención a lo que sabemos en cada punto. El resultado es que podemos identificar la decisión minimax sin evaluar dos de los nodos hoja.

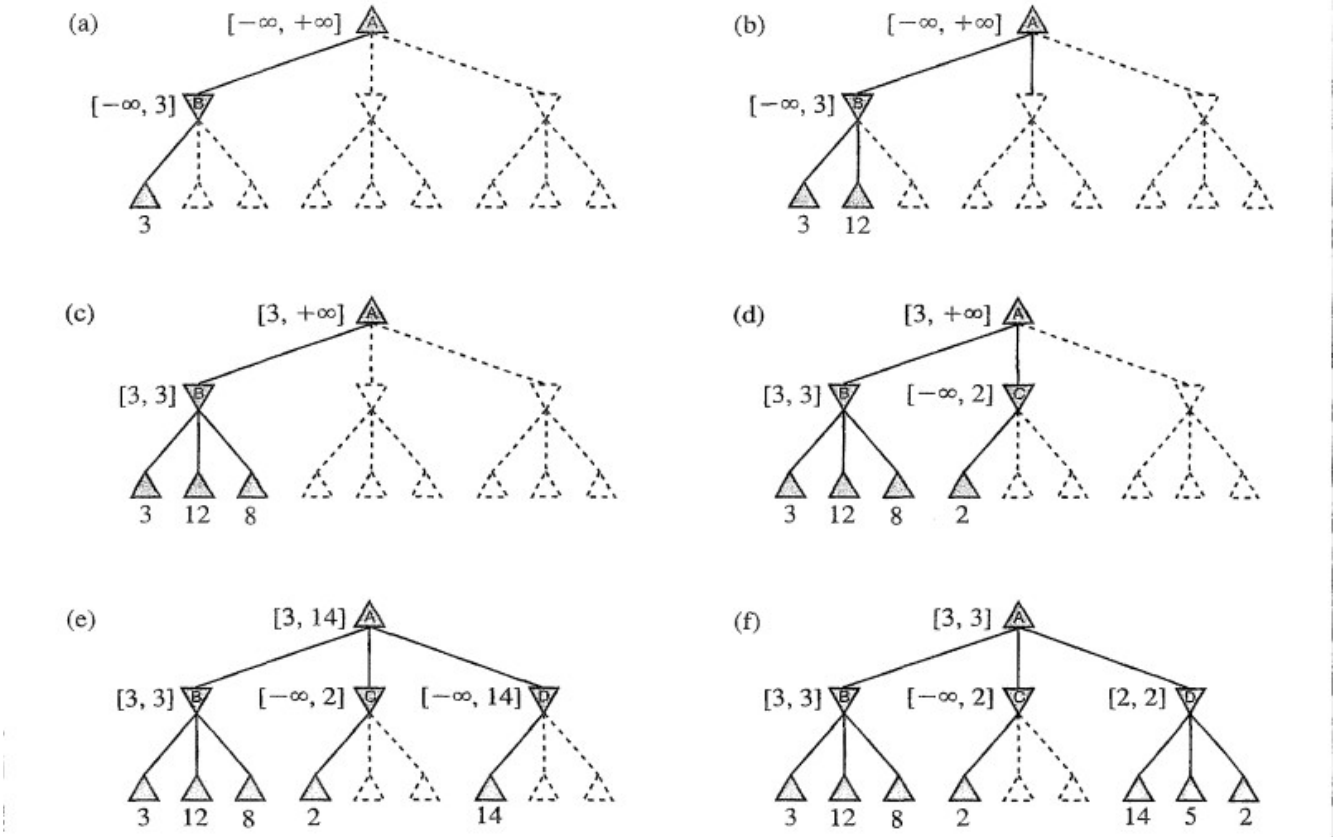


Figura 6.5 Etapas en el cálculo de la decisión óptima para el árbol de juegos de la Figura 6.2. En cada punto, mostramos el rango de valores posibles para cada nodo. (a) La primera hoja debajo de *B* tiene el valor 3. De ahí *B*, que es un nodo MIN, tiene un valor de *como máximo* 3. (b) La segunda hoja debajo de *B* tiene un valor de 12; MIN evitaría este movimiento, entonces el valor de *B* es todavía como máximo 3. (c) La tercera hoja debajo de *B* tiene un valor de 8; hemos visto a todos los sucesores de *B*, entonces el valor de *B* es exactamente 3. Ahora, podemos deducir que el valor de la raíz es *al menos* 3, porque MAX tiene una opción digna de 3 en la raíz. (d) La primera hoja debajo de *C* tiene el valor 2. De ahí, *C*, que es un nodo MIN, tiene un valor de *como máximo* 2. Pero sabemos que *B* vale 3, entonces MAX nunca elegiría *C*. Por lo tanto, no hay ninguna razón en mirar a los otros sucesores de *C*. Este es un ejemplo de la poda de alfa-beta. (e) La primera hoja debajo de *D* tiene el valor 14, entonces *D* vale *como máximo* 14. Este es todavía más alto que la mejor alternativa de MAX (es decir, 3), entonces tenemos que seguir explorando a los sucesores de *D*. Note también que ahora tenemos límites sobre todos los sucesores de la raíz, entonces el valor de la raíz es también como máximo 14. (f) El segundo sucesor de *D* vale 5, así que otra vez tenemos que seguir explorando. El tercer sucesor vale 2, así que ahora *D* vale exactamente 2. La decisión de MAX en la raíz es moverse a *B*, dando un valor de 3.

Otro modo de verlo es como una simplificación de la formula Valor-MiniMax. Los dos sucesores no evaluados del nodo C de la Figura 6.5 tienen valores *x* e *y*, y sea *z* el mínimo entre *x* e *y*. El valor del nodo raíz es:

$$\begin{aligned}\text{MINIMAX-VALUE}(\textit{raíz}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\ &= \max(3, \min(2, x, y), 2) \\ &= \max(3, z, 2) \qquad \text{donde } z \leq 2 \\ &= 3\end{aligned}$$

La poda alfa-beta consigue su nombre de los dos parámetros que describen los límites sobre los valores hacia atrás que aparecen a lo largo del camino:

α = el valor de la mejor opción (el valor más alto) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MAX.

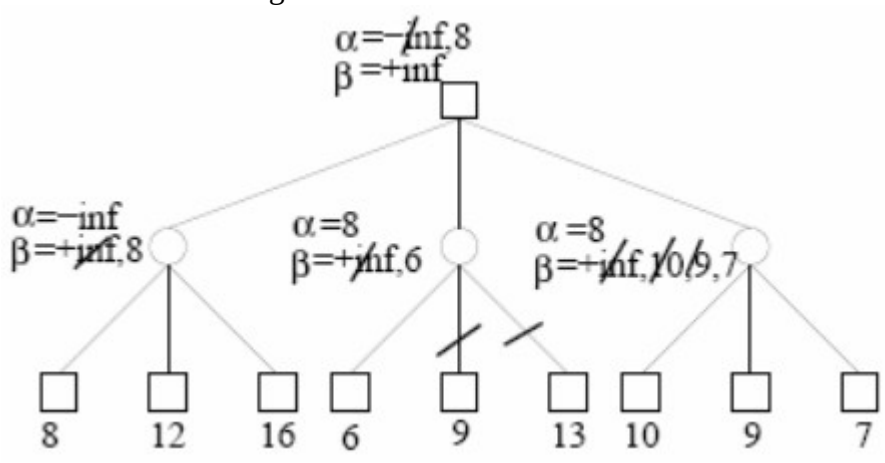
B = el valor de la mejor opción (el valor más pequeño) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MIN.

La búsqueda alfa-beta actualiza los valores de alfa y beta según se va recorriendo el árbol y poda las ramas restantes en un nodo tan pronto como el valor del nodo actual es peor que el valor actual de alfa o beta para MAX o MIN respectivamente.

```
función BÚSQUEDA-ALFA-BETA(estado) devuelve una acción
  variables de entrada: estado, estado actual del juego

  v ← MAX-VALOR(estado, −∞, +∞)
  devolver la acción de SUCESORES(estado) con valor v

---

función MAX-VALOR(estado,α, β) devuelve un valor utilidad
  variables de entrada: estado, estado actual del juego
            α, valor de la mejor alternativa para MAX a lo largo del camino a estado
            β, valor de la mejor alternativa para MIN a lo largo del camino a estado

  si TEST-TERMINAL(estado) entonces devolver UTILIDAD(estado)
  v ← −∞
  para a, s en SUCESORES(estado) hacer
  v ← MAX(V, MIN-VALOR(s, α, β))
      si v ≥ β entonces devolver v
      α ← MAX(α, v)
  devolver v

---

función MIN-VALOR(estado, α, β) devuelve un valor utilidad
  variables de entrada: estado, estado actual del juego
            α, valor de la mejor alternativa para MAX a lo largo del camino a estado
            β, valor de la mejor alternativa para MIN a lo largo del camino a estado

  si TEST-TERMINAL(estado) entonces devolver UTILIDAD(estado)
  v ← +∞
  para a, s en SUCESORES(estado) hacer
      v ← MIN(v, MAX-VALOR(s, α, β))
      si v ≤ α entonces devolver v
      β ← MIN(β, V)
  devolver v
```

Figura 6.7 El algoritmo de búsqueda alfa-beta. Notemos que estas rutinas son lo mismo que las rutinas de MINIMAX de la Figura 6.3, excepto las dos líneas MIN-VALOR y MAX-VALOR que mantienen α y β (y la forma de hacer pasar estos parámetros).

Ejemplo del funcionamiento del algoritmo.



Juegos en los que es imposible o indeseable representar y buscaren el grafo del juego

- Exploración en un nº determinado de niveles: profundidad del juego determinada
- n-movimiento hacia delante.
- A cada nodo hoja se le asigna un valor de acuerdo a una función de evaluación heurística
- El valor se propaga hacia atrás hasta el nodo raíz, es el mejor valor que se puede alcanzar en n movimientos
- Las heurísticas suelen medir la ventaja de un jugador sobre otro.
- Los grafos de juegos se examinan por nivel o por espacio y tiempo del ordenador
- Se puede aplicar el algoritmo MiniMax

EJEMPLO: TIC – TAC – TOE (2 - ply)

HEURÍSTICA: E(n) = X(n) – O(n)
X(n) = nº de líneas ganadoras posibles para MAX
O(n) = nº de líneas ganadoras posibles para MIN