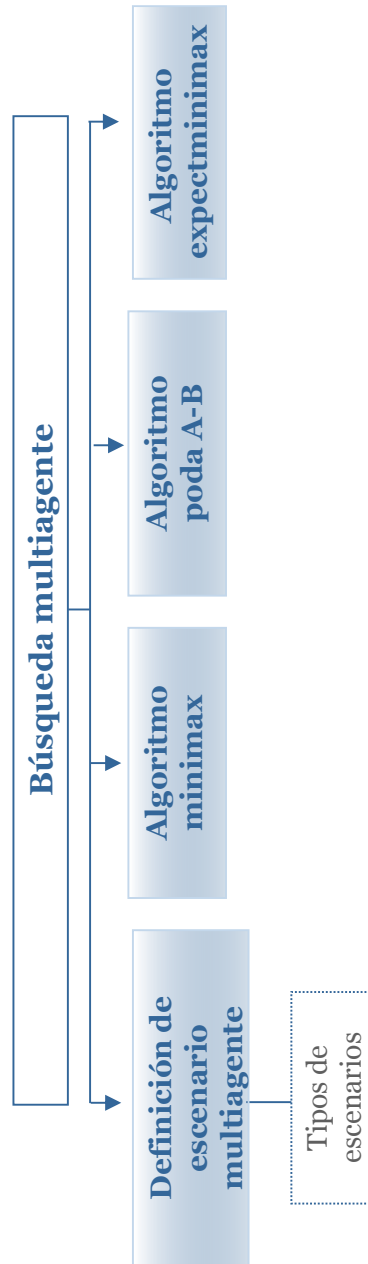


Razonamiento y Planificación Automática

Búsqueda multiagente

Índice

Esquema	3
Ideas clave	4
7.1. ¿Cómo estudiar este tema?	4
7.2. Introducción	4
7.3. Búsqueda minimax	9
7.4. La poda alfa-beta	13
7.5. Búsqueda expectminimax	17
7.6. Referencias bibliográficas	22
Lo + recomendado	23
+ Información	24
Test	26



7.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** que encontrarás a continuación.

En este tema trataremos aquellos problemas en los que nos encontramos con más de un agente interactuando sobre el mismo entorno. En ellos deberemos tener en cuenta el objetivo de cada uno de ellos y sus interrelaciones.

En esta clase de problemas, los agentes no siempre comparten sus objetivos y, como buenos entes racionales, desean alcanzarlos del modo más eficiente posible. En la mayoría de los casos, estos problemas se definen para entornos en los que hay dos agentes enfrentados con metas contrapuestas, pero las técnicas que se emplean pueden ser las mismas en otros muchos escenarios que no son necesariamente bipersonales y/o competitivos.

Hablaremos principalmente del algoritmo minimax presentado por von Neumann, en 1928, en su demostración teórica sobre juegos bipersonales de «suma cero». (Hazewinkel, 2013).

7.2. Introducción

Los problemas multiagente son aquellos en los que más de un agente especializado actúa de modo concurrente en un mismo entorno. Cualquier acción ejecutada por un agente influye en el rendimiento de los demás

agentes del entorno, que no pueden controlar las acciones que el resto va a realizar, aunque sí pueden intentar predecir el comportamiento de los mismos.

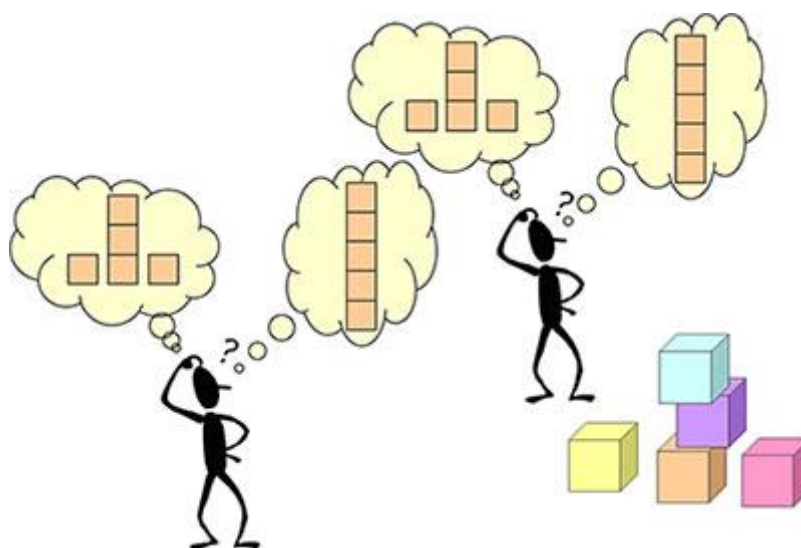


Figura 1. Varios agentes que tienen que interactuar sobre un mismo entorno pueden o no compartir objetivos.

En estos escenarios nos encontramos con distintos tipos de problemas entre varios agentes, atendiendo a su comportamiento entre sí:

- ▶ Escenarios cooperativos: en los que los agentes tienen metas compartidas y, por tanto, las acciones de unos deberían facilitar los objetivos de todos los agentes.
- ▶ Escenarios parcialmente cooperativos: aquellos escenarios en los que algunas metas son compartidas por varios agentes, pero otras pueden ser opuestas.
- ▶ Escenarios antagónicos: donde las metas de todos los agentes son opuestas. El ejemplo «clásico» de escenarios antagónicos es el juego de la suma nula, donde un (grupo de) jugador(es) solo puede ganar algo a coste de otro (grupo de) jugador(es).

Se pueden distinguir diferentes tipos de juegos de suma nula en función de:

- ▶ Número de jugadores: juegos bipersonales (damas) frente a juegos de varios jugadores (Monopoly o Catán).

- ▶ Elementos de azar: juegos con elementos de azar (parchís) frente a juegos sin elementos de azar (ajedrez).
- ▶ Información: juegos con información perfecta (damas) frente a juegos con información incompleta (*black jack*).

Ejemplo del juego de las tres en raya

Emplearemos este juego durante el tema para ejemplificar el funcionamiento de los distintos algoritmos.

En el juego de las tres en raya participan dos jugadores (que denominaremos **min** y **max**). Min y max colocan fichas en las casillas de un tablero 3x3. Por convenio, max usa las fichas X y min usa las fichas O.

En cada casilla solo puede haber una ficha. El tablero comienza vacío y max empieza. Posteriormente, y de modo alternado, min pondrá otra pieza. Así, hasta que se alcance un resultado (vence max, vence min o empate). Al ser un juego de «suma cero», estos resultados siempre implican que o bien se empata, o bien todo lo que gana un jugador es lo que pierde el otro.

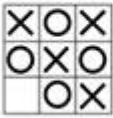
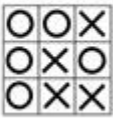
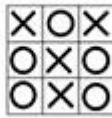
Gana Max	Gana Min	Empate
 <p>gana max</p>	 <p>gana min</p>	 <p>empate</p>

Figura 2. Posibilidades en el juego de las tres en raya. Fuente: Billhart, Fernandez y Ossowski (2015).

Modelos de juegos bipersonales

Nos vamos a centrar en los juegos bipersonales porque permiten ver claramente cómo funcionan los distintos algoritmos que conforman el tema.

Los agentes max y min tienen un conocimiento mínimo *a priori* que se puede ver como:

- s_0	<i>posición inicial (estado inicial)</i>
- <i>expandir</i> : $s \mapsto \{s_{i_1}, \dots, s_{i_n}\}$	<i>cjto. finito de posiciones sucesores</i>
- <i>terminal?</i> : $s \mapsto \text{true} \mid \text{false}$	<i>prueba terminal</i>
- $U: s \mapsto k, k \in \mathfrak{R}$	<i>función parcial de utilidad del juego</i>

Figura 3. Conocimiento a priori de un agente.

La función *expandir* crea los estados derivados de emplear cualquiera de las acciones permitidas en el estado s .

En este tipo de problemas, evaluaremos el desempeño de los agentes en base a una función de utilidad, que solo se define, inicialmente, en los estados meta. Por ejemplo, en los problemas de suma nula, max gana si y solo si min pierde. Por tanto, la función de utilidad $U(s)$ es:

gana max: $U(s) = +\infty$	empate: $U(s) = 0$	gana min: $U(s) = -\infty$
----------------------------	--------------------	----------------------------

Tabla 1. Función de utilidad básica.

Al igual que en los algoritmos de búsqueda, los problemas de resolución de escenarios bipersonales emplean la construcción de árboles de búsqueda para gestionar el proceso de concesión de la meta u objetivo del agente.

En el caso de escenarios de juego como los descritos anteriormente, el árbol que generamos se llama **árbol de juego**. Y está formado por la tupla $G = \langle N, E, L \rangle$, donde N es un conjunto de nodos, $E \subseteq N \times N$, $L = \{\text{max}, \text{min}\}$.

En este árbol, G empieza siendo vacío y etiquetaremos la raíz como max, indicando que el turno inicial es siempre de este. Todos los sucesores se irán entregando de modo ordenado.

Cada nivel del árbol representa media jugada (un *ply*). Por otro lado, todas las hojas de un árbol, si se expande completamente, representan nodos terminales del juego.

Ejemplo de árbol de juego para tres en raya:

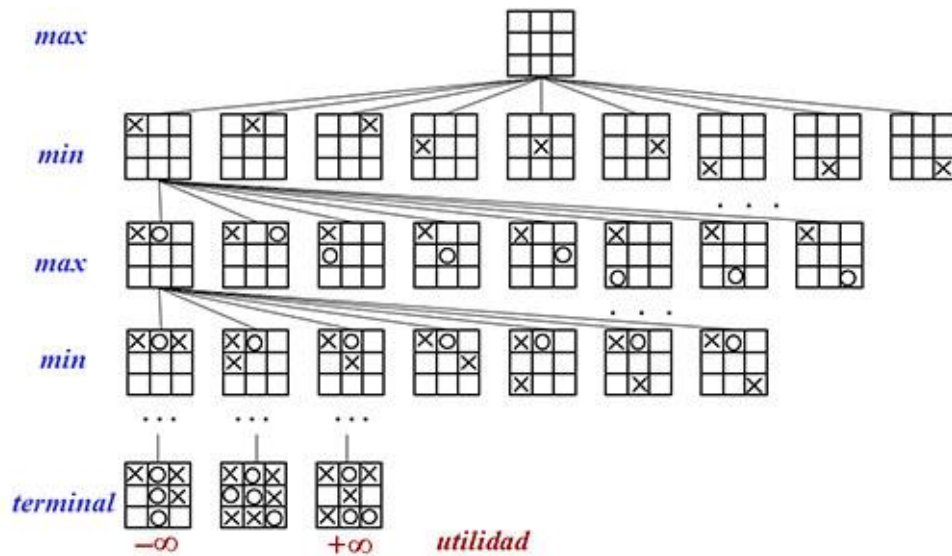


Figura 4. Árbol de juego para tres en raya.

La mejor jugada del agente max

La estrategia define las jugadas de max en todos los estados alcanzables del juego, es decir, da una respuesta a todas las jugadas posibles de min. Un subárbol del árbol de juego.

Si aplicásemos únicamente algoritmos de búsqueda estándar, nos encontraríamos con que min nunca querrá realizar acciones que se encuentren en el camino principal de max.

Estrategia óptima o racional

En los escenarios competitivos con agentes racionales siempre se asume que min realizará la mejor acción para sí mismo de aquellas que el controlador max le haya

dejado en su momento. La estrategia óptima para max es la **estrategia minimax**: maximizar la utilidad mínima en cada jugada.

7.3. Búsqueda minimax

Minimax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta. Es un cálculo que se emplea de forma recursiva. Su funcionamiento se resume en elegir el mejor movimiento para uno mismo partiendo de la suposición de que el contrincante escogerá el peor para el adversario.



Figura 5. Método minimax.

En último lugar llegará el valor de utilidad al nodo raíz. Y tendremos que la acción que lleva al camino del nodo meta con mayor valor es la jugada óptima.

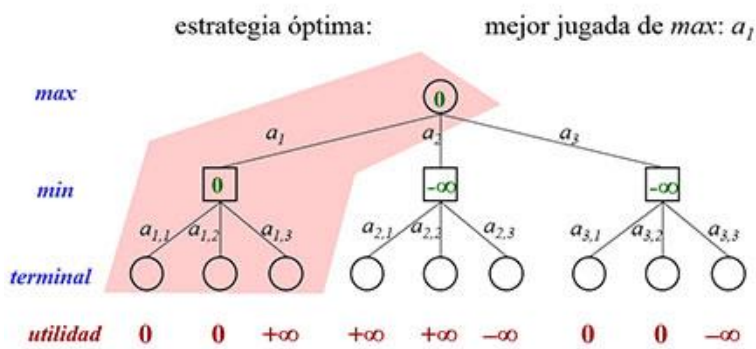


Figura 6. Estrategia óptima para una búsqueda minimax.

El algoritmo explorará los nodos del árbol y les asigna un valor numérico mediante una función de utilidad, empezando por nodos terminales y subiendo a la raíz. La función de utilidad definirá lo buena que es la posición de un jugador cuando la alcanza.

Algoritmo: <ul style="list-style-type: none"> • funciones mutuamente recursivas • <i>estado</i> es el estado actual 	<ul style="list-style-type: none"> • α: máximo de la utilidad de los sucesores de un nodo <i>max</i> • β: mínimo de la utilidad de los sucesores de un nodo <i>min</i>
{MaxValor en el Minimax básico}	{MinValor en el Minimax básico}
Función MaxValor(estado) Si <i>terminal?</i> (estado) entonces devolver (<i>U</i> (estado)) sucesores \leftarrow <i>expandir</i> (<i>max</i> , estado) $\alpha \leftarrow -\infty$ Para cada <i>s</i> \in sucesores hacer $\alpha \leftarrow \max(\alpha, \text{MinValor}(s))$ devolver (α) Fin {MaxValor}	Función MinValor(estado) Si <i>terminal?</i> (estado) entonces devolver (<i>U</i> (estado)) sucesores \leftarrow <i>expandir</i> (<i>min</i> , estado) $\beta \leftarrow +\infty$ Para cada <i>s</i> \in sucesores hacer $\beta \leftarrow \min(\beta, \text{MaxValor}(s))$ devolver (β) Fin {MinValor}

Figura 7. Algoritmo minimax (versión preliminar).

El principal problema de esta técnica es que, incluso en juegos muy simples, resulta imposible crear un árbol completo de juego. Esto es debido al crecimiento exponencial del árbol de juego.

Minimax con suspensión

Cuando el problema manifiesta unos problemas graves de expansión de los nodos, la solución se basa en heurísticas.

La idea es reemplazar la prueba terminal por una prueba de suspensión, es decir, en vez de tener que desarrollar todo el árbol de juego, realizamos una expansión hasta el nodo de profundidad *l*. Para ello deberemos emplear una **función de evaluación** *e*, que estima la utilidad esperada del juego correspondiente a una posición *s* determinada (*e* debe coincidir con la función de utilidad *u* en los nodos terminales).

Lo más habitual es emplear una función de tipo lineal ponderada:

$$e(s)w_1f_1(s) + w_2f_2(s) + \dots + w_nf_n(s)$$

Por ejemplo, para el ajedrez nos podríamos encontrar con una función de la forma:

$$e(s) = \text{"suma de los valores materiales en } s\text{"}$$

Mientras que, en el juego de las tres en raya, la función de evaluación puede ser de la forma:

$$e(s) = \text{"número de líneas abiertas para líneas max en } s\text{"} - \text{"número de líneas abiertas para líneas min en } s\text{"}$$

Al emplear funciones heurísticas fuertes, dado que deseamos limitar la complejidad, no podremos garantizar que la estrategia minimax sea óptima, esto dependerá de la calidad de las heurísticas.

En el siguiente ejemplo se muestra minimax con suspensión:

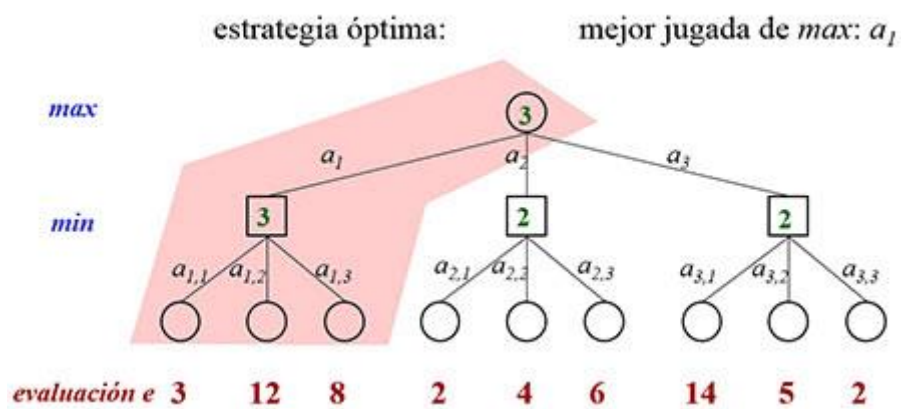


Figura 8. Minimax con suspensión.

El algoritmo es:

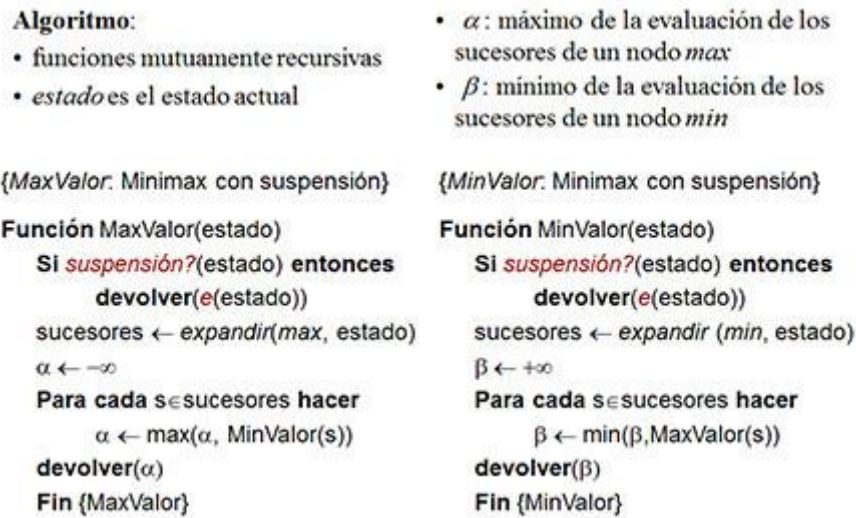


Figura 9. Algoritmo minimax con suspensión.

Ejemplo de las tres en raya:

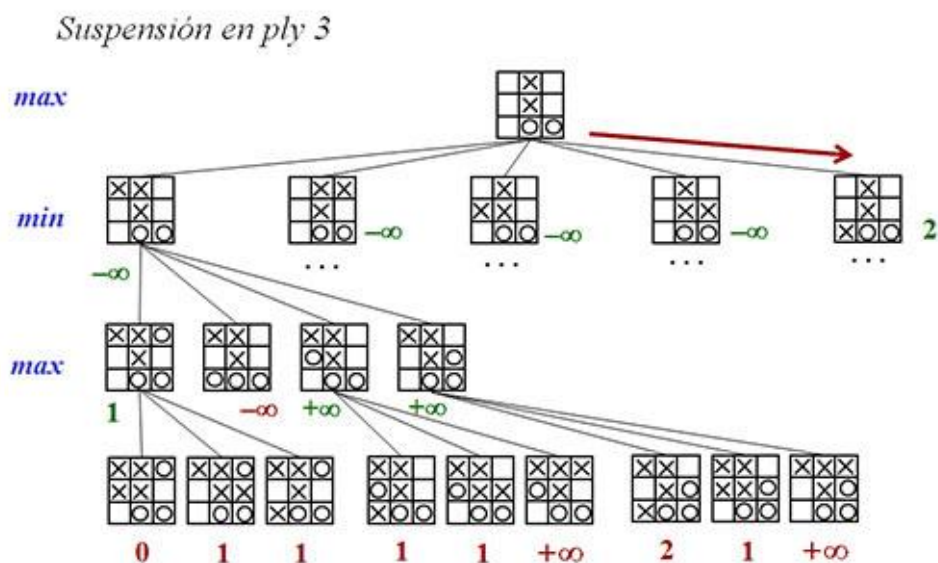


Figura 10. Ejemplo de algoritmo minimax con suspensión para el juego tres en raya.

El algoritmo minimax tiene un problema de complejidad importante en el caso general, $O(b^d)$ (factor de ramificación b ; número de *plys* explorados d).

Minimax genera el árbol de juego completo, en profundidad primero, hasta los nodos (o el nivel) de suspensión.

Existen extensiones de la estrategia minimax, algunas las veremos en las secciones siguientes:

- ▶ Mejorar la complejidad (poda a-b: descartar nodos irrelevantes).
- ▶ Juegos con elementos de azar (expectminimax: hay un nuevo jugador «azar»).
- ▶ Juegos con información parcial (búsqueda en estados de creencias).
- ▶ Mejorar las heurísticas: aprender funciones de evaluación y suspensión (por ejemplo, por el «efecto horizonte»).

7.4. La poda alfa-beta

El crecimiento exponencial de los nodos que se deben explorar en la estrategia minimax hace de él un algoritmo que presenta una complejidad poco deseable. Si bien este exponente no se puede eliminar, por lo menos es posible dividir a la mitad su valor, con la mejora de rendimiento que eso conlleva.

Es decir, que existe la posibilidad de tomar una decisión minimax correcta sin tener que mirar todos los nodos en el árbol. La poda alfa-beta permite eliminar partes del árbol.

La idea inicial es que, en muchas ocasiones, no es necesario evaluar todos los sucesores de un nodo para obtener el valor exacto de dicho nodo.

Por ejemplo:

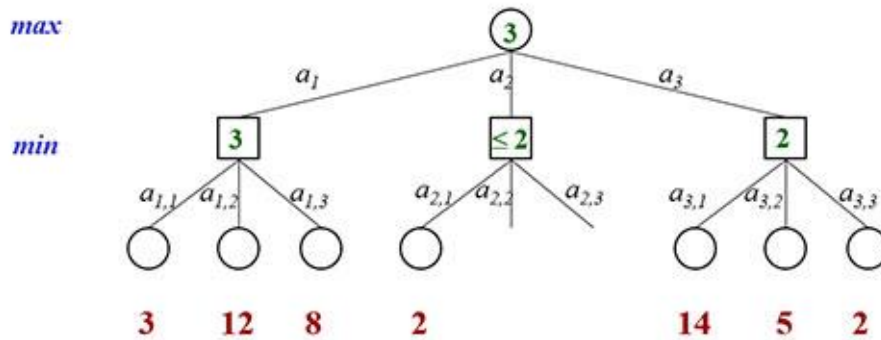


Figura 11. Ejemplo de poda α - β .

Poda alfa:

- α : evaluación más alta encontrada (de momento) en un nodo max.
- β : evaluación más baja encontrada (de momento) en su sucesor directo min.

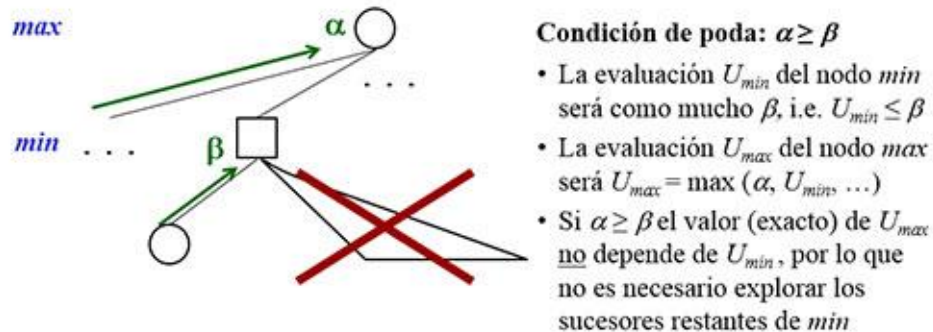


Figura 12. Poda alfa.

Poda beta:

- β : evaluación más baja encontrada (de momento) en un nodo min.
- α : evaluación más alta encontrada (de momento) en su sucesor directo max.

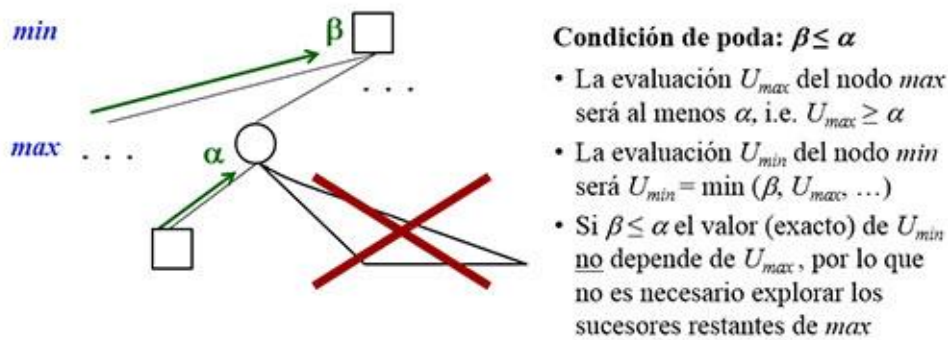


Figura 13. Poda beta.

Poda alfa profunda:

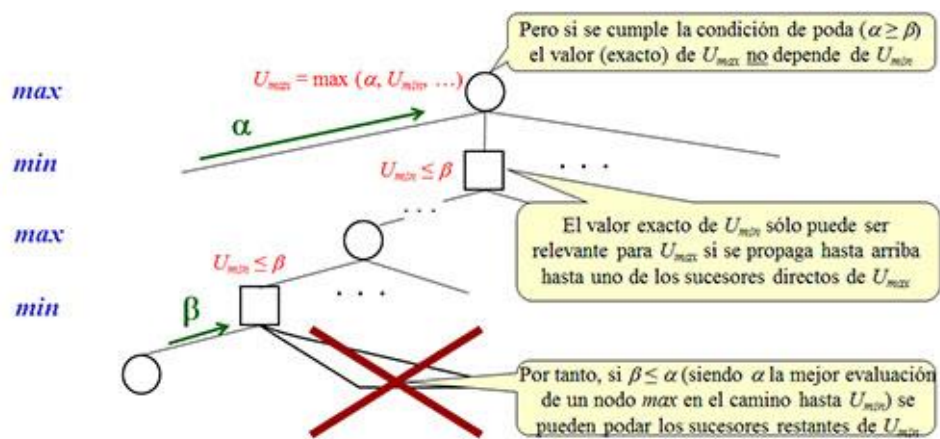


Figura 14. Poda alfa profunda.

La poda beta profunda funciona de forma dual. Emplearemos dos funciones mutuamente recursivas que irán creando las expansiones y podas de max y min de modo alternado.

Minimax con poda alfa-beta:

Algoritmo:

- funciones mutuamente recursivas
- *estado* es el estado actual

- α es el mejor valor de evaluación para *max* en el camino hasta *estado*
- β es el mejor valor de evaluación para *min* en el camino hasta *estado*

{MaxValor, Minimax con poda α - β }

Función MaxValor(estados, α , β)

Si suspensión?(estado) **entonces**

devolver(e(estado))

sucesores \leftarrow expandir(max, estado)

Para cada s \in sucesores **hacer**

$\alpha \leftarrow \max(\alpha, \text{MinValor}(s, \alpha, \beta))$

Si $\alpha \geq \beta$ **entonces devolver**(α)

devolver(α)

Fin {MaxValor}

{MinValor, Minimax con poda α - β }

Función MinValor(estados, α , β)

Si suspensión?(estado) **entonces**

devolver(e(estado))

sucesores \leftarrow expandir(min, estado)

Para cada s \in sucesores **hacer**

$\beta \leftarrow \min(\beta, \text{MaxValor}(s, \alpha, \beta))$

Si $\beta \leq \alpha$ **entonces devolver**(β)

devolver(β)

Fin {MinValor}

Figura 15. Minimax con poda alfa-beta.

Ejemplo minimax con poda alfa-beta:

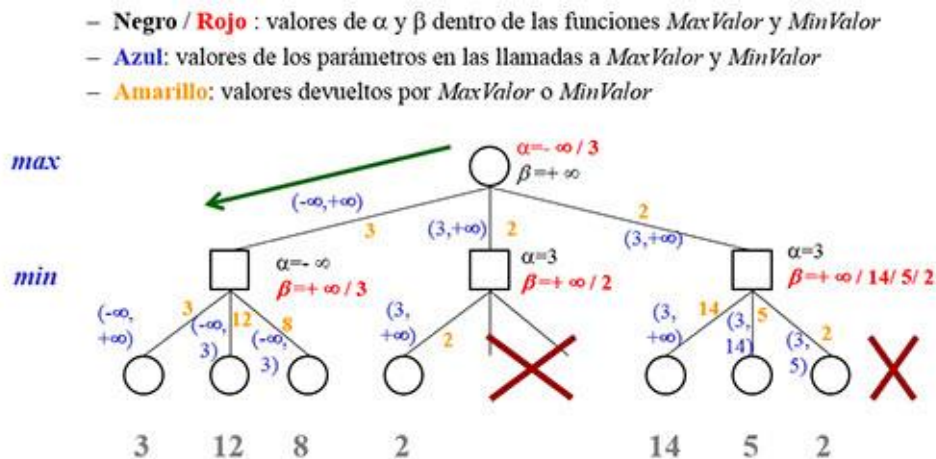


Figura 16. Ejemplo minimax con poda alfa-beta.

El algoritmo minimax con poda α - β siempre produce el mismo resultado que sin la poda. Pero existe una relación directa en la mejora del rendimiento de la poda asociada al orden de exploración de los nodos. La eficiencia de minimax con poda α - β depende del orden en el que se exploran los nodos.

Ejemplo de complejidad para factor de ramificación b y con d plys explorados		$B=10, d=4$
Minimax sin poda	$O(b^d)$	$10^4 = 10\,000$ nodos
Minimax con poda (mejor caso)	$O(b^{d/2})$	$10^2 = 100$ nodos
Minimax con poda (caso medio)	$O(b^{3d/4})$	$10^3 = 1\,000$ nodos

Tabla 2. Análisis de complejidad del algoritmo minimax.

7.5. Búsqueda expectminimax

Para los juegos bipersonales con elementos de azar se suele emplear el algoritmo expectminimax (Billhart, Fernandez y Ossowski, 2015). Cuando nos encontramos en un escenario donde aparecen elementos de azar en entornos multiagente, hay que evolucionar la idea del minimax. Para ello, añadimos un nuevo jugador «azar», que se incluye en el árbol siempre que haya un evento independiente de los jugadores y cuyo resultado es aleatorio.

Los nodos sucesores de estas jugadas de «azar» son los posibles valores resultantes de los elementos de azar. Por ejemplo, los distintos valores de una tirada de dado o el éxito o fracaso de una acción azarosa. Cada uno de los sucesores de un nodo «azar» tiene, por tanto, asociada una probabilidad de ocurrencia.

En un ejemplo como el del Backgammon simplificado en el que partimos de un estado inicial como este:

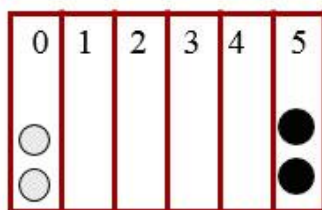


Figura 17. Backgammon.

Tenemos el objetivo de mover nuestras fichas (por ejemplo, blanca para max hasta el campo cinco y min lo tendrá que hacer hasta el campo cero), contando con que empieza max y que en cada jugada primero se tira una moneda (donde obtener cara tiene el valor uno y cruz el valor dos).

Después, se mueve una de las fichas uno o dos campos en la dirección deseada (dependiendo del resultado de la tirada de la moneda), no siendo posible mover fichas a un campo donde el adversario tenga ya una. Y, en el caso de que el jugador no pueda mover, pierde el turno.

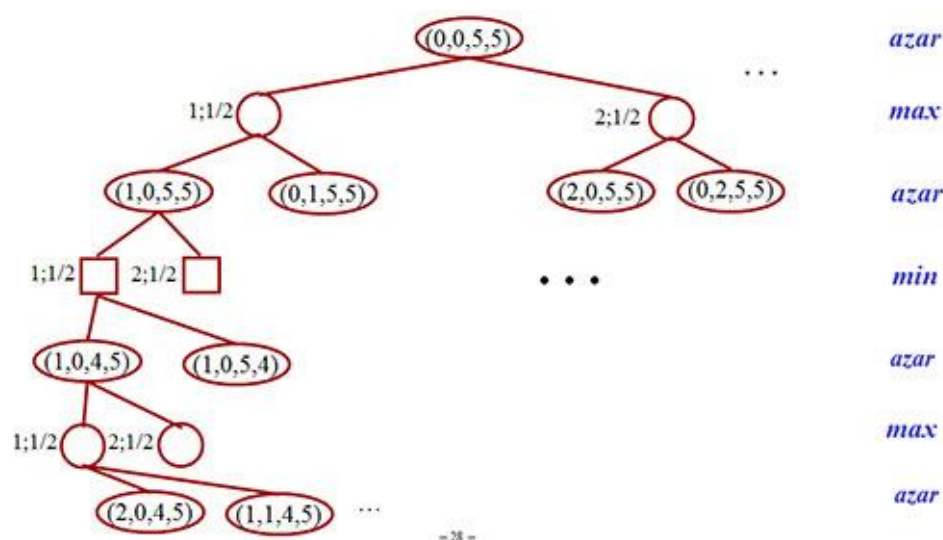


Figura 18. Árbol del juego. Representación de estados: (x_1, x_2, y_1, y_2) , donde x_1 y x_2 son posiciones de las fichas blancas y las y_1 y y_2 posiciones de las fichas negras.

ExpectMinimax tiene como objetivo, al igual que el minimax general, elegir la mejor jugada para max. Para ello debemos propagar los valores de utilidad/evaluación hacia los nodos padres.

$$ExpectMinimax(n) = \begin{cases} e(n) & \text{si } n \text{ es nodo suspensión} \\ \max_{s \in \text{expandir}(n)} (ExpectMinimax(s)) & \text{si } n \text{ es nodo max} \\ \min_{s \in \text{expandir}(n)} (ExpectMinimax(s)) & \text{si } n \text{ es nodo min} \\ \sum_{s \in \text{expandir}(n)} (p(s) \cdot ExpectMinimax(s)) & \text{si } n \text{ es nodo azar} \end{cases}$$

Figura 19. Valor propagado al padre en función del tipo de nodo.

Por ejemplo, partiendo de la situación de partida siguiente y tocándole a max:

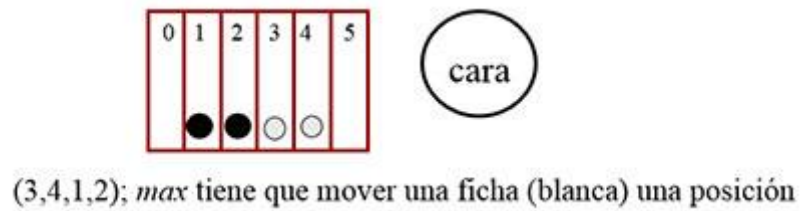


Figura 20. Situación de una partida de Backgammon.

Si el nivel de suspensión cinco y la función de evaluación que empleamos es la siguiente:

$$e((a,b,c,d)) = a+b+c+d$$

Dado que los valores de a , b , c y d son mejores para max cuanto más cerca del cinco se encuentren, tendríamos una evaluación para el estado anterior de:

$$e((3,4,1,2))=10$$

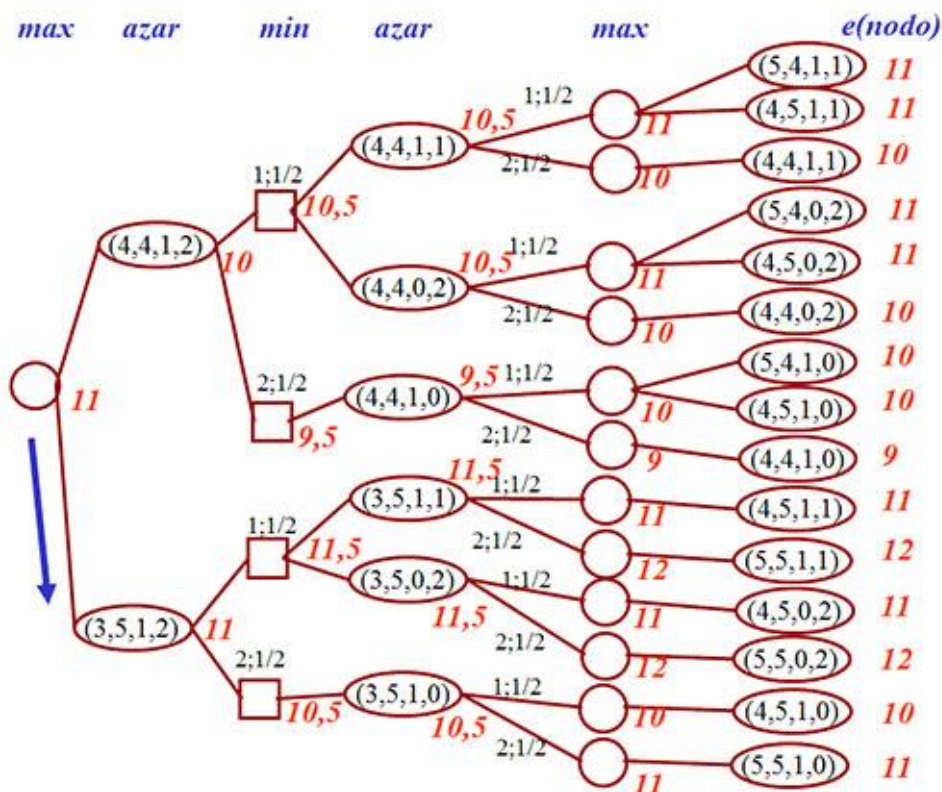


Figura 21. Expansión de nodos para un algoritmo expectminimax para el backgammon.

Criterios para las funciones de evaluación/utilidad

Es importante tener en cuenta que la escala de los valores de dichas funciones sí importa (no como ocurría en el algoritmo minimax):

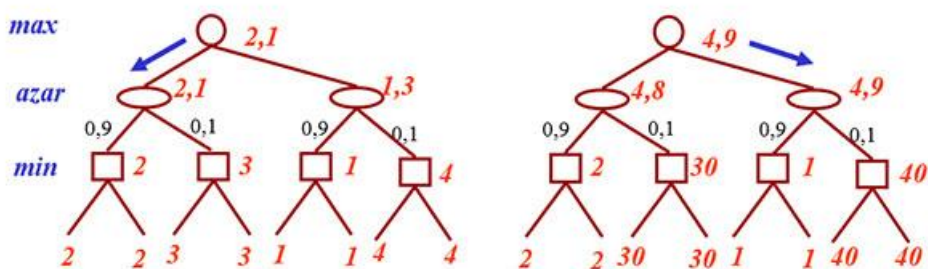


Figura 22. Dos ejemplos de árbol con distintos valores de escala generan distintos recorridos.

Por tanto, las funciones no deben devolver $+\infty$ o $-\infty$ porque reservaremos estos valores para los nodos de azar, por lo que es interesante escalar los valores a un intervalo finito (por ejemplo, entre 0 y 1).

La **función de evaluación** e debe estimar la probabilidad de ganar. En el caso ideal, e es una **transformación lineal positiva** de dicha probabilidad.

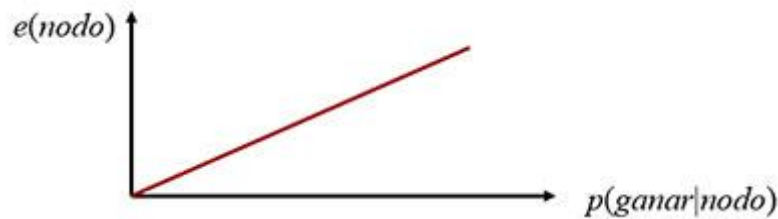


Figura 23. Estimación ideal de la probabilidad de victoria en función del valor $e(\text{nodo})$.

Habitualmente, no es sencillo establecer una función e que cumpla este criterio para todos los estados del juego (nodos). Por lo que, normalmente, una función e es más «exacta» cuanto más cerca está el estado actual del juego de un estado terminal.

Es preferible aplicar la búsqueda (*expect*)*minimax* en el máximo número de *plys* posibles y, solo al final, aplicar la función de evaluación e .

La complejidad *expectminimax* es muy elevada y depende del nivel de suspensión d (*plys*), el factor de ramificación b (jugadas posibles) y los elementos de azar con n posibilidades, pero si incluimos los nodos de azar, nos encontramos con una complejidad de $O(b^d \cdot n^d)$.

Número de <i>plys</i> d anticipados	Número de nodos en el árbol
1	$20 \cdot 21 = 420$
2	176.400
3	74.088.000

Tabla 3. Ejemplo Backgammon: $n=21$ (2 dados) y $b \gg 20$

Para más detalles sobre la aplicación de la poda α - β al algoritmo *expectminimax*, puedes ver la sección 5.5.1. de (Russell y Norvig, 2004).

7.6. Referencias bibliográficas

Billhart, H., Fernandez, A. Y Ossowski, S. (2015). *Inteligencia Artificial: Ejercicios Resueltos*. Madrid: Editorial Universitaria Ramón Areces.

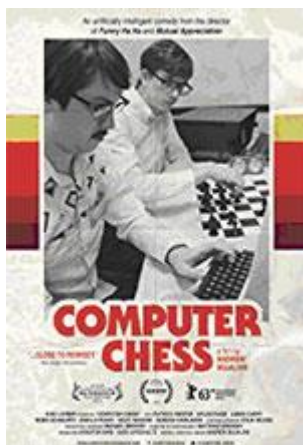
Hazewinkel, M. (2013). Minimax Principle. En Hazewinkel, M. (Ed.), *Encyclopaedia of Mathematics, Volume 6* (pp. 237-245). Dordrecht: Springer.

Russell, S. y Norvig, P. (2004). *Inteligencia Artificial: Un Enfoque Moderno*. Madrid: Pearson Educación.

Lo + recomendado

No dejes de ver

Computer Chess



Título original: *Computer Chess*.

Año: 2013.

Duración: 92 min.

Director: Andrew Bujalski.

Interpretación: Kris Schludermann, Tom Fletcher, Wiley Wiggins...

Una comedia existencial de los hombres brillantes que enseñaban a jugar al ajedrez a las máquinas antes, cuando las máquinas parecían torpes y nosotros inteligentes.

Juegos de guerra



Título original: *War games*.

Año: 1983.

Duración: 114 min.

Director: John Badham.

Interpretación: Matthew Broderick, Dabney Coleman, John Wood, Ally Sheedy...

Es una película que narra las aventuras de un muchacho fan de la informática que por error entra en contacto con un «superordenador» del ejército americano. Presenta una escena en la que se pone a jugar el superordenador contra sí mismo a las tres en raya, claro ejemplo de búsqueda minimax.

A fondo

Algoritmos de juegos

Aguilar, P. (2008). *IA - Algoritmos de juegos*. Cataluña: Facultad de Informática de Barcelona, Universidad Politécnica de Cataluña.

Este documento proporciona una visión divulgativa sobre el área de los algoritmos de juegos de la inteligencia artificial. Proporciona un buen punto de partida para profundizar en los algoritmos de búsqueda con adversarios. Presenta los algoritmos avanzados que se basan en minimax, como Negamax u otros algoritmos como SSS* o Scout.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

http://www.lsi.upc.edu/~bejar/ia/material/trabajos/Algoritmos_Juegos.pdf

Algoritmo minimax

Adrigm. (2010). Algoritmo minimax, un jugador incansable [Mensaje en un blog]. Razón artificial.

Características y funcionamiento del algoritmo minimax en la teoría de juegos.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

<http://razonartificial.com/2010/08/algoritmo-minimax-un-jugador-incansable/>

Webgrafía

Asociación Española para la Inteligencia Artificial

Página web de la AEPIA, Asociación Española para la Inteligencia Artificial, donde podrás estar al día de todo lo relacionado con esta área.



Accede a la página web a través del aula virtual o desde la siguiente dirección web:

<http://www.aepia.org/aepia/index.php/materiales>

Bibliografía

Trigozo-Galoc, G., Julca-Huancas, U. y Gómez-Díaz, M. (1994). *Inteligencia Artificial y Algoritmo: Minimax*. Perú: Facultad de Ingeniería y Arquitectura, EAP Ingeniería de Sistemas, Universidad Peruana Unión.

1. Se llama estrategia óptima a la que:
 - A. Implica el mejor resultado garantizado para max.
 - B. Implica el mejor resultado garantizado para min.
 - C. Implica que tanto max como min obtengan el mejor resultado garantizado.

2. Los tipos de problemas multiagente contemplan:
 - A. Escenarios parcialmente cooperativos y escenarios con metas opuestas.
 - B. Escenarios con metas compartidas y escenarios con metas opuestas.
 - C. Escenarios con metas compartidas, escenarios parcialmente cooperativos y escenarios con metas opuestas.

3. Los juegos se pueden clasificar en función de:
 - A. Número de jugadores, elementos de azar e información.
 - B. Información y número de jugadores.
 - C. Elementos de azar y número de jugadores.

4. En un juego en el que max puede asumir que min hará lo mejor para sí mismo, lo que implica que es lo peor para max, la estrategia óptima para max es:
 - A. Minimizar la utilidad mínima en cada jugada, minimax.
 - B. Maximizar la utilidad mínima en cada jugada, minimax.
 - C. No existe una estrategia óptima para estos casos.

5. Minimax genera:
 - A. El árbol de juego incompleto, primero en profundidad, pero no llega a cubrir todos los nodos.
 - B. El árbol de juego completo, primero en profundidad, hasta los nodos suspensión.
 - C. No genera ningún árbol.

6. El algoritmo minimax con poda alfa-beta:
- A. Produce el mismo resultado que sin poda.
 - B. Nunca produce el mismo resultado que sin poda.
 - C. El resultado depende de la exploración de los nodos, por lo que no se tiene el mismo resultado nunca.
7. El algoritmo expectminimax utiliza el algoritmo minimax y:
- A. Añade un árbol complejo teniendo en cuenta el azar para que en cada uno de los nodos ocurra siempre el mismo resultado.
 - B. Añade el azar como si fuera un jugador y lo incluye en el árbol siempre.
 - C. Añade el azar como si fuera un jugador, siempre que haya un evento con independencia de los jugadores cuyo resultado sea aleatorio.
8. El caso ideal de e en la función de evaluación e es:
- A. Una transformación lineal positiva de dicha probabilidad.
 - B. Una regresión lineal positiva de dicha probabilidad.
 - C. Una transformación lineal negativa de dicha probabilidad.
9. La complejidad de expectminimax es:
- A. Inversamente proporcional al número de nodos del árbol.
 - B. Proporcional al número de nodos en el árbol.
 - C. No tiene relación con el número de nodos en el árbol.
10. La eficiencia de minimax con poda alfa-beta:
- A. Es independiente del orden en el que se exploran los nodos.
 - B. El orden en el que se exploran los nodos no influye.
 - C. Depende del orden en el que se exploran los nodos.