Universidad
Francisco de Vitoria
UFV Madrid

Ingeniería del Conocimiento

Tema 5:
Búsqueda Informada (II)

Roberto Rodriguez Galán

Departamento de Ingeniería Informática
Universidad Francisco de Vitoria

Objetivos del tema

- Ubicación
  - Unidad 2: **BUSQUEDA EN ESPACIO DE ESTADOS**
  - *Tema 5: Búsqueda Informada: Heurísticas (II)*

- Objetivos generales
  - Entender el concepto de *árbol de búsqueda* cuando se trata de un juego con adversarios
  - Conocer los *distintos tipos de juegos* y su facilidad de resolución
  - Entender los algoritmos de búsqueda *minimax* con sus variaciones y mejoras: *Decisiones imperfectas* y *poda alfa-beta*
  - Saber *aplicar de cada método* en función de la completitud y *complejidad* espacial y temporal

Contenido

1. Búsqueda con adversarios
2. Búsqueda Minimax
    1. Minimax con decisiones imperfectas
    2. Poda Alfa-Beta
3. Otros tipos de juegos
4. Estado del arte en problemas de juegos

1. Búsqueda con adversarios
2. Búsqueda Minimax
    1. Minimax con decisiones imperfectas
    2. Poda Alfa-Beta
3. Otros tipos de juegos
4. Estado del arte en problemas de juegos

1. Búsqueda con adversarios

- Búsqueda en un entorno hostil, competitivo e impredecible donde interviene al menos un adversario contrario a nuestros objetivos
  - Los movimientos de un jugador, por sí solos no aseguran la victoria: es necesaria una estrategia de oposición
  - El tiempo disponible para cada movimiento (reglas de juego) impone soluciones aproximadas (no sirve la fuerza bruta)
  - No se puede calcular exactamente las consecuencias de cada movimiento ➔ hay que tomar una decisión sin la certeza de que sea la correcta

- Los problemas de búsqueda con adversario (o de *conflicto de intereses*) se denominan **juegos**. La resolución del problema es la estrategia para ganar el juego
  - Estrategia: búsqueda en un espacio de billones de nodos con profundización selectiva

1. Búsqueda con adversarios

- ¿Por qué gustan tanto en IA?
  - Divertidos
  - Difíciles (el ajedrez tiene un factor de ramificación medio de 35 y pueden sucederse 50 movimientos de cada jugador)
  - Fáciles de formalizar y con un número pequeño de acciones

  Los juegos son para la IA como la F1 para la ingeniería del automóvil

- Programa con muy buen rendimiento: ajedrez
  - Importante: desde 1956, el ajedrez era un objetivo para IA
  - Deep Blue gano a Kasparov en 1997

1. Búsqueda con adversarios

- Tipos de juegos:
  - *Información completa*
    - *Deterministas*: sin azar (4-en-raya, ajedrez, damas)
    - *No deterministas*: con azar (backgammon, parchís, monopoly…)
  - *Información incompleta* con azar (juegos de cartas, dominó)

- De cara a simplificar, consideraremos los juegos con estas características:
  - Información completa: se conoce en cada momento el estado completo del juego
  - Deterministas: no entra en juego el azar (se puede generalizar)
  - 2 jugadores cuyas jugadas se alternan
  - Juegos de suma cero: Lo que "gana" uno lo "pierde" el otro. Al acabar, cada jugador pierde, gana o empata

1. Búsqueda con adversarios

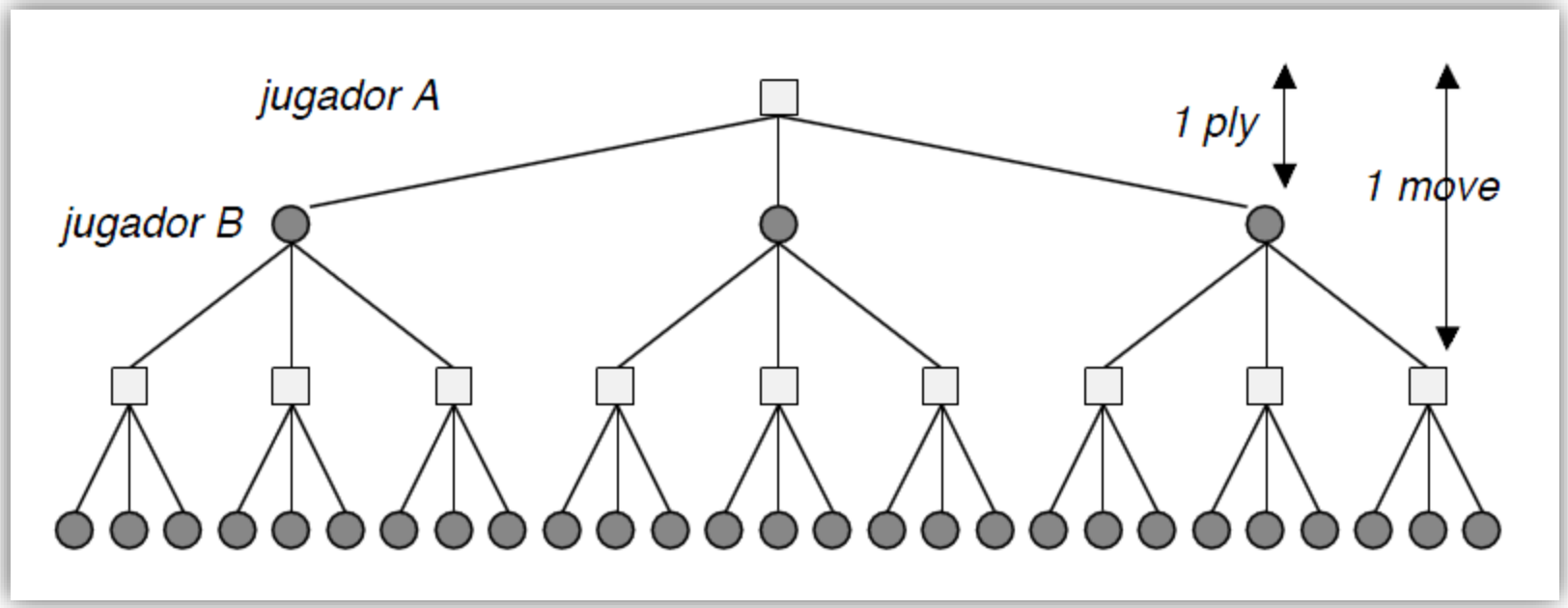
- Un juego puede definirse como un *problema* de búsqueda:
  - *Estado inicial*: posición inicial del tablero con una indicación de quien debe mover
  - *Operadores*: movimientos legales posibles
  - *Test terminal*: determina cuando ha finalizado el juego
  - *Función de utilidad f(n)*: asigna un valor numérico al resultado del juego

- Árbol de juego: representación explícita de todas las secuencias de jugadas posibles en una partida
  - Cada nivel representa, alternativamente, los movimientos posibles de cada jugador
  - Las hojas corresponden a estados GANAR (+∞), PERDER (-∞) o EMPATAR (0)
  - Cada camino desde la raíz (el estado inicial del juego) hasta una hoja representa una partida completa

1. Búsqueda con adversarios

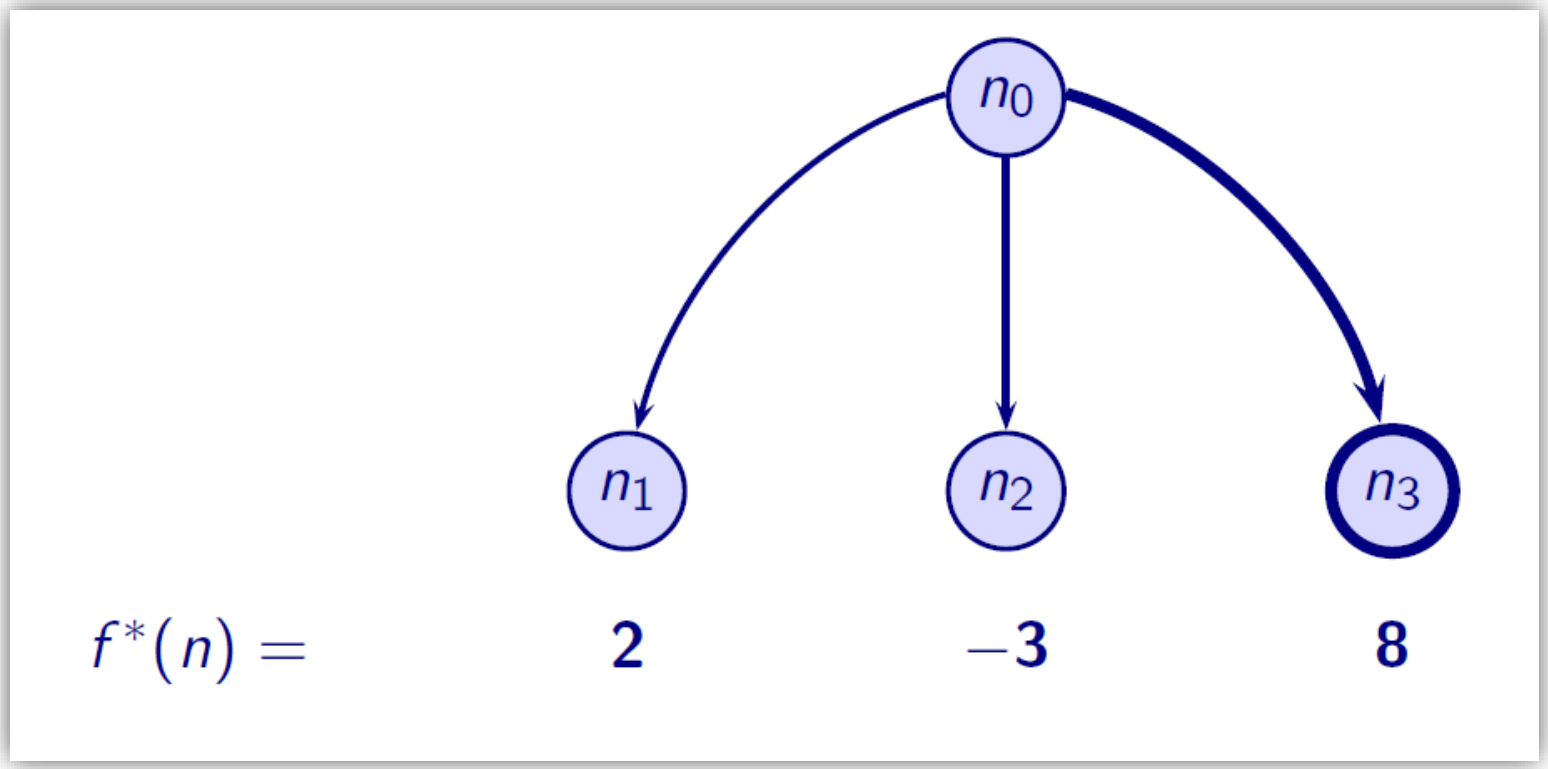
- Los algoritmos de búsqueda vistos hasta ahora no sirven:

  El problema ya no es encontrar un camino en el árbol de juego (esto depende de los movimientos futuros del oponente), sino decidir el mejor movimiento dado el estado actual del juego



1. Búsqueda con adversarios

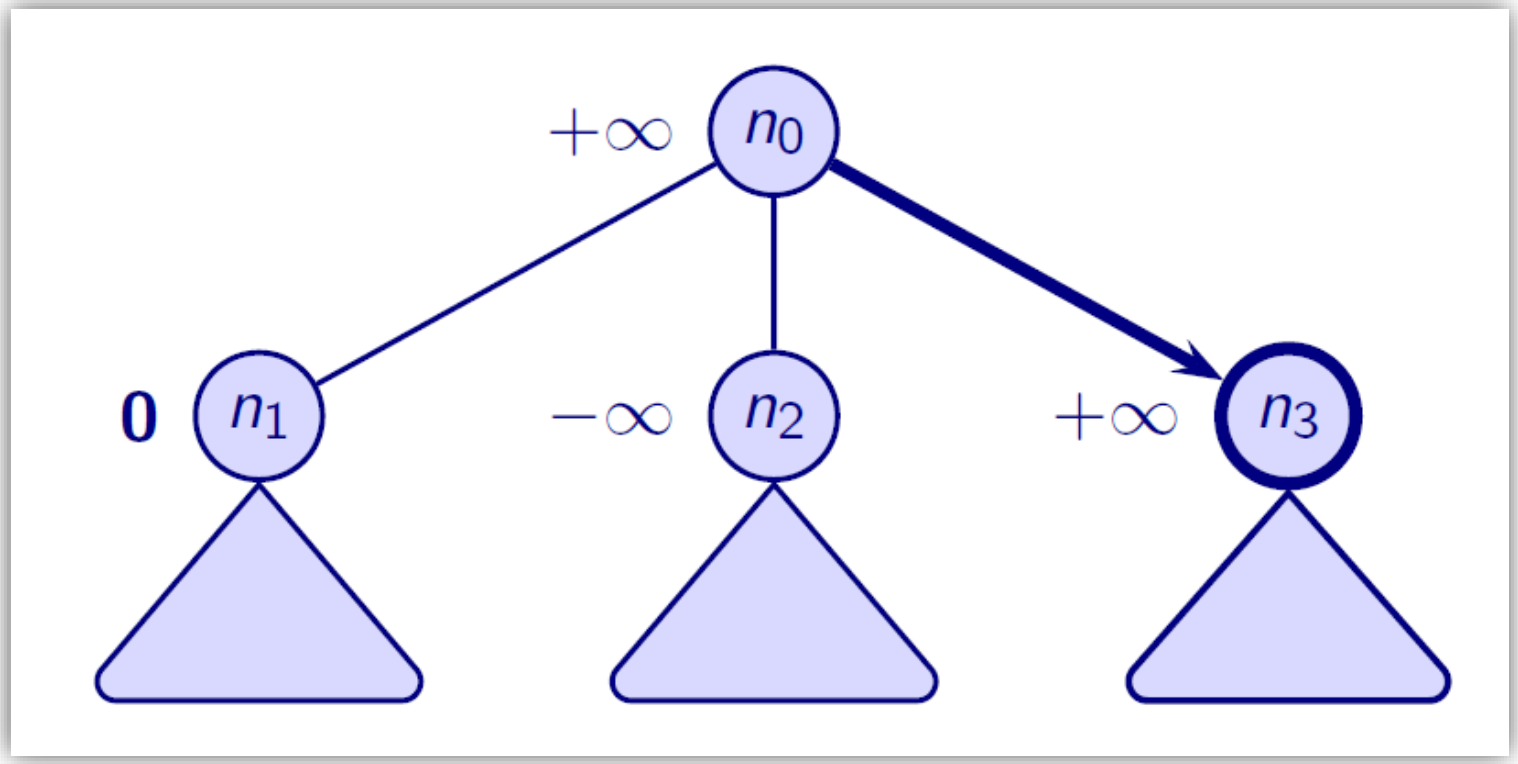
- Resolución con función de utilidad perfecta



PROBLEMA: Normalmente no se conoce esa función

1. Búsqueda con adversarios

- Resolución con búsqueda completa



PROBLEMA: Es intratable, no se puede realizar en un tiempo razonable

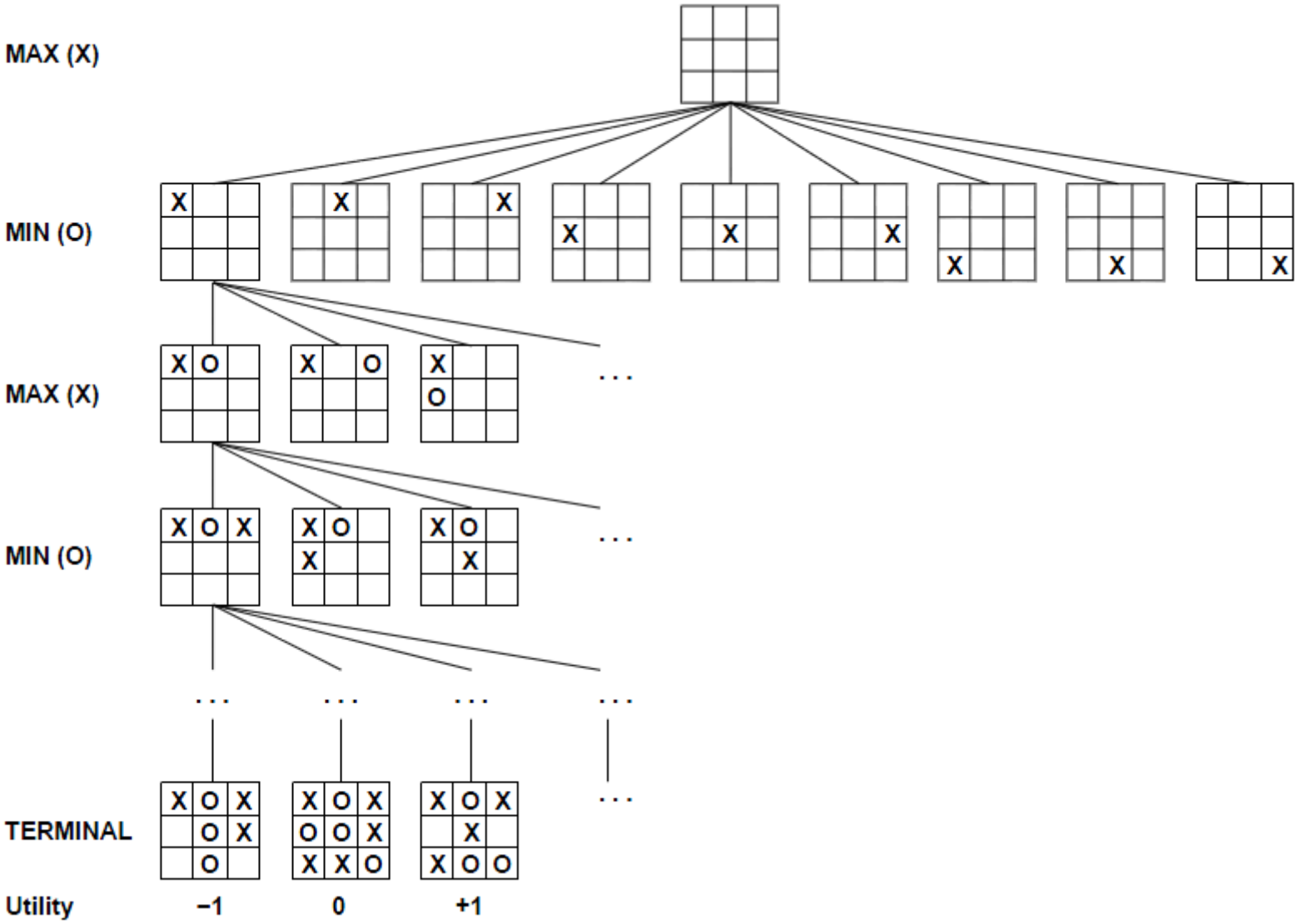
1. Búsqueda con adversarios
2. Búsqueda Minimax
    1. Minimax con decisiones imperfectas
    2. Poda Alfa-Beta
3. Otros tipos de juegos
4. Estado del arte en problemas de juegos

2. Búsqueda Minimax

- Algoritmo iterativo para un juego con dos jugadores: *MAX* y *MIN*:
  - Primero juega *MAX*, después *MIN*, y así hasta terminar
  - Son nodos *MAX/MIN* aquéllos en los que juega *MAX/MIN*
  - Si llamamos nivel 0 a la raíz, los nodos de nivel par son nodos *MAX*y los de nivel impar son nodos *MIN*

- *f(n)* es la función de utilidad y asigna un valor numérico al resultado del juego desde el punto de vista de *MAX*.
  - Es la misma para los nodos de ambos jugadores
  - En los nodos terminales el valor de *f(n)* es
    - *f(n)*= +∞: gana *MAX* (G)
    - *f(n)*=-∞:gana *MIN* (P)
    - *f(n)*=0: *MAX* y *MIN* empatan (E)

2. Búsqueda Minimax



2. Búsqueda Minimax

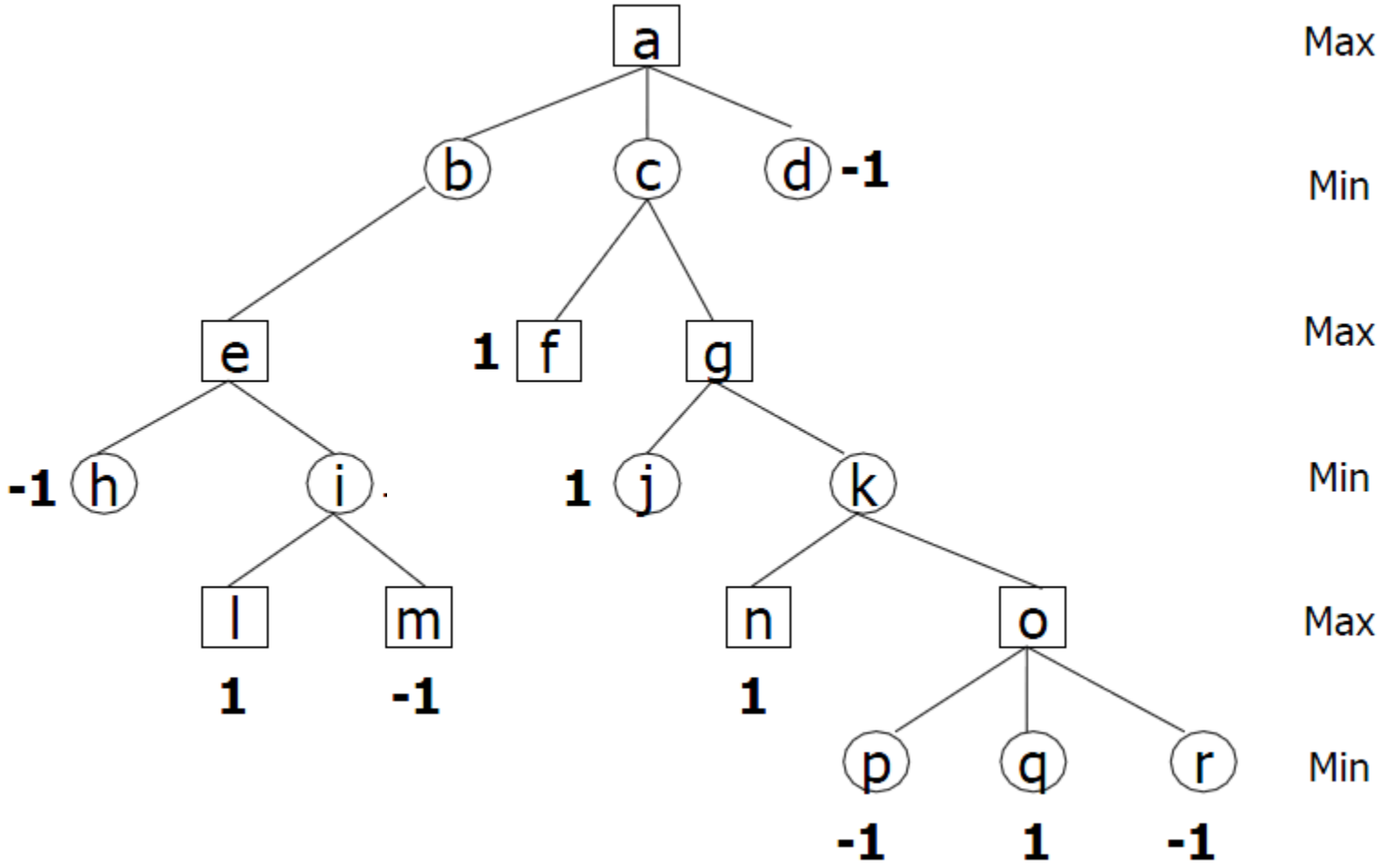
- A partir de los valores asignados a los nodos terminales, podemos **ascender** hasta la raíz *(propagar hacia arriba)*:
  - A cada nodo *MAX* se le asigna el máximo de los valores de sus hijos (nodos *MIN*)
    - *MAX* intenta maximizar su ventaja
    - Buscamos el mejor movimiento para *MAX*
  - A cada nodo *MIN* se le asigna el mínimo de los valores de sus hijos (nodos *MAX*)
    - *MIN* procura minimizar la puntuación de *MAX*
    - Asumimos que siempre elegirá el **peor** movimiento para *MAX*, es decir, el **mejor** para sus intereses *(siempre jugará óptimamente)*

- *MAX* tiene que encontrar una estrategia que le permita maximizar *f(n)* (ganar) independientemente de lo que haga *MIN*, que juega a minimizar *f(n)* (ganar él)
  - Elige como siguiente movimiento el del sucesor del nodo inicial con mayor valor

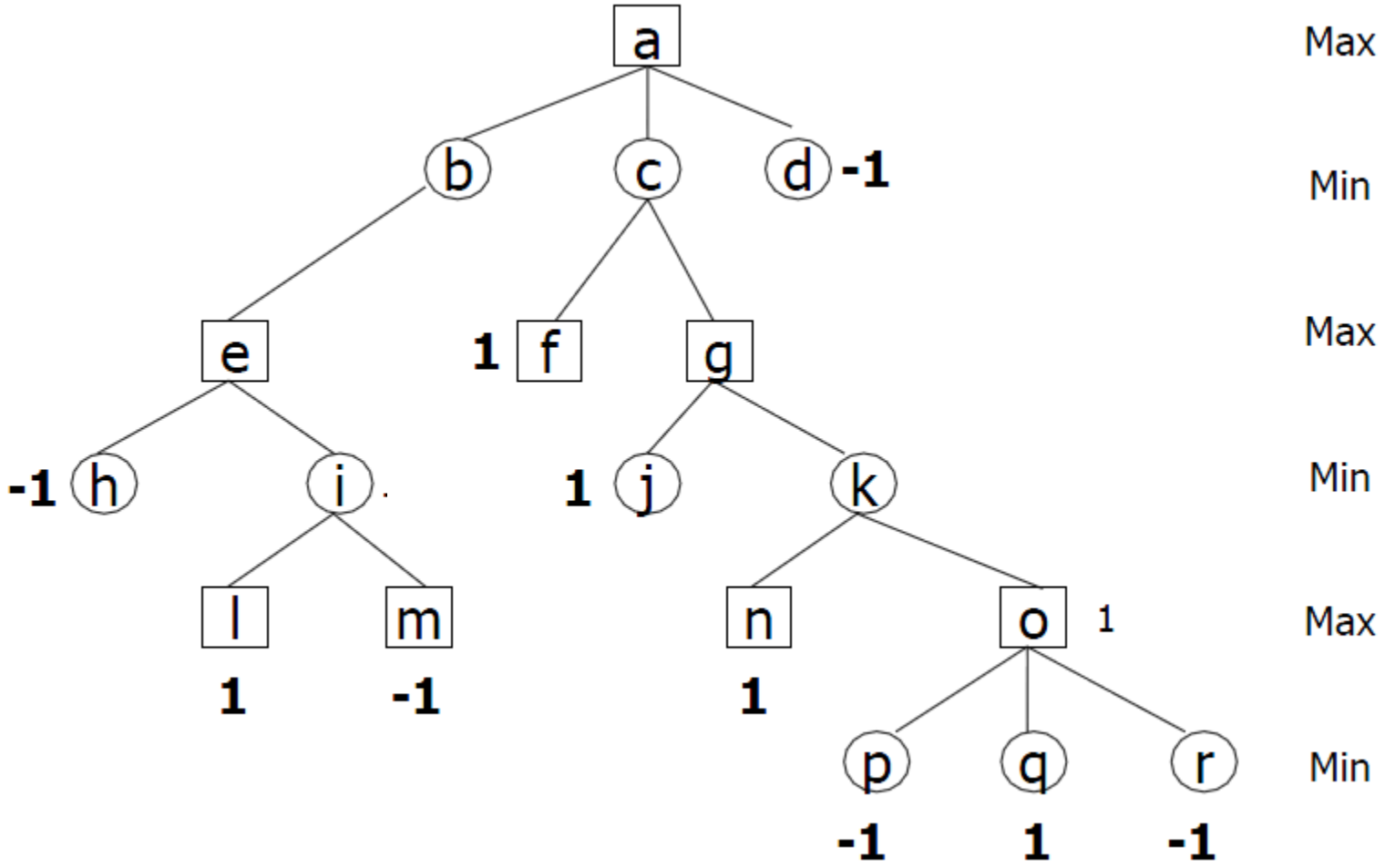
2. Búsqueda Minimax

- Pasos del algoritmo minimax:
  1. Generar el árbol de búsqueda completo, hasta los estados terminales.
  2. Aplicar la función de utilidad a los estados terminales.
  3. Usar la utilidad de los estados terminales para determinar la de los nodos de un nivel superior
  4. Continuar propagando los valores hacia la raíz.
  5. Al llegar a la raíz, escoger la acción que lleva a una mayor utilidad.

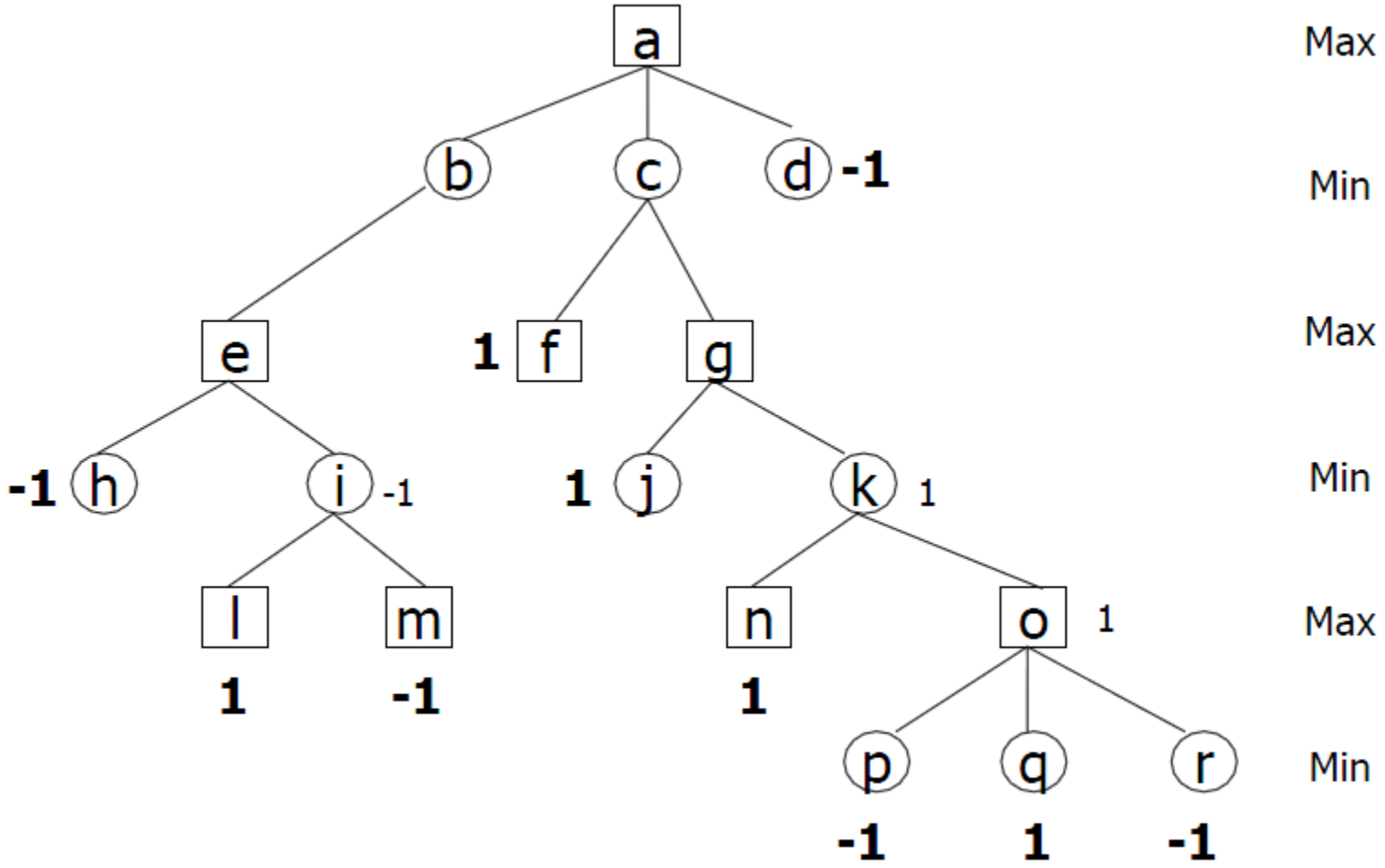
2. Búsqueda Minimax



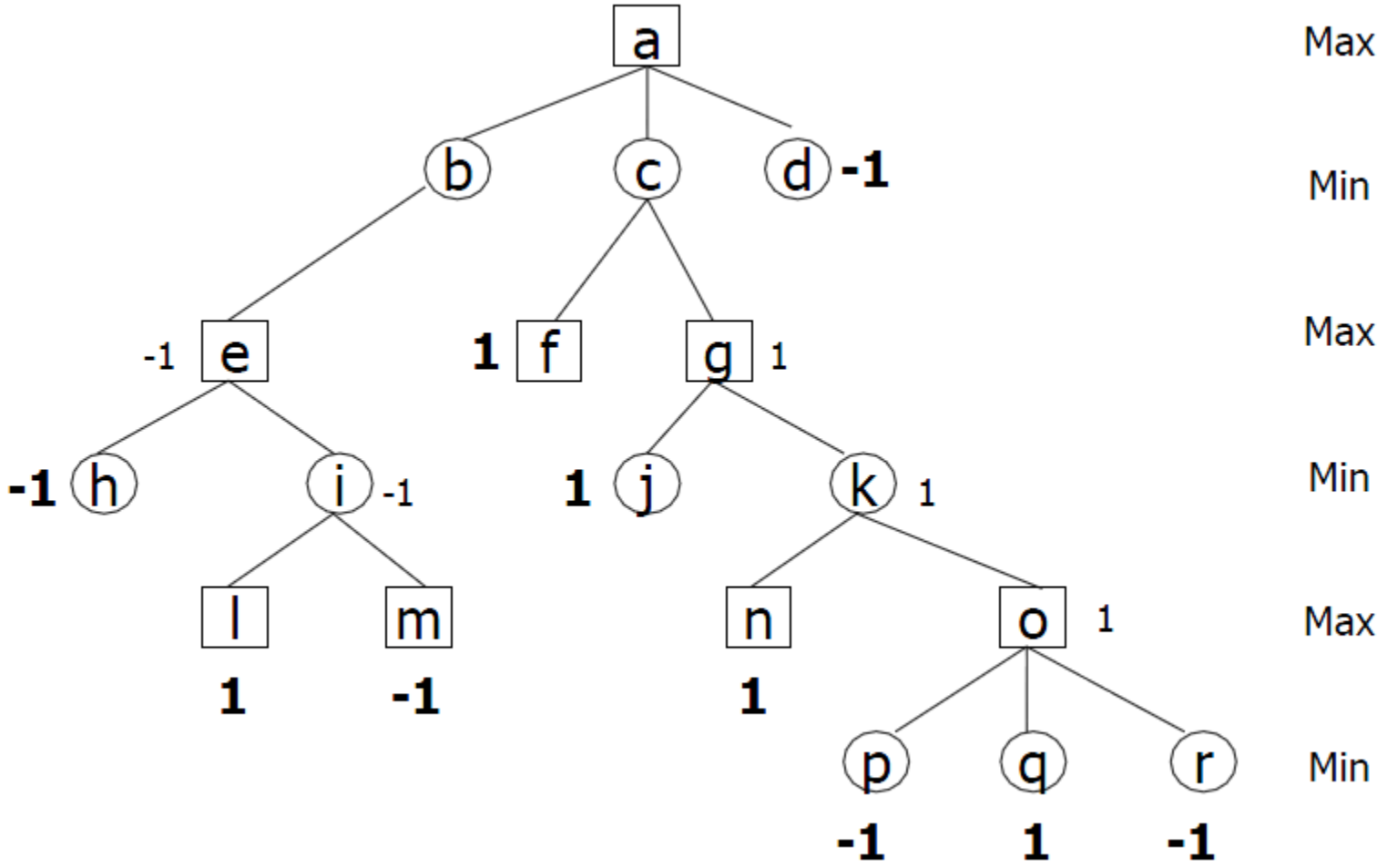
2. Búsqueda Minimax



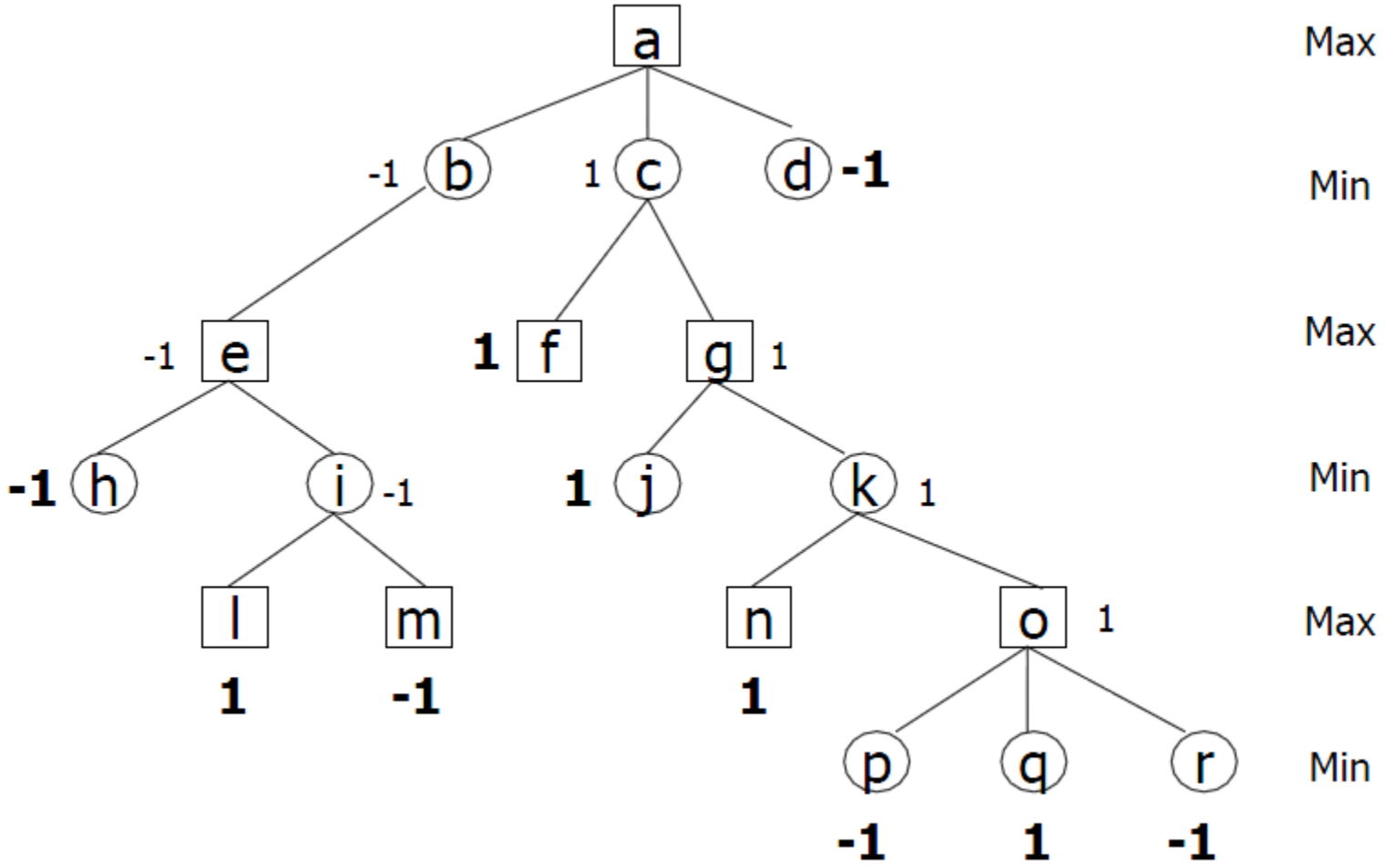
2. Búsqueda Minimax



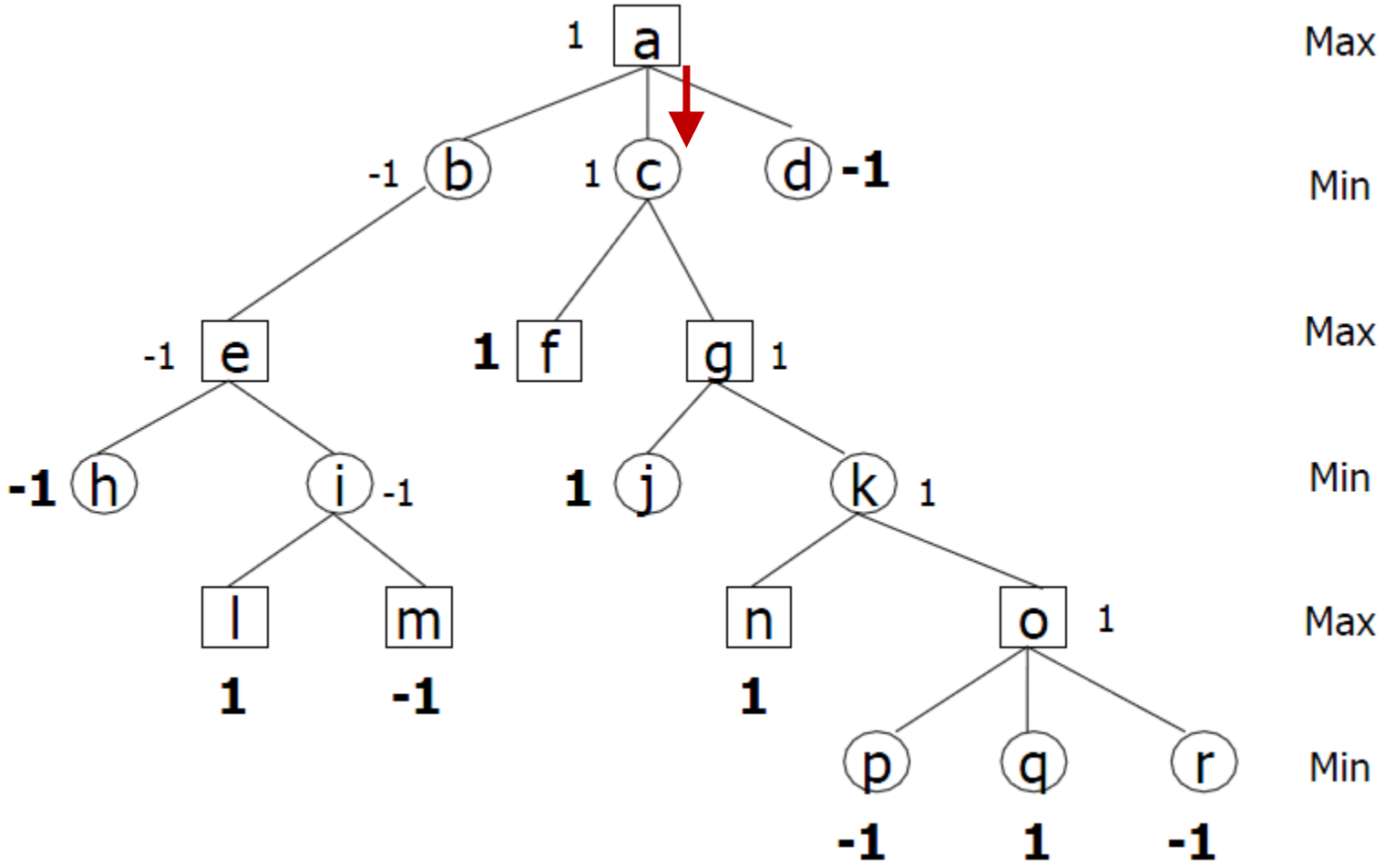
2. Búsqueda Minimax



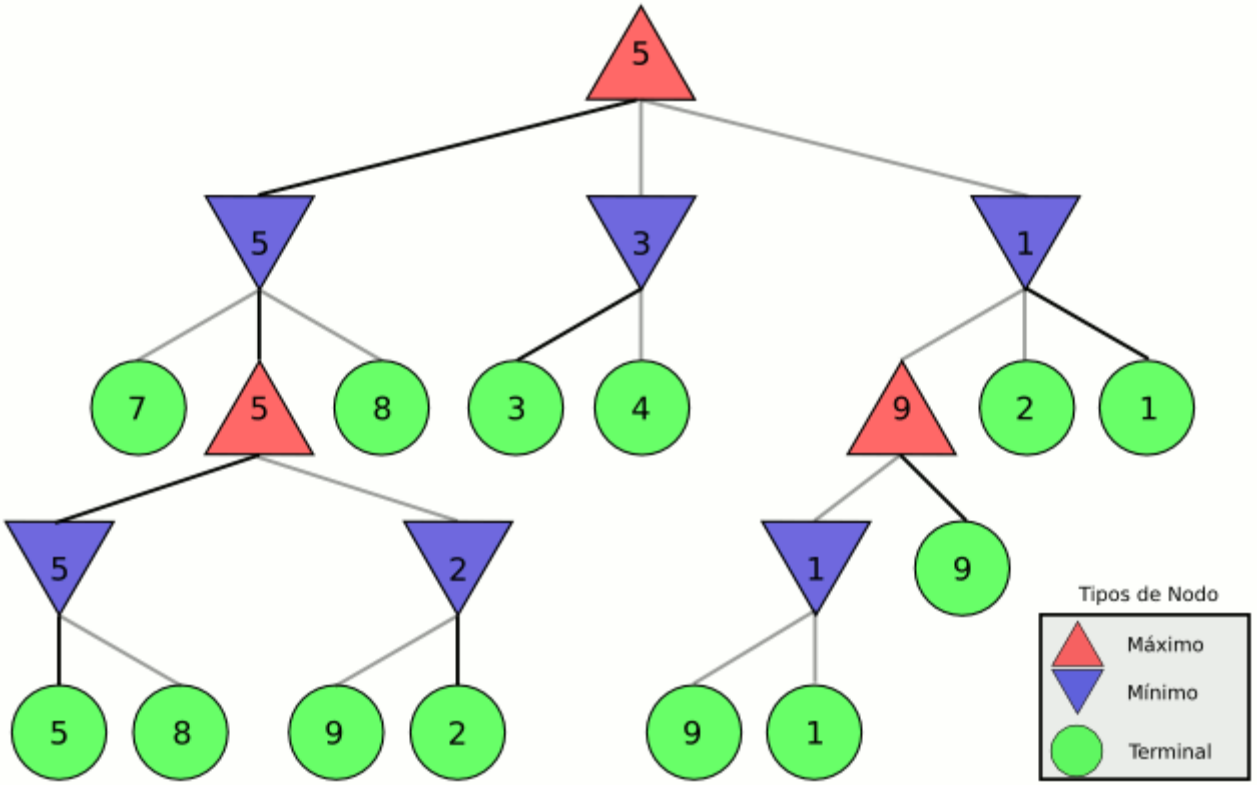
2. Búsqueda Minimax



2. Búsqueda Minimax



2. Búsqueda Minimax



2. Búsqueda Minimax

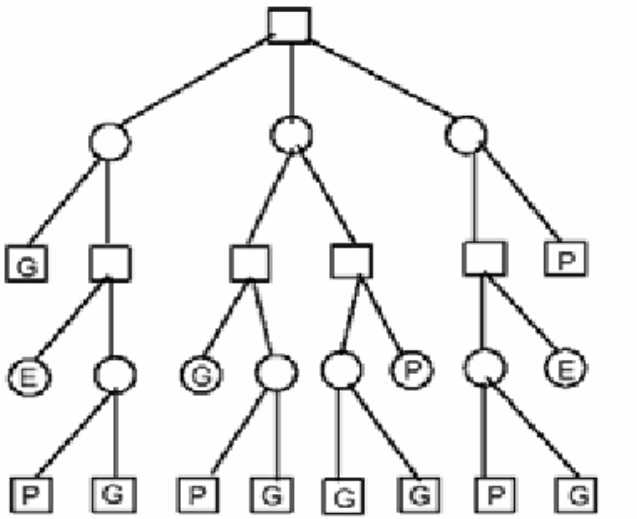
- El valor ascendido a la raíz indica el valor del mejor estado que el jugador *MAX* puede aspirar a alcanzar
  - La etiqueta de un nodo nos indica lo mejor que podría jugar *MAX* en el caso de que se enfrentase a un oponente perfecto

- Árbol solución (o estrategia de juego) para *MAX*
  - Subárbol del árbol de juego con los movimientos que debe realizar *MAX* ante cualquier movimiento posible de *MIN*
  - Contiene
    - La raíz
    - Un sucesor de cada nodo *MAX* no terminal que aparezca en él
    - Todos los sucesores de cada nodo *MIN* que aparezca en él

- Árbol ganador para *MAX*: árbol solución en el que todos los nodos terminales dan a *MAX* ganador
  - Asegura que el jugador *MAX* ganará, haga lo que haga *MIN*

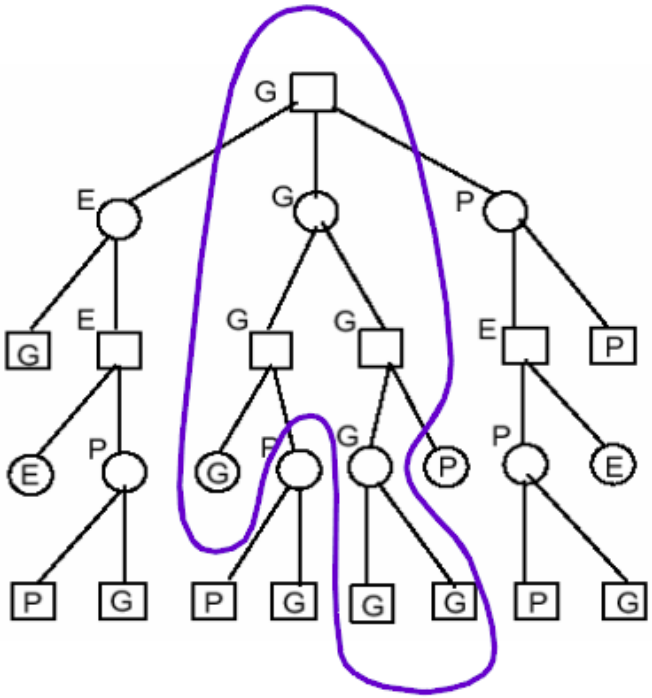
2. Búsqueda Minimax

Árbol de juego sin resolver



G = 1, E = 0, P = -1
Nodos MAX: cuadrados
Nodos MIN: círculos

Árbol de juego resuelto



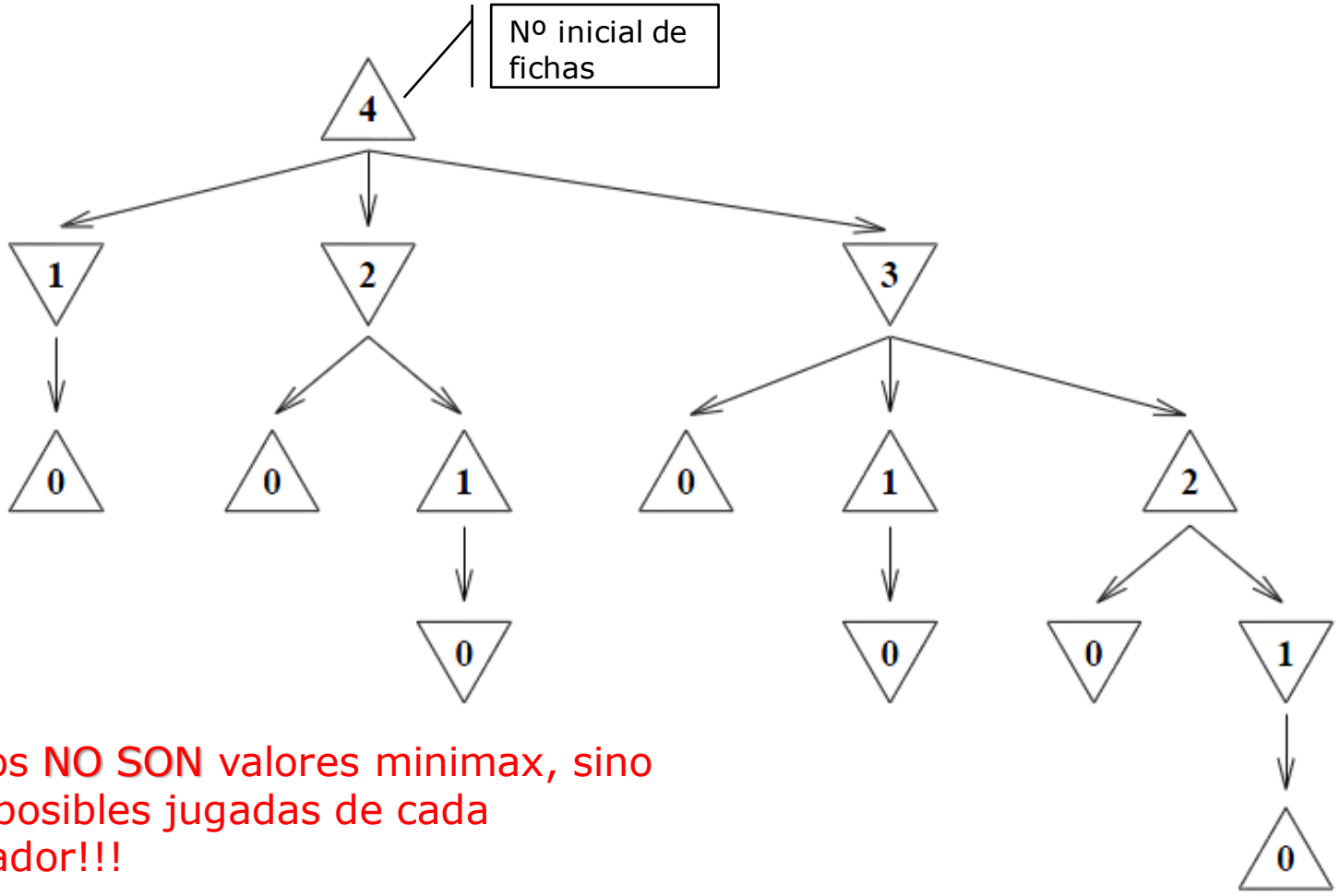
Árbol ganador para MAX

2. Búsqueda Minimax

Nim

- Descripción
  - Una pila con N fichas de la que cada jugador puede tomar alternativamente 1, 2 o 3 fichas.
  - Objetivo: obligar al adversario a coger la última ficha.
- Elementos del juego
  - Estado inicial: numero de fichas al comienzo (N)
  - Estado final: 0 (único)
  - Estados: numero fichas que quedan en la mesa
  - Un jugador gana si en su turno el estado del sistema es el estado final
  - Función de utilidad (para el estado final):
    - 1 si le toca a *MAX*
    - -1 si le toca a *MIN*

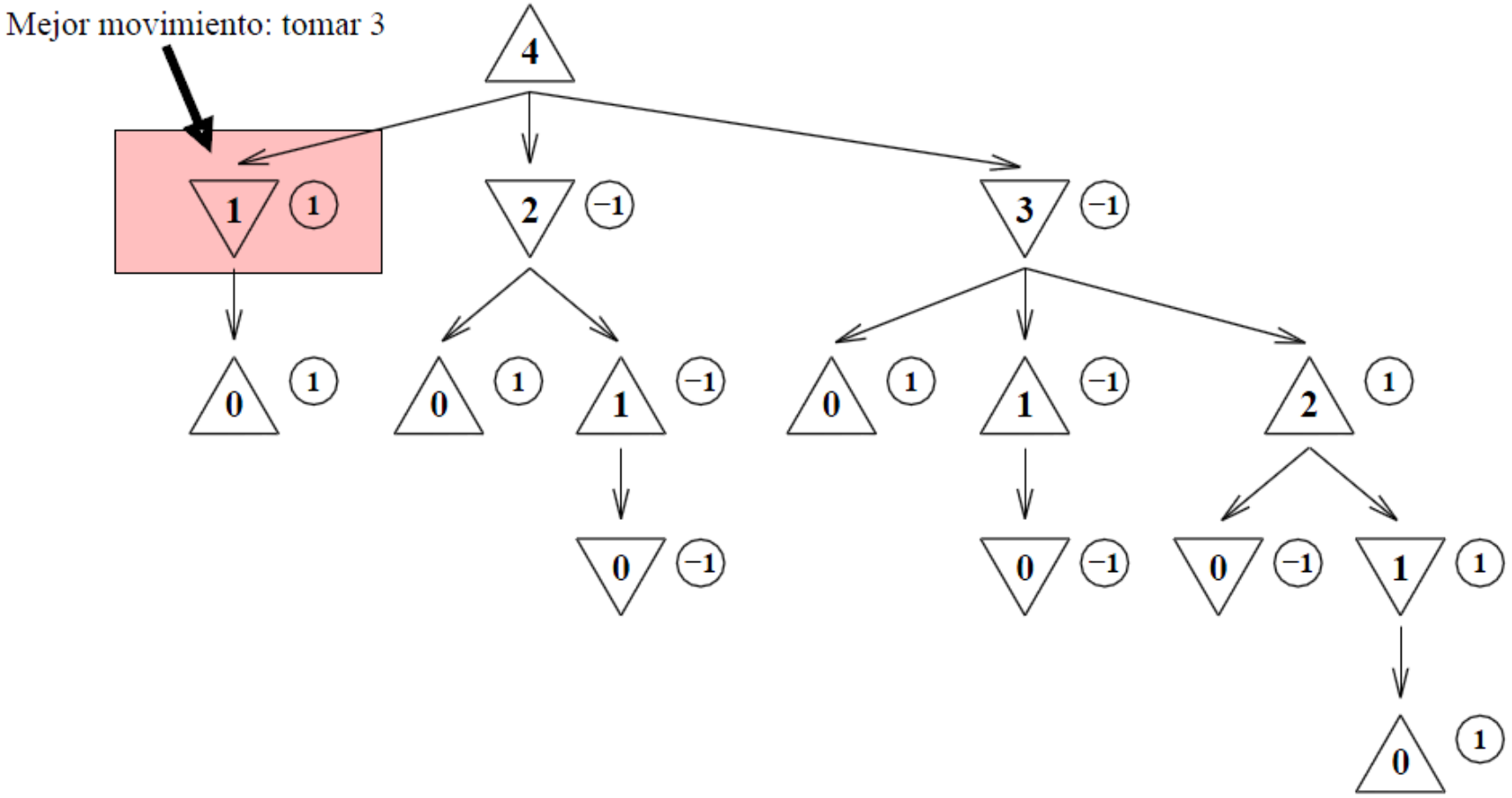
2. Búsqueda Minimax



Estos NO SON valores minimax, sino las posibles jugadas de cada jugador!!!

Resolver el árbol minimax

2. Búsqueda Minimax



2. Búsqueda Minimax

- Propiedades de la búsqueda minimax:
  - Completa: **Si**, si el árbol es finito
  - Complejidad tiempo: $O(b^d)$
  - Complejidad espacio: $O(b^d)$ (igual que la búsqueda primero en profundidad)
  - Optima: **Si**, si el contrincante juega perfectamente

- Para etiquetar la raíz es necesario un árbol de juego completo, pero…

- *Los costes temporal y espacial son excesivos e invalidan la aplicación de minimax a programas de juegos reales*
  - $\sim 10^{21}$siglos para generar el árbol completo de búsqueda de las damas (10^{40} nodos, $3x10^{12}$nodos/seg)
  - $\sim 10^{100}$siglos para generar el árbol completo de búsqueda del ajedrez (10^{120} nodos)

2. Búsqueda Minimax

- Para solventar este problema se han propuesto dos aproximaciones distintas:
  - *Minimax con decisiones imperfectas*: Búsqueda con horizonte limitado y estimación en nodos límite
  - *Poda alfa-beta*

2.1 Minimax con decisiones imperfectas

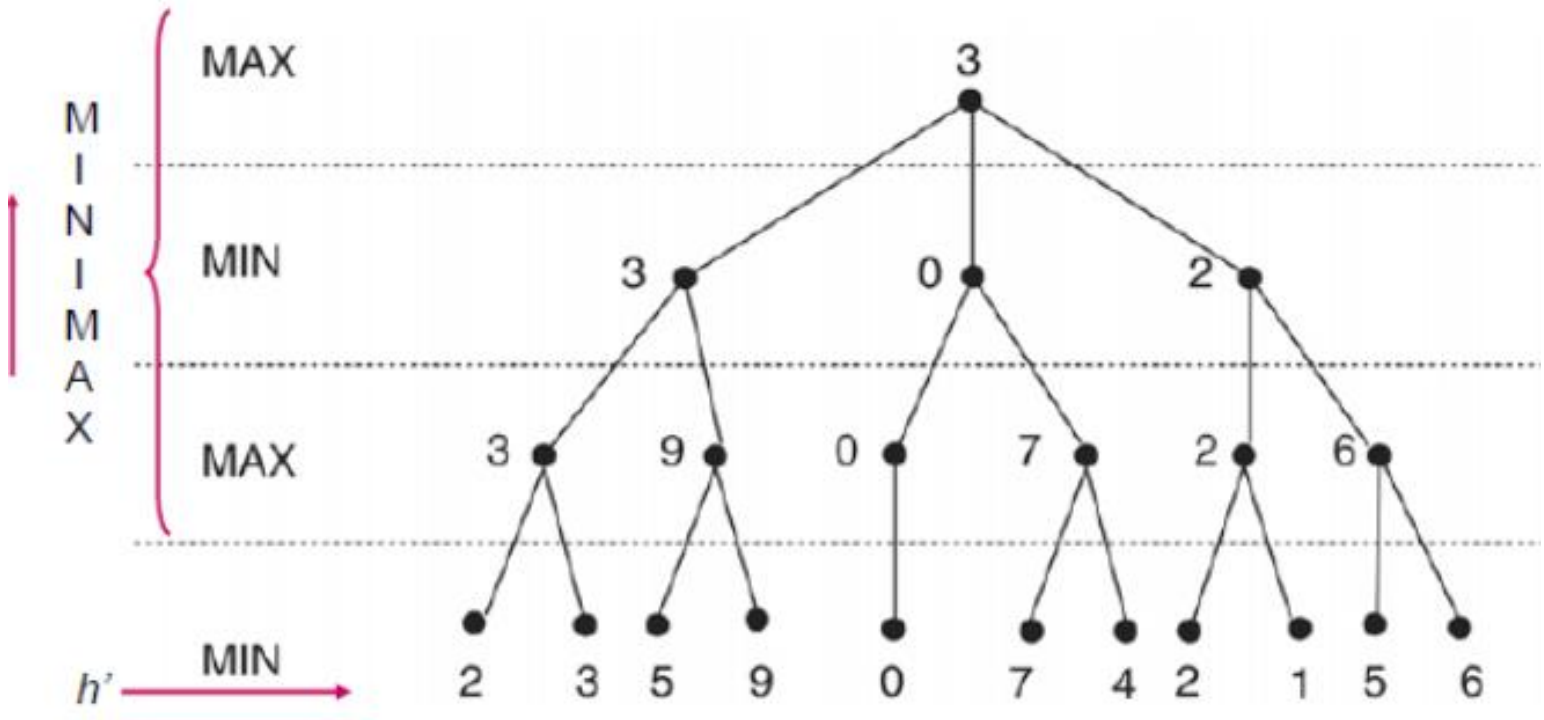
- Interrumpir la búsqueda antes de llegar a los estados terminales ➔ *horizonte limitado*
  - Profundidad *k*: número de pares de movimientos alternativos que se exploran antes de decidir el movimiento que hará MAX
    - La profundidad del conjunto de nodos es *2k* para *MAX* y *2k-1* para *MIN*
  - Las "hojas" de este árbol no necesariamente son finales de partida
    - No siempre se puede asignar valor de *f(n)*
    - Se añade una Función estática de evaluación *h(n):*
      - *H*eurística de la expectativa de ganar de *MAX*
  - Se sustituye el test terminal por un *test de interrupción*
    - Nodos terminales (finales de partida): *f(n)*
    - Nodos profundidad *k* (terminales o no): *h(n)*
  - Los valores se propagan (ascienden) usando el principio minimax

2.1 Minimax con decisiones imperfectas

Valor de la Función de Utilidad en el modelo Minimax con decisiones imperfectas

$$f(n) = \begin{cases} +\infty & \text{si n es una situación ganadora} \\ -\infty & \text{si n es una situación perdedora} \\ 0 & \text{si n es una situación de empate} \\ h(n) & \text{si p = Profundidad-máxima} \\ \text{máx}_{S_i \in S(n)} f(S_i) & \text{si n es nodo MAX y } p < p_{max} \\ \text{mín}_{S_i \in S(n)} f(S_i) & \text{si n es nodo MIN y } p < p_{max} \end{cases}$$

2.1 Minimax con decisiones imperfectas



2.1 Minimax con decisiones imperfectas

- Función Estática de Evaluación
  - Heurística que codifica todo el *conocimiento* que poseemos acerca del juego
    - en el ajedrez se puede dar un *valor material* a cada pieza: 1 al peón, 3 al caballo y al alfil,. . .
  - Dado un estado del juego y un turno, es una *estimación* de las posibilidades de ganar de *MAX* en tal situación
  - Están basadas normalmente en la experiencia previa

- Características de una función estática de evaluación:
  - Debe acercarse a la función de utilidad en los estados terminales
  - Su calculo no debe ser muy costoso
  - Debe reflejar de forma precisa la probabilidad de resultar ganador
  - *h(n)* grande y positivo es bueno para *MAX* y malo para *MIN*

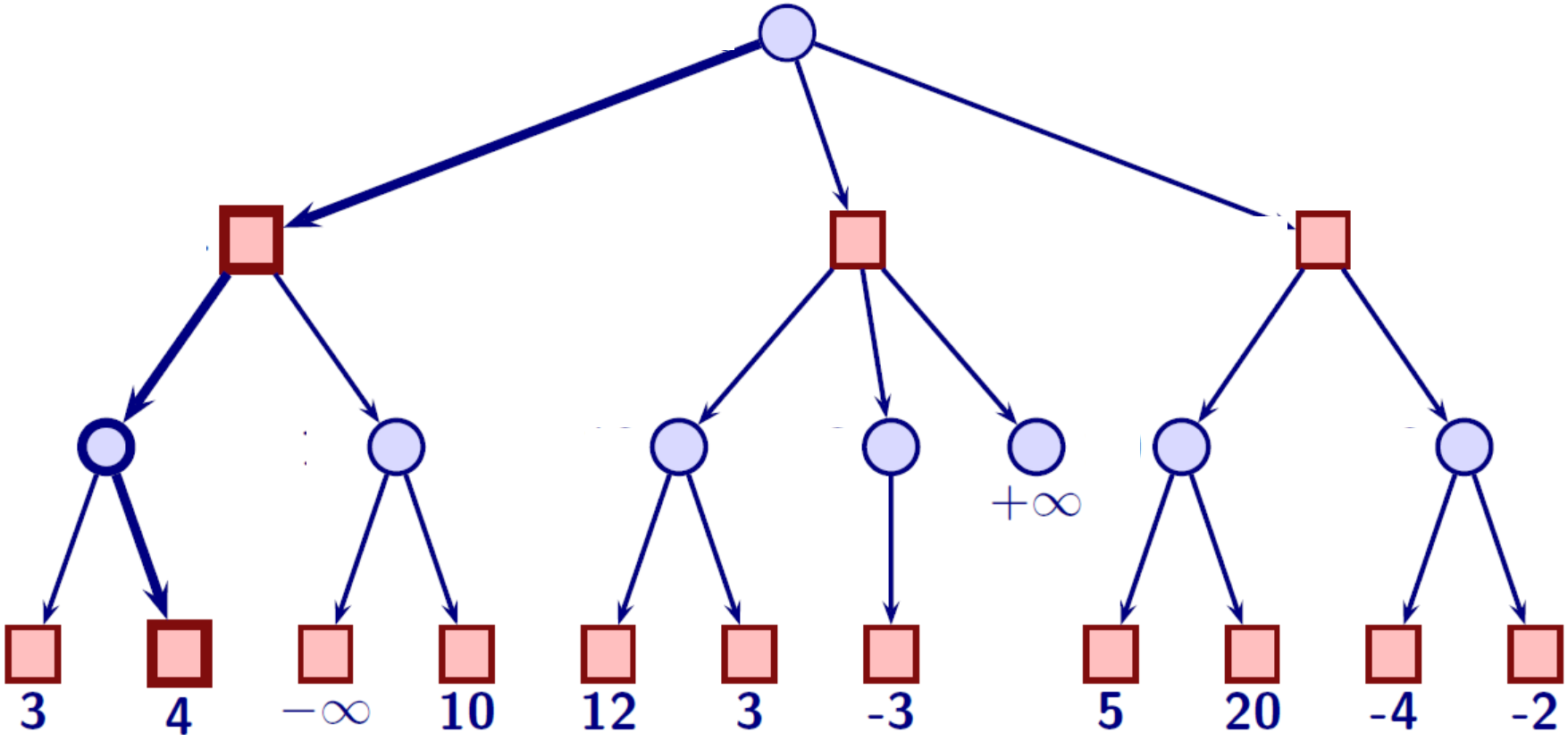
2.1 Minimax con decisiones imperfectas

- Se asume que el valor ascendido hasta la raíz, obtenido en una profundidad *k*, es una estimación mejor que aplicar directamente la función de estimación a los sucesores del nodo raíz

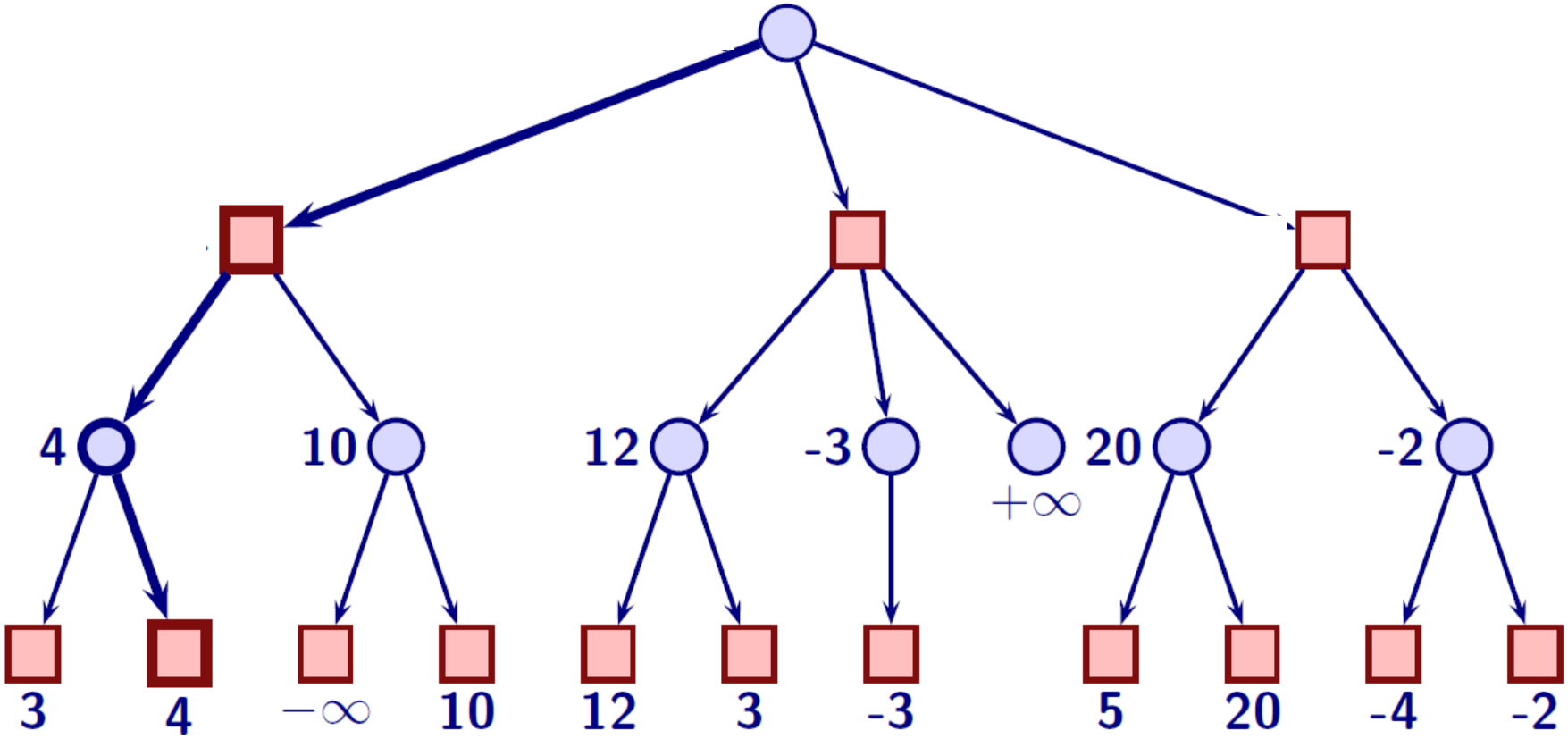
- A la raíz le llega la medida heurística del mejor estado alcanzable en *k* movimientos
  - Cuanto mayor sea el horizonte, más seguros son los elementos de decisión para elegir la mejor jugada
  - Así se prevén las consecuencias de la jugada a más largo plazo
  - Pero no hay garantías

- Lo importante es la comparación del valor entre los estados

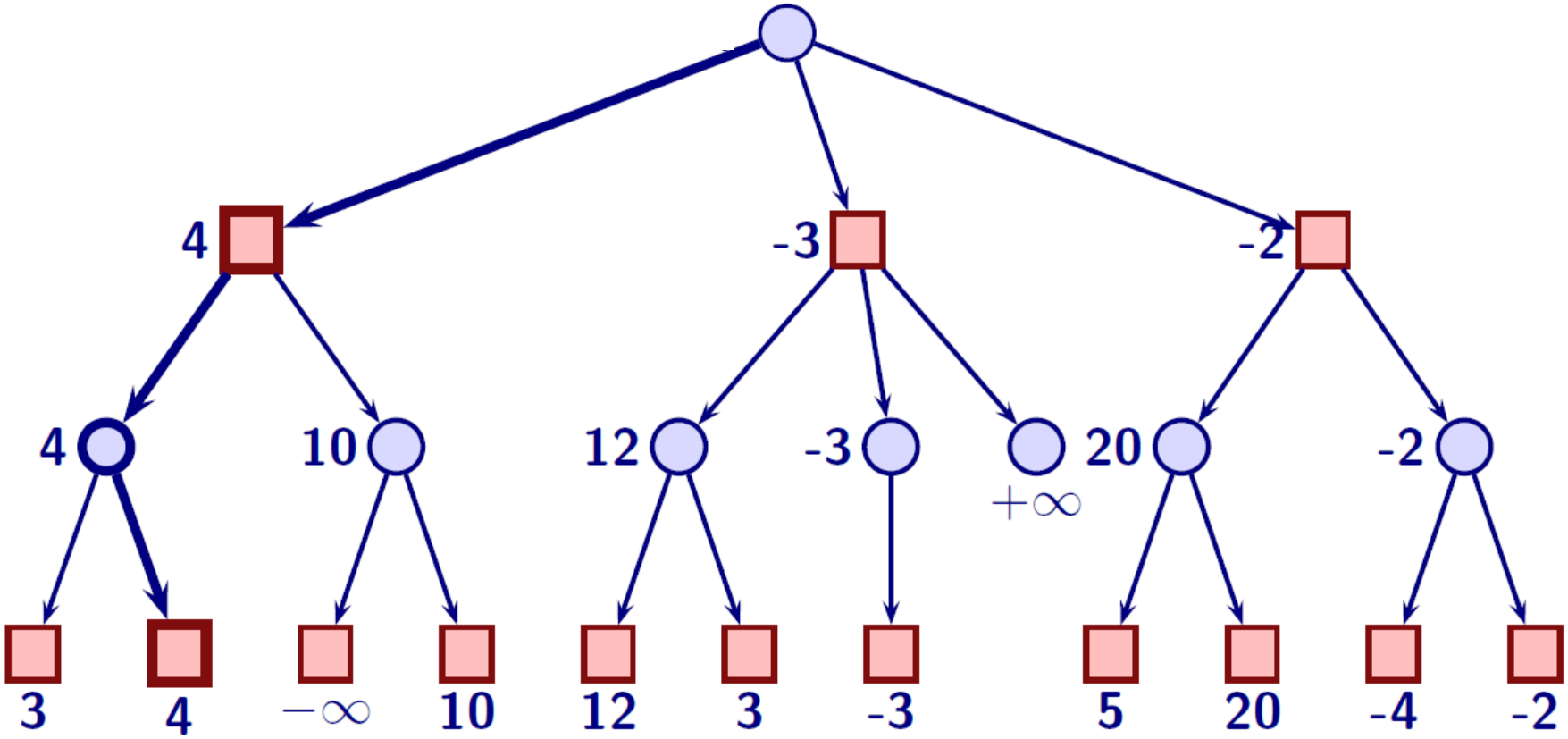
2.1 Minimax con decisiones imperfectas



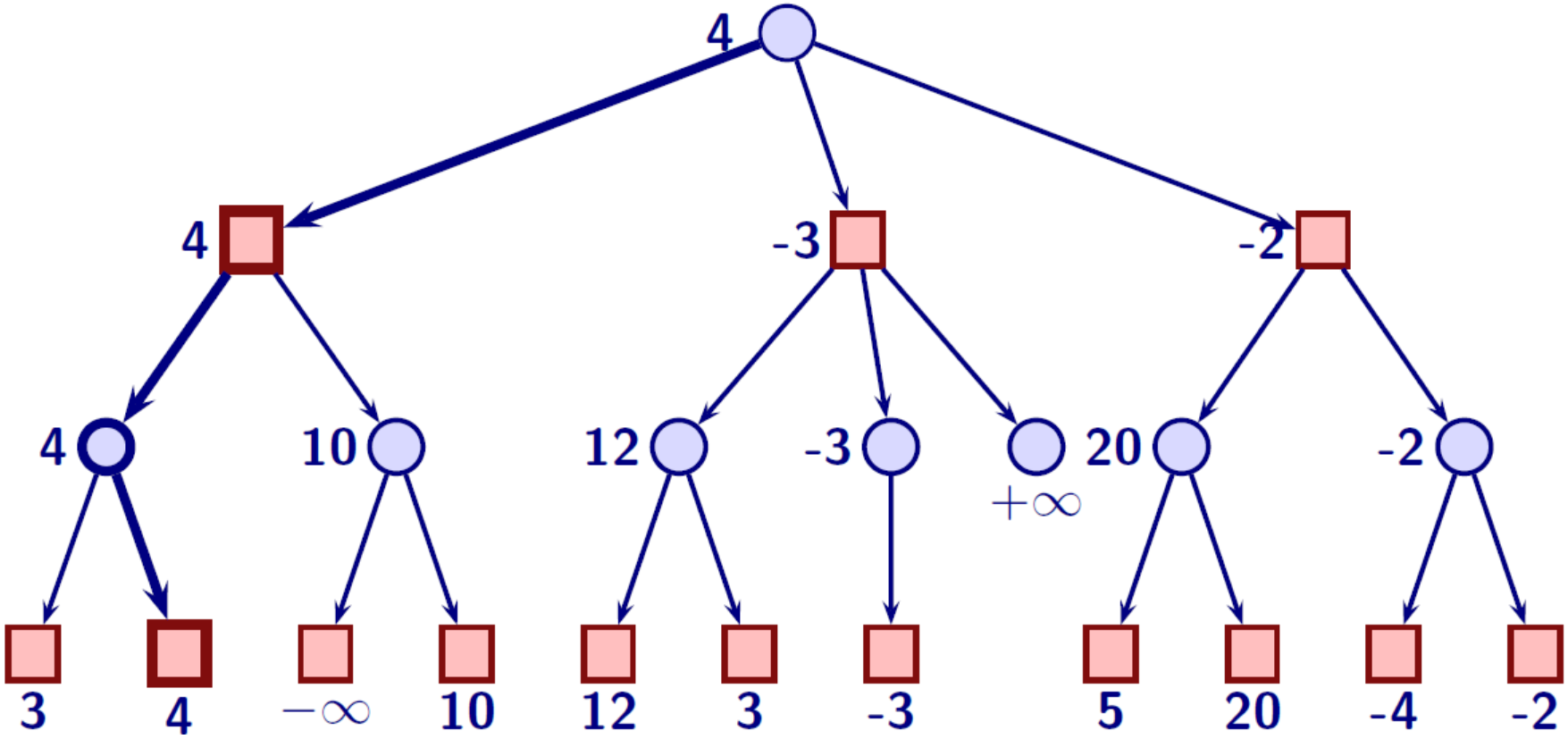
2.1 Minimax con decisiones imperfectas



2.1 Minimax con decisiones imperfectas



2.1 Minimax con decisiones imperfectas



2.1 Minimax con decisiones imperfectas

Tres en raya

- Descripción
  - En un tablero de 3x3 alternativamente un jugador pone una ficha **X** y el otro jugador pone una ficha **O**
  - Gana el que consigue colocar tres de sus fichas en línea (vertical, horizontal o diagonal)
- Elementos del juego
  - Estado inicial: tablero vacío + ficha de salida
  - Estados finales: tableros completos o con línea ganadora
  - Estados: tablero + ficha que se pondrá a continuación
  - Estados ganadores para un jugador: estados finales en los que no le toca poner
- Movimientos:
  - 9 movimientos posibles, uno por casilla
  - Aplicable si la casilla no esta ocupada

2.1 Minimax con decisiones imperfectas

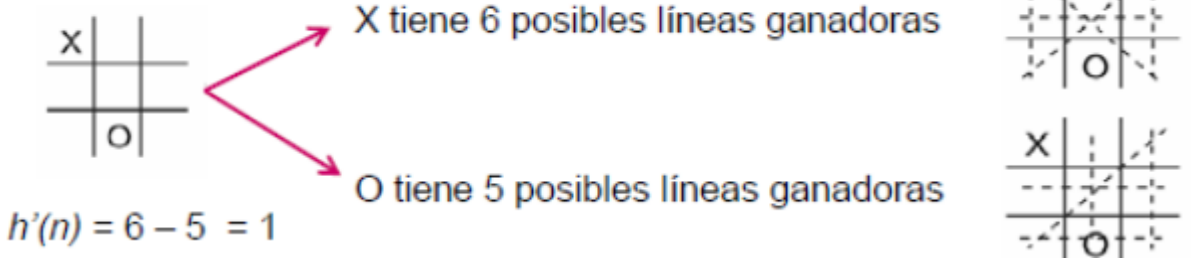
- Función de utilidad:
  - +∞ si es ganador para *MAX*
  - 0 si es tablas
  - -∞ si es ganador para *MIN*

- Función Estática de Evaluación (heurística):
  - Diferencia entre el numero de posibles líneas ganadoras para *MAX* y el numero de posibles líneas ganadoras para *MIN*
  - Líneas ganadoras: Número de filas, columnas y diagonales que se pueden completar, incluyendo las vacías.

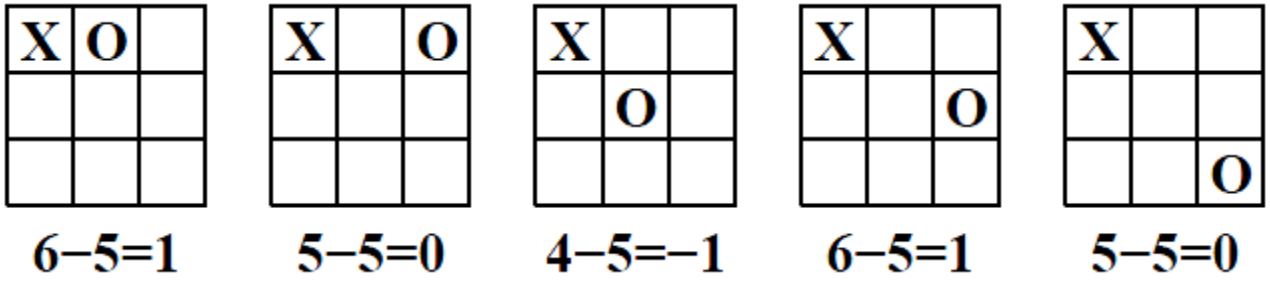
$$h(n) = Líneas(MAX) - Líneas(MIN)$$

2.1 Minimax con decisiones imperfectas

- Si *MAX* juega con X

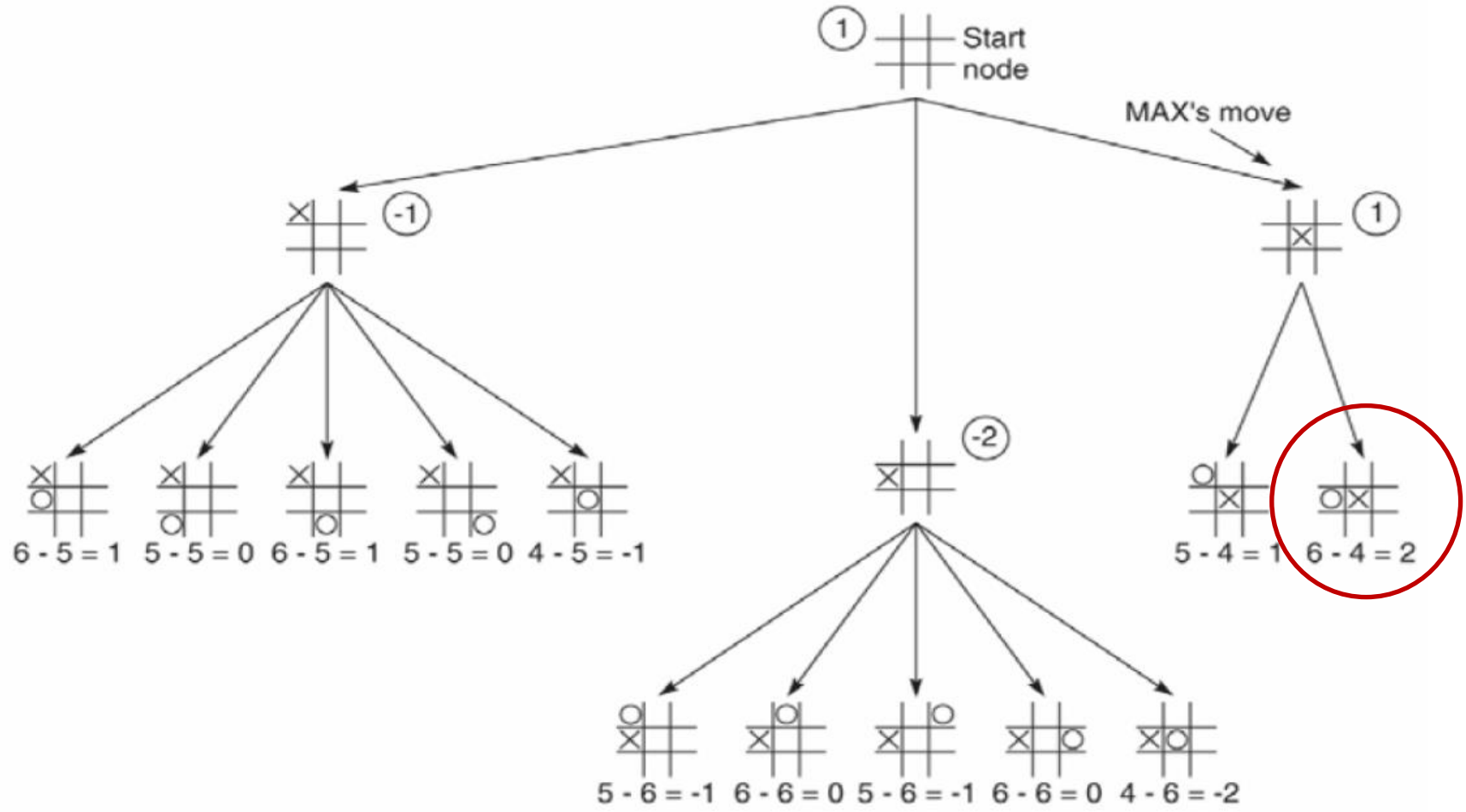


- Para otros casos:



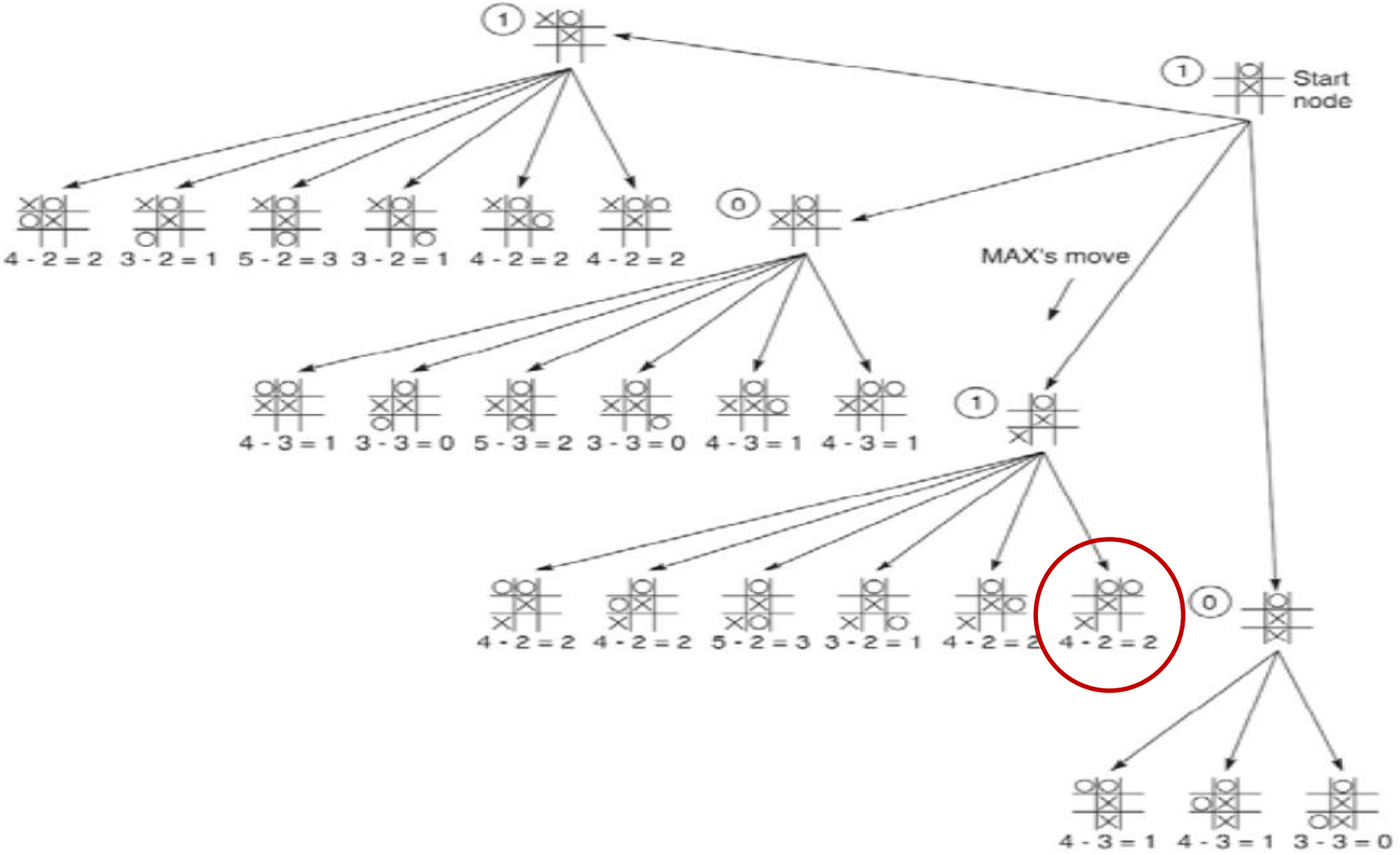
2.1 Minimax con decisiones imperfectas

- Movimiento inicial (con profundidad k=1 para *MAX*)



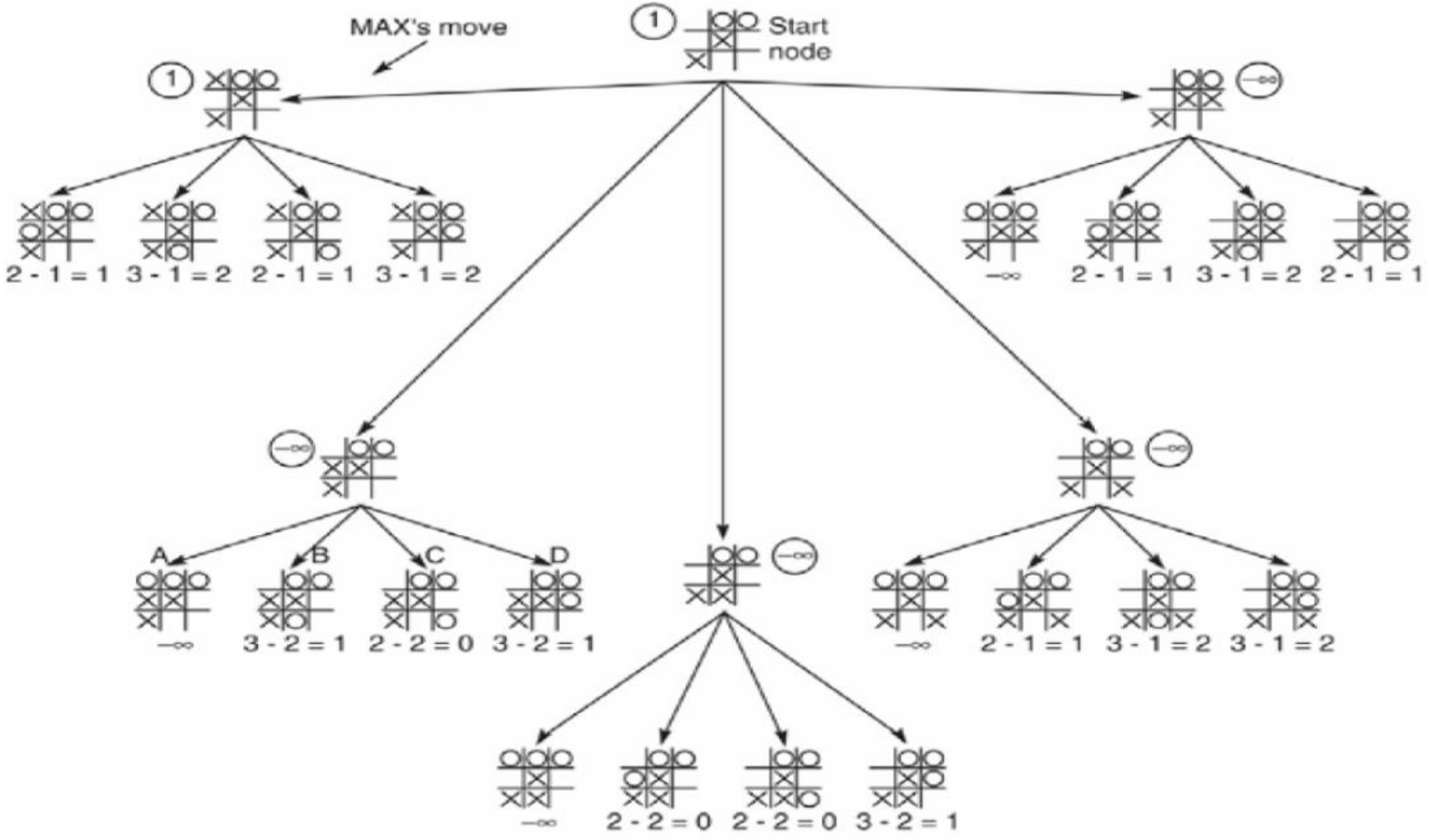
2.1 Minimax con decisiones imperfectas

- Segundo movimiento de *MAX* (k=1)



2.1 Minimax con decisiones imperfectas

- Tercer movimiento de *MAX* (k=1)

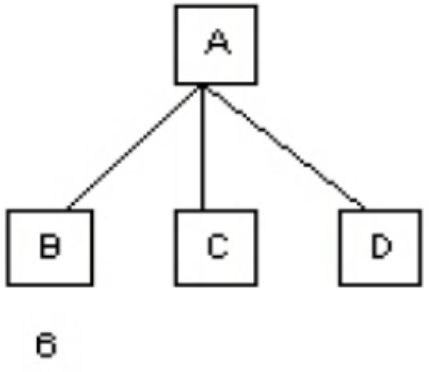


2.1 Minimax con decisiones imperfectas

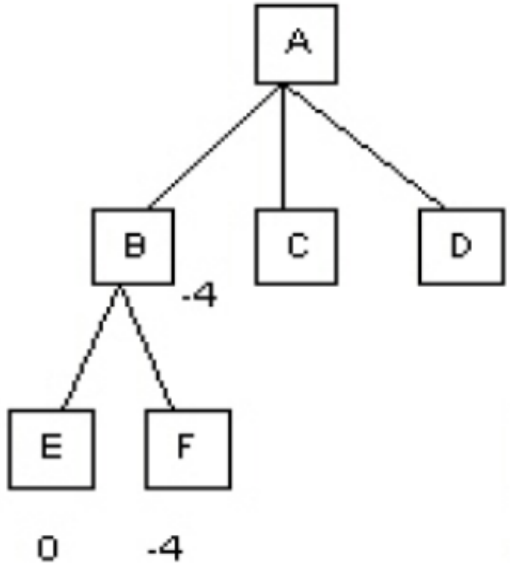
- Existen distintas opciones para interrumpir la búsqueda, pero todas tienen problemas derivados del *efecto horizonte*

  - Simple: llegar al límite máximo de profundidad del árbol de acuerdo en el tiempo disponible
  - Astuta:
    - Distintas búsquedas por profundización iterativa. Se devuelve la búsqueda más profunda que se haya conseguido en el tiempo disponible
    - Búsquedas secundarias: búsquedas parciales en los nodos en los que se detecten problemas
      - Posiciones en reposo: No se puede cortar si la posición no es estable. *h(n)* debe ser aplicada solo a posiciones que no cambien bruscamente de valor después
        - Ajedrez: que un jugador capture una pieza importante puede resultar en ventaja para el otro (*sacrificios*)
      - Extensiones singulares: Si un nodo hoja es muy diferente a sus hermanos, el nodo se expande una capa más
        - Ajedrez: por situación de captura inminente (*jugada forzada*)

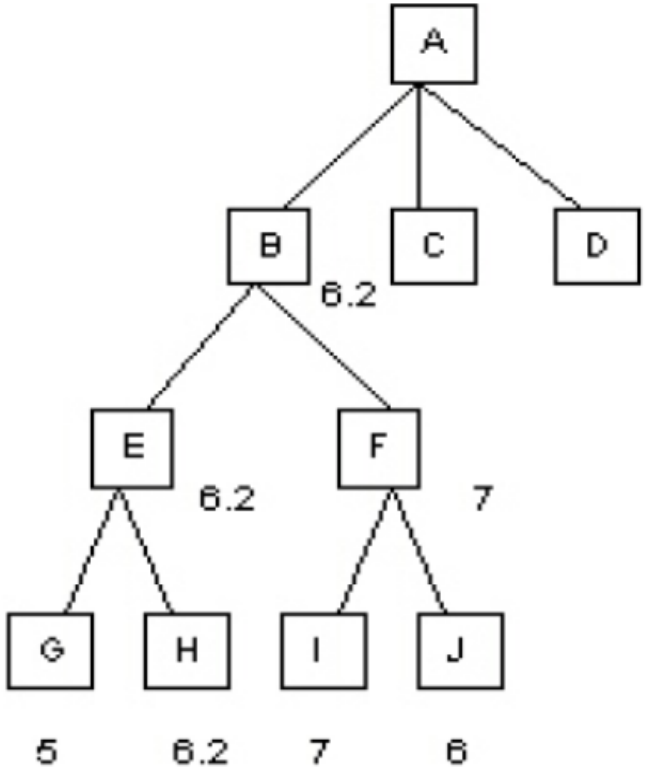
2.1 Minimax con decisiones imperfectas



Si exploramos un nivel, B es la mejor opción

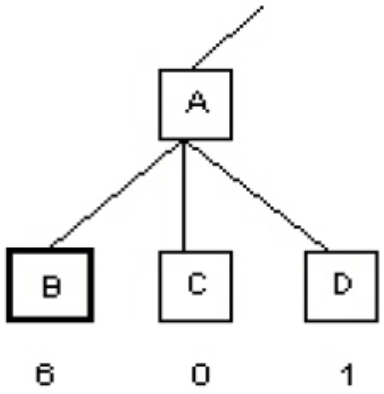


La cosa cambia si exploramos un nivel más

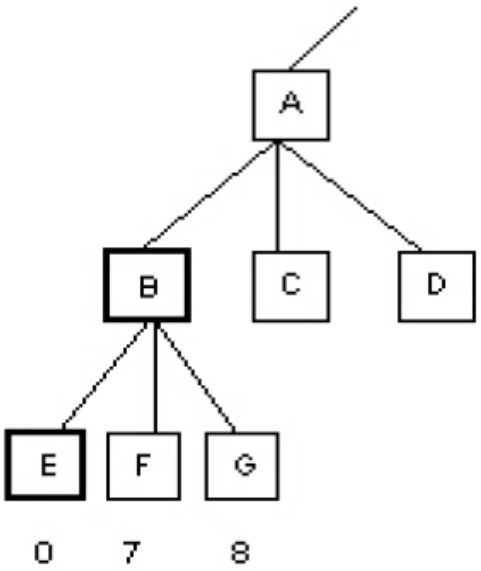


Posición de reposo: la situación se parece a la inicial

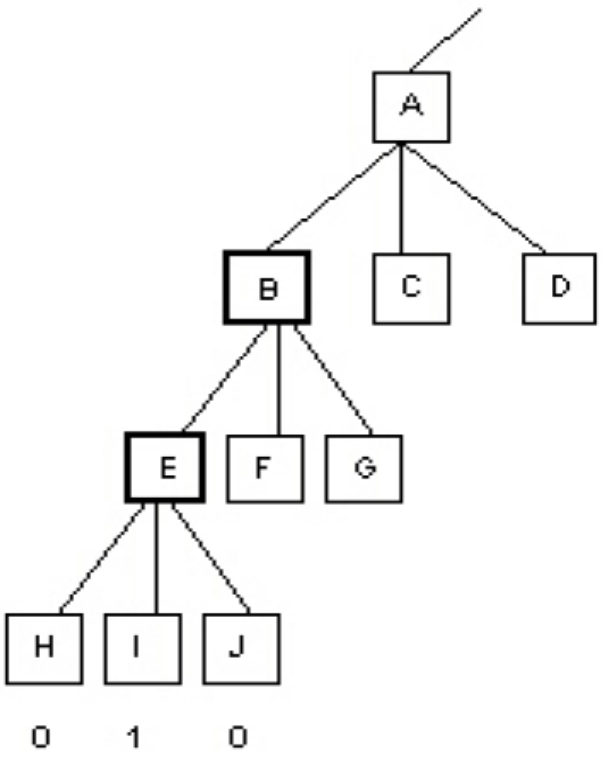
2.1 Minimax con decisiones imperfectas



B tiene un valor superior a sus hermanos.
¿Captura inminente?



¿Captura por parte del oponente?



No hay captura: valores en un intervalo estrecho. Se interrumpe la búsqueda secundaria

2.2 Poda Alfa-Beta

- Es posible calcular la decisión minimax correcta sin explorar todos los nodos del árbol de búsqueda
- El proceso por el cual determinadas ramas no se consideran en la búsqueda se denomina **poda** del árbol de búsqueda

- Optimización del minimax: *podar las ramas que no proporcionen mejoras sobre el mejor camino hasta el momento*
  - Poda α–β ($\alpha-\beta$ *pruning*)
  - Elimina los nodos que ofrezcan un valor menor (para *MAX*) o mayor (para *MIN*) que los que hemos encontrado ya
  - Sin comprobar cuan malo es el sub-árbol
  - Convierte un factor de ramificación de b en √b
  - Búsqueda Primero en Profundidad

2.2 Poda Alfa-Beta

- α = valor de la mejor jugada hasta el momento para *MAX*
  - guarda el valor máximo de los sucesores de *MAX* encontrados hasta el momento
  - cota inferior para el valor final que pueda alcanzar el nodo *MAX* (lo peor que le podría ir a *MAX*)
  - Valor monótono no-decreciente
  - $\alpha_0 = -\infty$

- β = valor de la mejor jugada hasta el momento para *MIN*
  - guarda el valor mínimo de los sucesores de *MIN* encontrados hasta el momento
  - cota superior para el valor final que pueda alcanzar el nodo *MIN* (lo mejor que le podría ir a MIN)
  - Valor monótono no-creciente
  - $\beta_0 = +\infty$

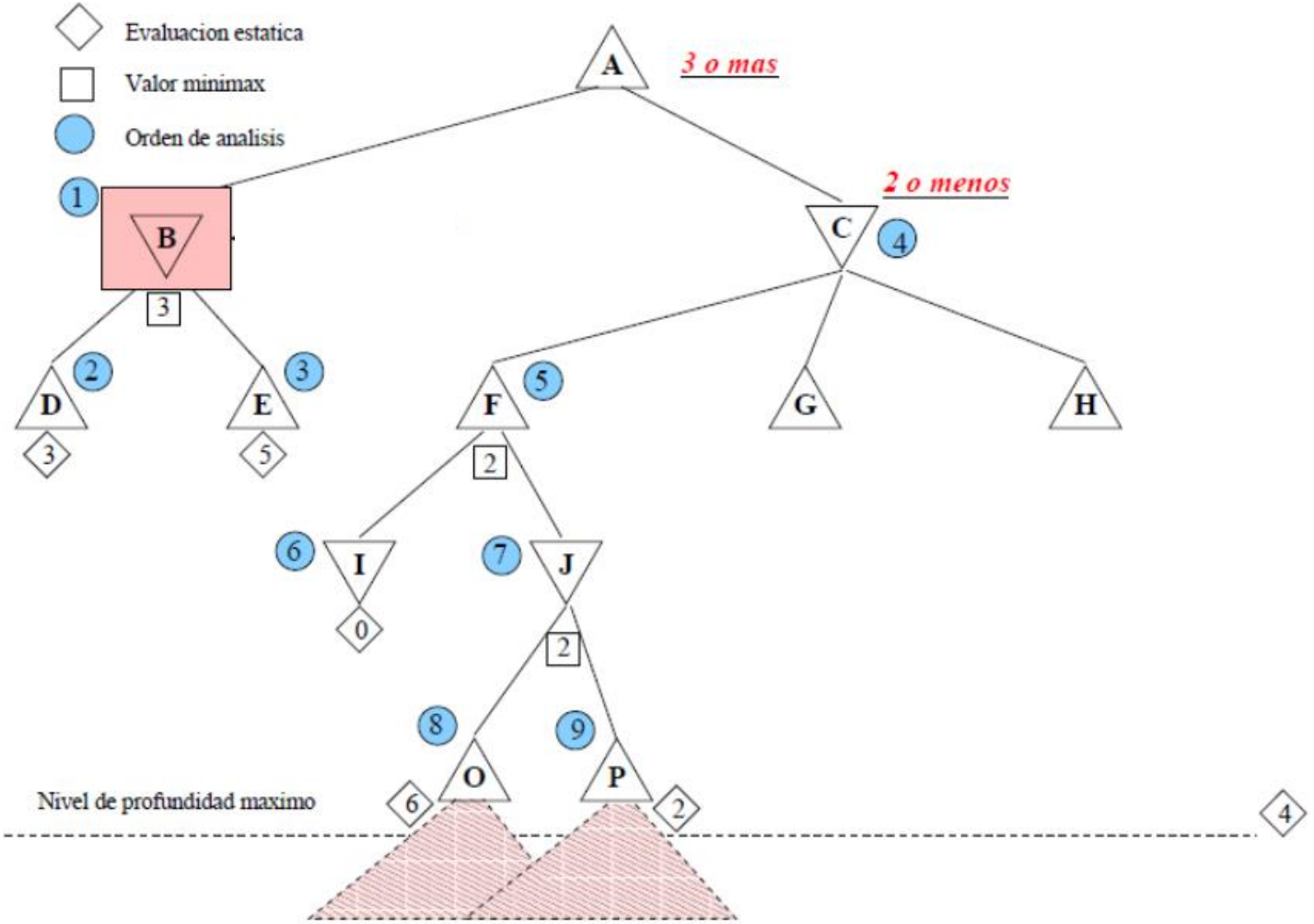
2.2 Poda Alfa-Beta

- Nodos *MAX*: Poda α
  - α es el valor actual del nodo (que tendrá eso o más) y β es el valor actual del padre (que tendrá eso o menos)
  - Poda en (n) cuando $\alpha_n \geq \beta_{n-1}$ porque ya hay jugadas mejores para *MIN*
  - No hace falta analizar los restantes sucesores del nodo

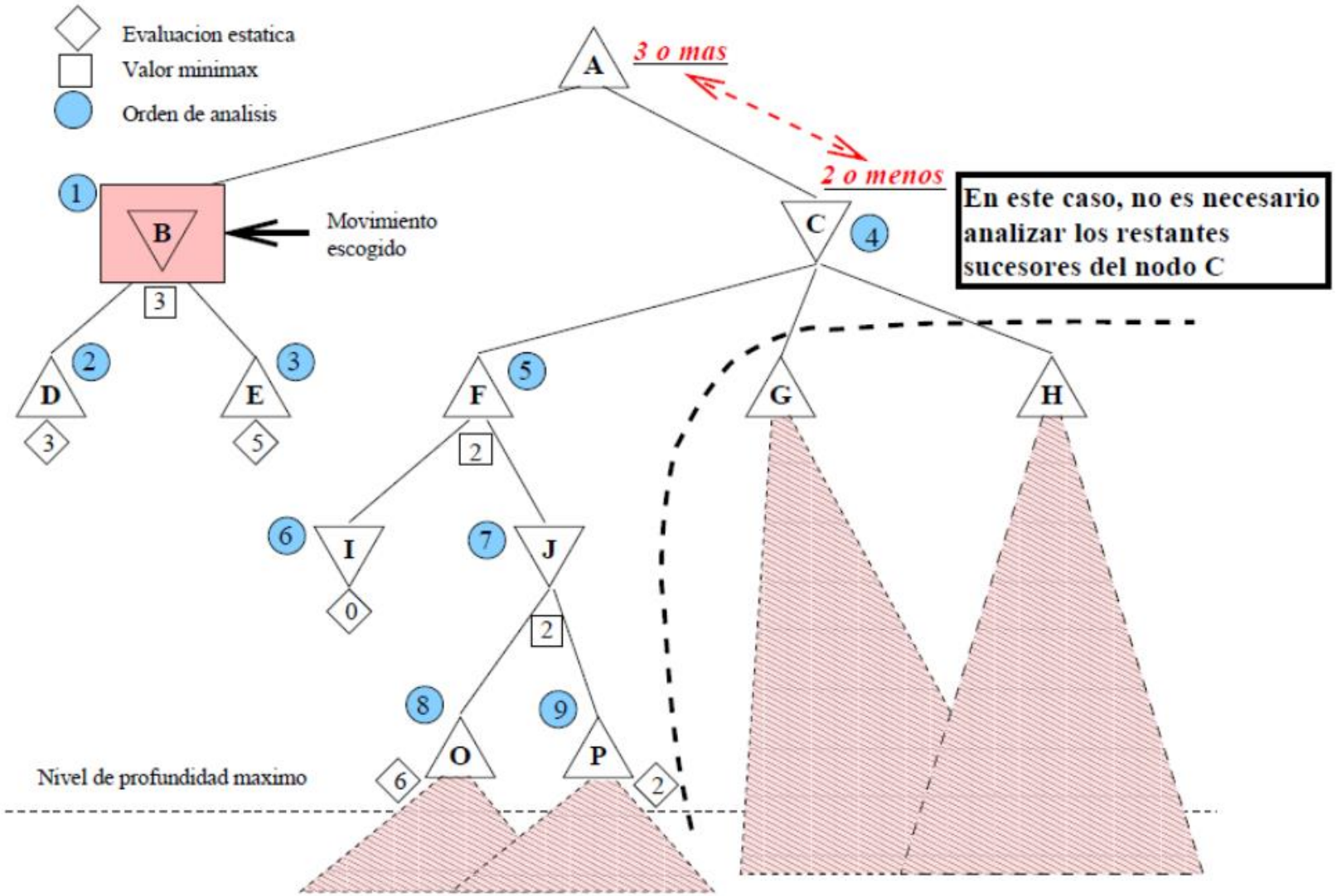
- Nodos *MIN*: Poda β
  - β es el valor actual del nodo (que tendrá eso o menos) y α es el valor actual del padre (que tendrá eso o más)
  - Poda en (n): cuando $\beta_n \leq \alpha_{n-1}$ porque ya hay jugadas mejores para *MAX*
  - No hace falta analizar los restantes sucesores del nodo

2.2 Poda Alfa-Beta

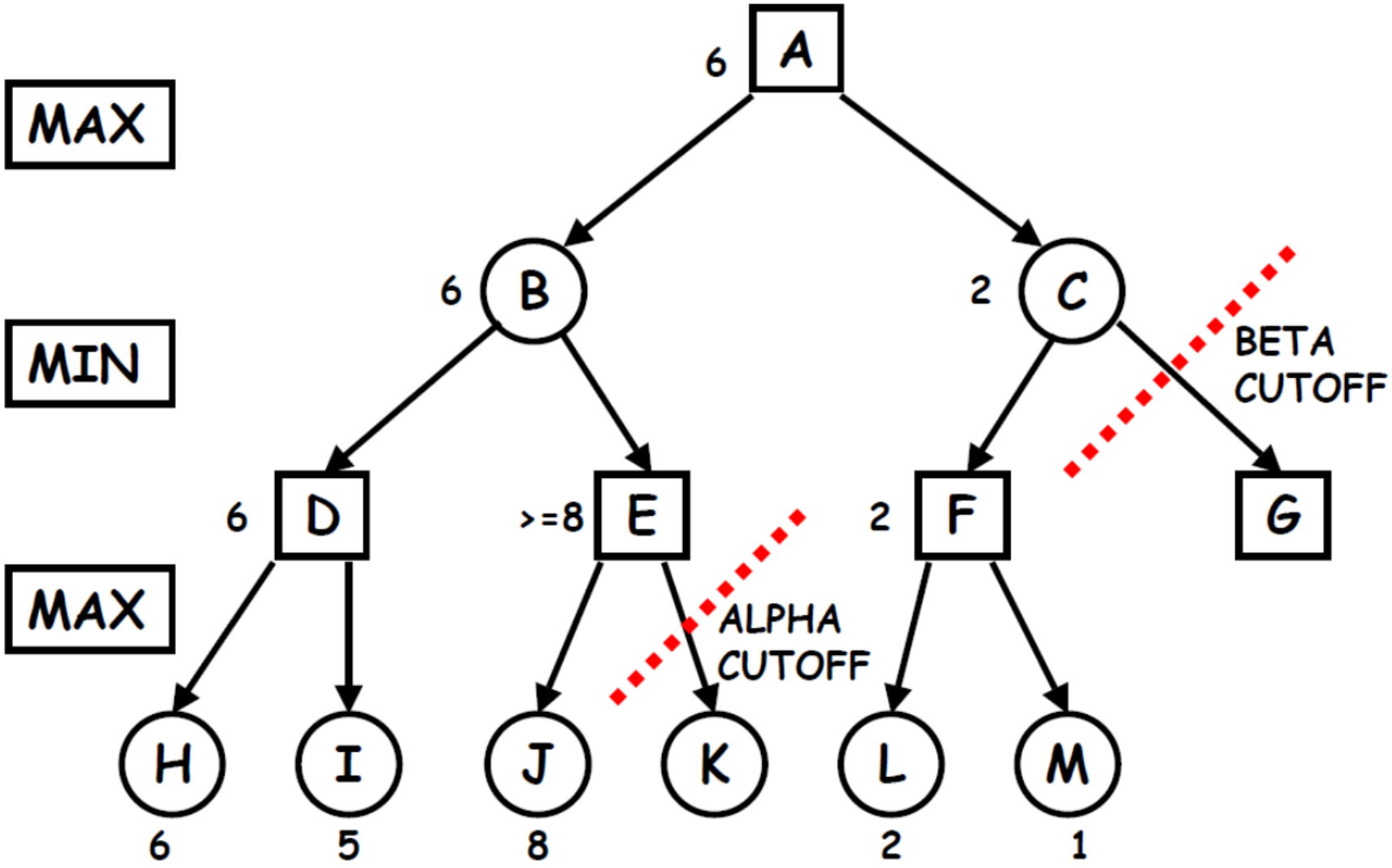
- Situación justo después de analizar el subárbol F:



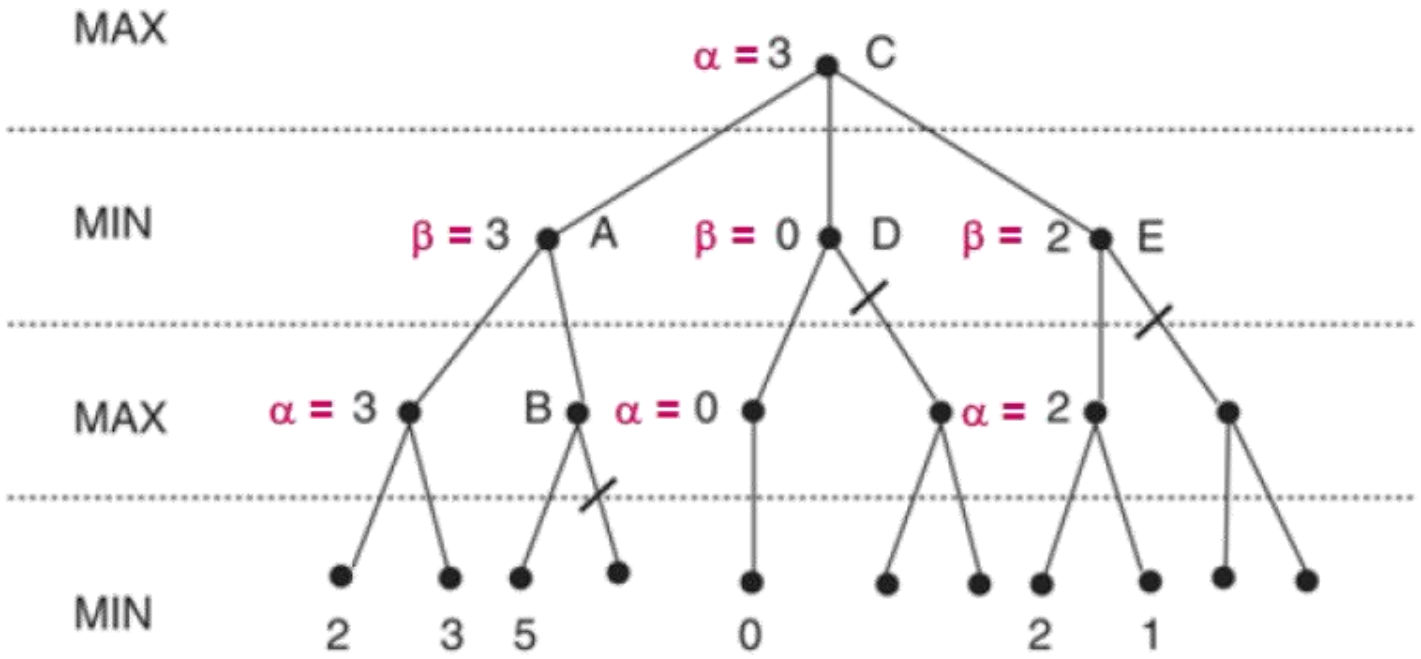
2.2 Poda Alfa-Beta



2.2 Poda Alfa-Beta



2.2 Poda Alfa-Beta

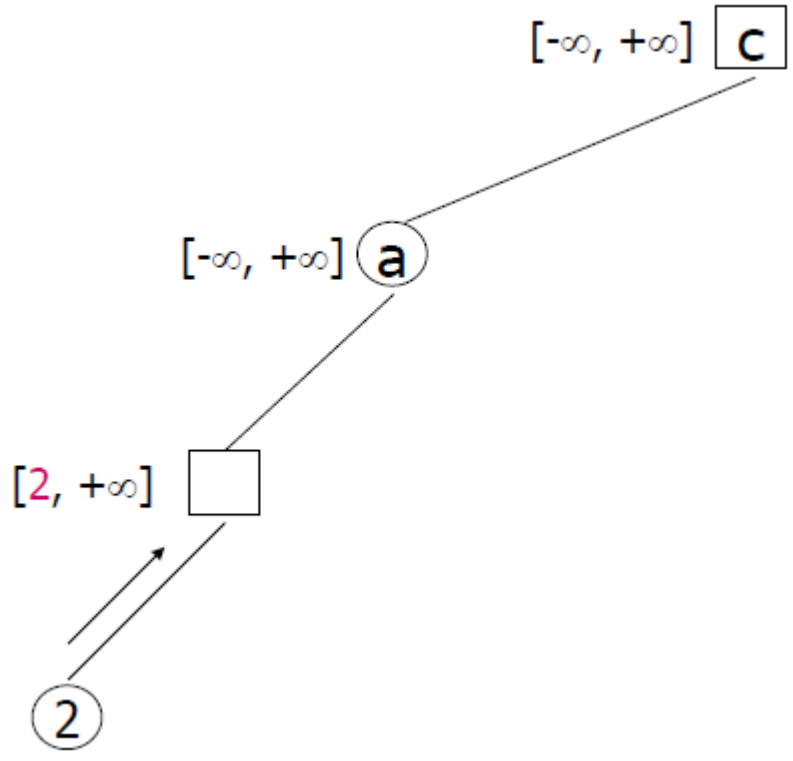


A ≤ 3 ➔ β_A= 3
B ≥ 5; 5 > 3 ➔ B es β-podado

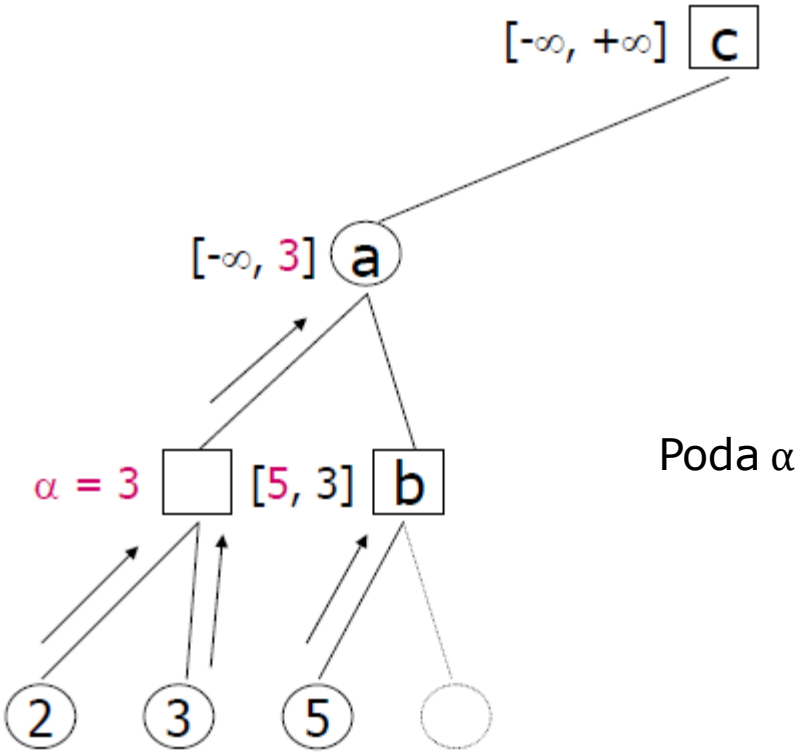
C ≥ 3 ➔ α_C=3
D ≤ 0; 0 < 3 ➔ D es α-podado
E ≤ 2; 2 < 3 ➔ E es α-podado

Valor minimax de C = 3

2.2 Poda Alfa-Beta

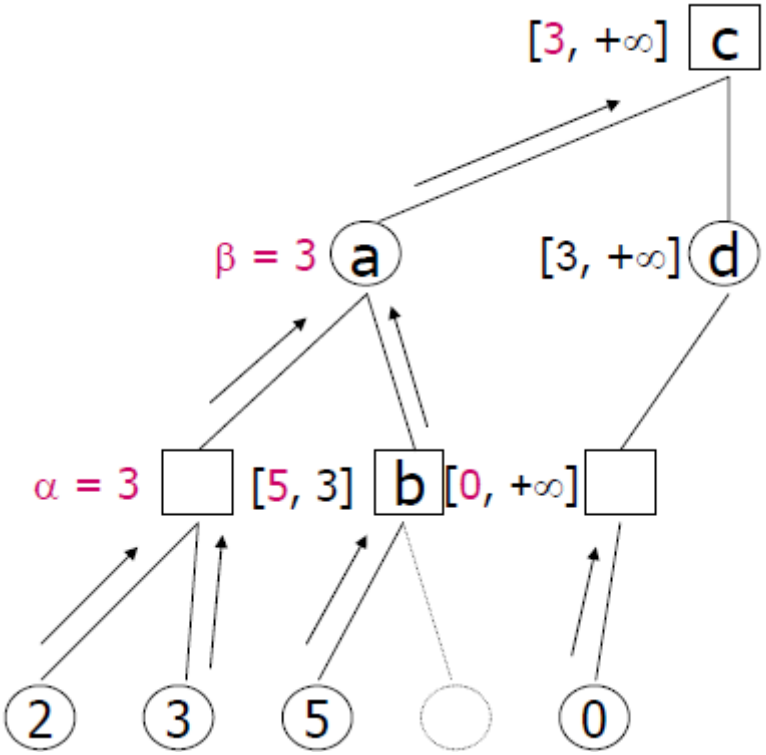


2.2 Poda Alfa-Beta

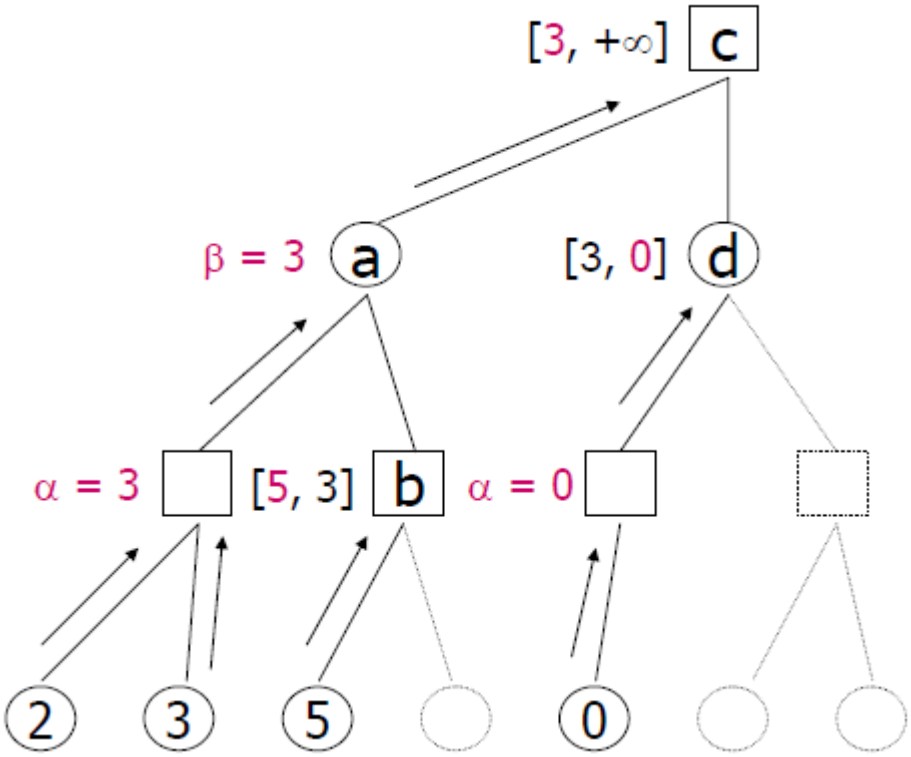


Poda α

2.2 Poda Alfa-Beta

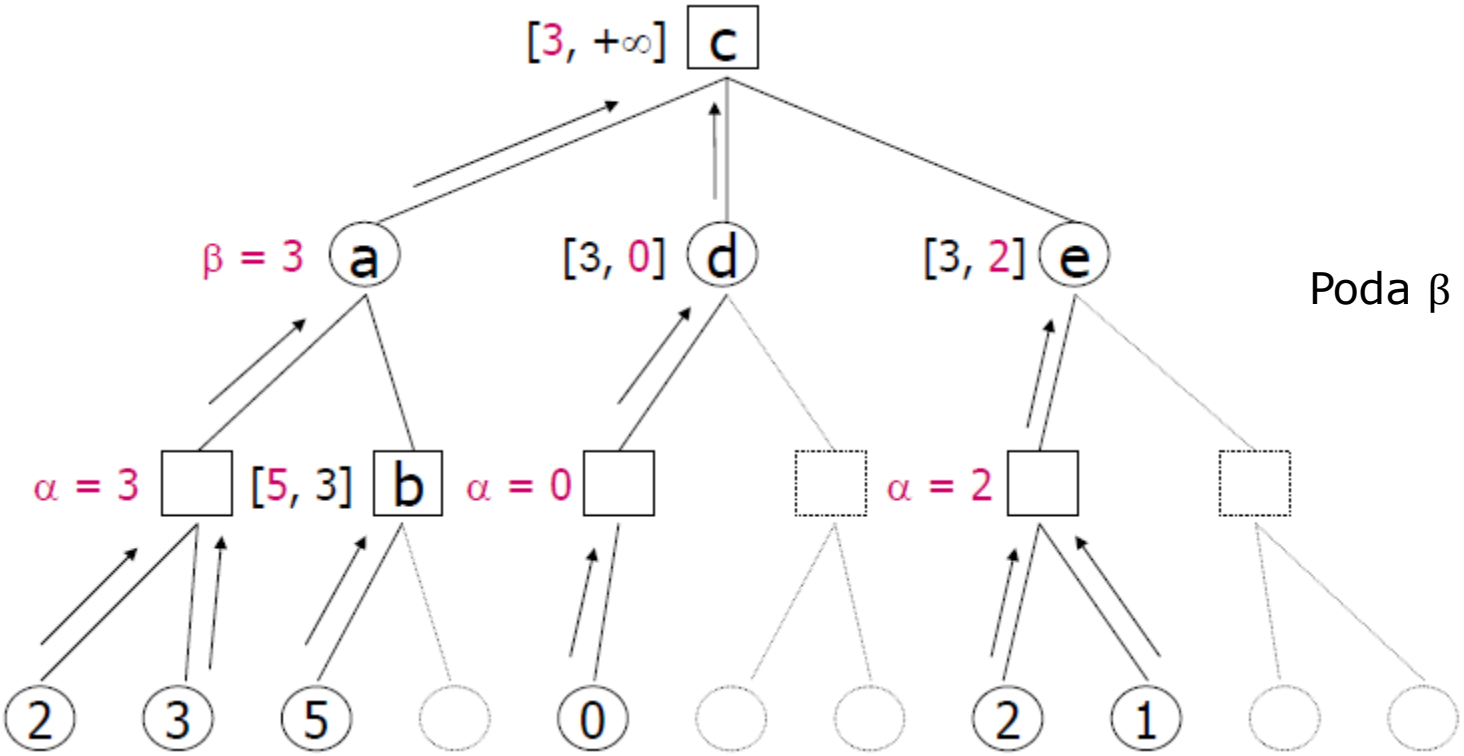


2.2 Poda Alfa-Beta

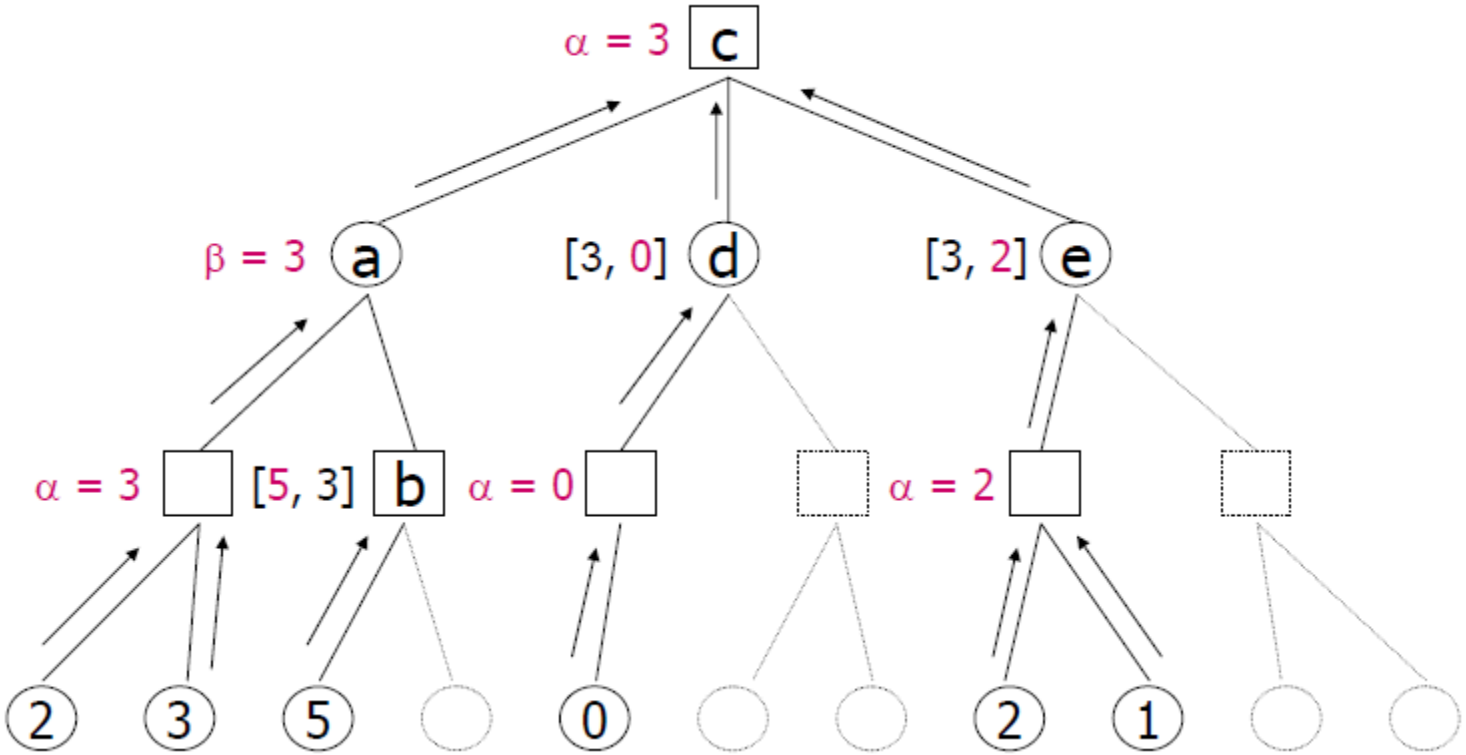


Poda β

2.2 Poda Alfa-Beta



2.2 Poda Alfa-Beta



2.2 Poda Alfa-Beta

- Complejidad:
  - Complejidad en tiempo: $O(b^{3d/4})$
  - Complejidad en espacio: $O(bd)$.

- La poda $\alpha-\beta$, aplicada a árboles minimax, devuelve el mismo resultado que minimax, pero más eficientemente
  - Si los sucesores se exploran en orden de valor minimax (creciente o decreciente dependiendo de si es un nodo MIN o MAX) se produce la máxima poda
    - La complejidad temporal se reduce a $O(b^{d/2})$
    - ¡¡*En el mismo tiempo se puedan considerar el doble de jugadas que en minimax*!!
    - La máquina es más competitiva
- Sitio web para probar Minimax y Poda $\alpha-\beta$:

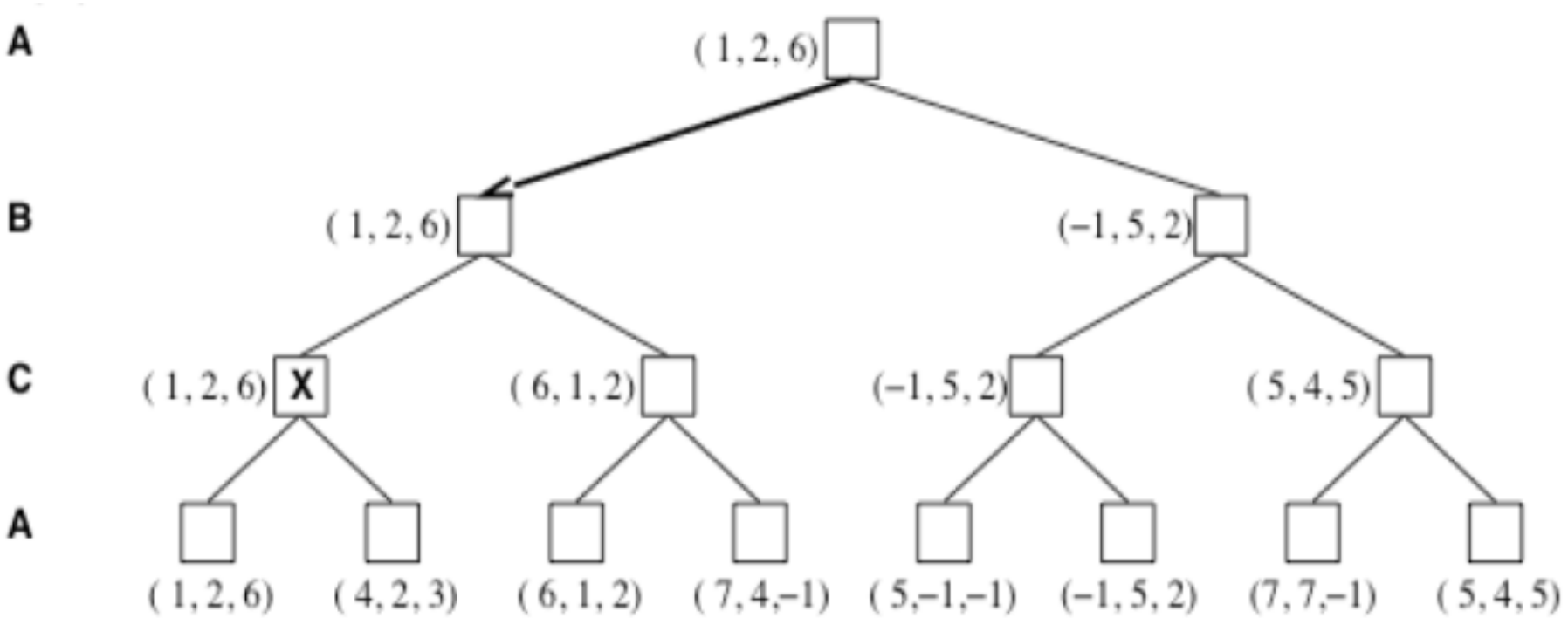
https://raphsilva.github.io/utilities/minimax_simulator/#

1. Búsqueda con adversarios
2. Búsqueda Minimax
    1. Minimax con decisiones imperfectas
    2. Poda Alfa-Beta
3. Otros tipos de juegos
4. Estado del arte en problemas de juegos

3. Otros tipos de juegos

Juegos con varios jugadores

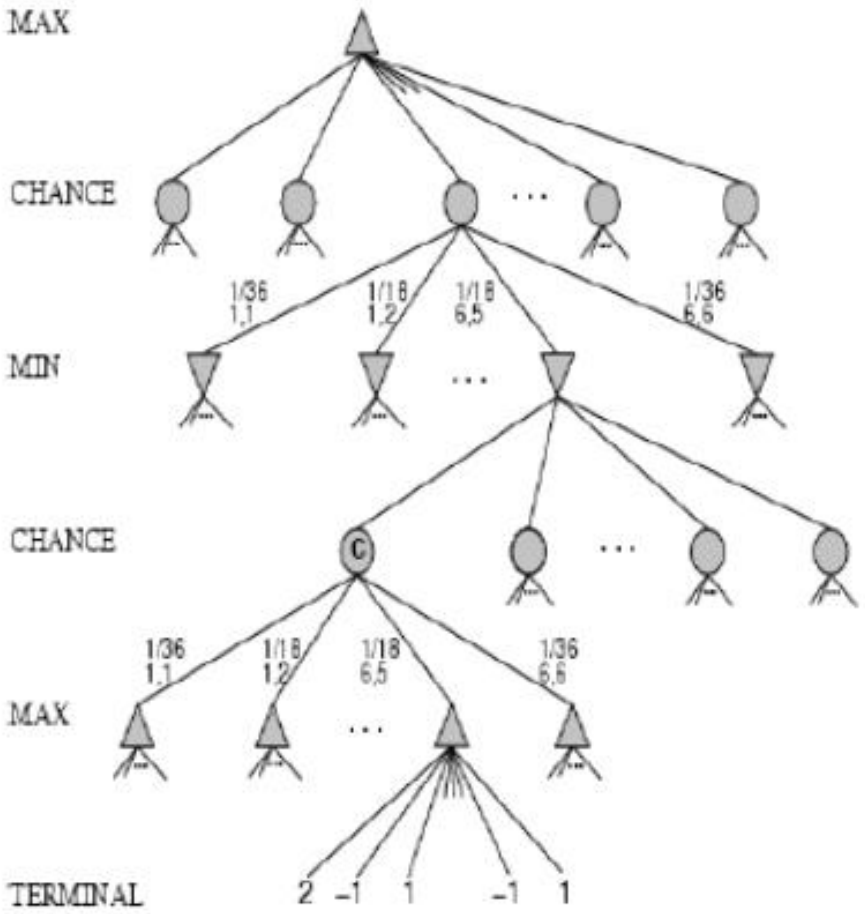
- La función de utilidad se convierte en un vector de valores



3. Otros tipos de juegos

Juegos con azar

- Se modelan probabilidades de jugada mediante un jugador ficticio (*CHANCE*)
- Se alternan nodos *MAX* y *MIN* con *CHANCE*
- El valor de la función de utilidad es aproximado
- Muy complejo



3. Otros tipos de juegos

Juegos con suma no nula

- Minimax se puede aplicar a juegos con suma cero y en los que el oponente juega a ganar
  - La ganancia o pérdida de MAX se equilibra exactamente con las pérdidas o ganancias de MIN

- Sin embargo, muchas de las situaciones del mundo real habitualmente tienen suma no nula: los participantes pueden beneficiarse o perder al mismo tiempo
  - Ejemplos: ciclismo, actividades económicas, la guerra
  - En estas situaciones, la conclusión es que resulta mejor maximizar el beneficio conjunto (Teoría de Juegos)

3. Otros tipos de juegos

- Ejemplo: El dilema del prisionero

  La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarles, y tras haberles separado, les visita a cada uno y les ofrece el mismo trato:

	Tú niegas	Tú confiesas
Él niega	Ambos condena a 5 meses	Él 10 años, tú libre
Él confiesa	Él libre, tú 10 años	Ambos condena a 6 años

- Estrategia dominante: confesión
  - Independientemente de lo que decida el otro, puedes reducir tu condena confesando
  - Sin embargo, el resultado no es óptimo
- *Estrategia óptima: colaboración* (equilibrio de Nash)
- La decisión en realidad depende de la matriz de costes

1. Búsqueda con adversarios
2. Búsqueda Minimax
    1. Minimax con decisiones imperfectas
    2. Poda Alfa-Beta
3. Otros tipos de juegos
4. Estado del arte en problemas de juegos

4. Estado del Arte en problemas de juegos

- Ajedrez:
  - Software para enseñar ajedrez o máquinas dedicadas
  - Distribución de tiempos (en % de tiempo dedicado)
    - Generación de movimientos (ordenados por *h*) y poda $\alpha - \beta$: 50%
    - Evaluación estática: 40%
    - Búsqueda ¡10%!
  - Aperturas y finales almacenados en bases de datos
  - DEEP BLUE (1997)
    - Derrota a Kasparov 3,5 a 2,5
    - Explora hasta 30 capas por delante
  - Mejoras algorítmicas: bajan ramificación efectiva a 3 (de 35)
  - AlphaZero (Diciembre, 2018)

- Otello
  - Los ordenadores superiores al campeón del mundo
  - LOGISTELLO (1997)

4. Estado del Arte en problemas de juegos

- Damas (oficialmente resuelto desde 2007),
  - Chinook (1994)
    - Poda alfa-beta con 444 posiciones de finales
    - BD de finales de partida: juego perfecto para configuraciones de tablero con 8 o menos piezas
    - Campeón del mundo en 1994
- Backgammon
  - Complicación: aleatoriedad (tiradas de dados)
  - Los programas de búsqueda simple son malos
  - TD-GAMMON (1995) entre los 3 mejores del mundo
    - Sistema de aprendizaje por refuerzo (mucha búsqueda y uso de los resultados para construir una muy buena función heurística)
- Bridge
  - Información oculta (las cartas de los otros jugadores)
  - Comunicación con el compañero mediante un lenguaje restringido
  - Nivel bueno

4. Estado del Arte en problemas de juegos

- Go
  - Como el ajedrez: información perfecta, no aleatoriedad ni comunicación
  - Problema: el enorme factor de ramificación (b ~361)
    - Los métodos de búsqueda que funcionan bien en ajedrez no son susceptibles de aplicarse en el go
    - Los jugadores humanos parecen basarse en algo mucho más complejo: comprensión de patrones espaciales
    - Planteamiento de métodos basados en mejores heurísticas y menos en búsqueda por fuerza bruta, con bases de conocimiento de patrones
  - Nivel de campeón (AlphaGo de Google, Marzo, 2016, Seul)
  - AlphaGo Zero (Octubre, 2017) autoentrenado