Búsqueda con adversarios

Inteligencia Artificial e Ingeniería del Conocimiento

Constantino Antonio García Martínez

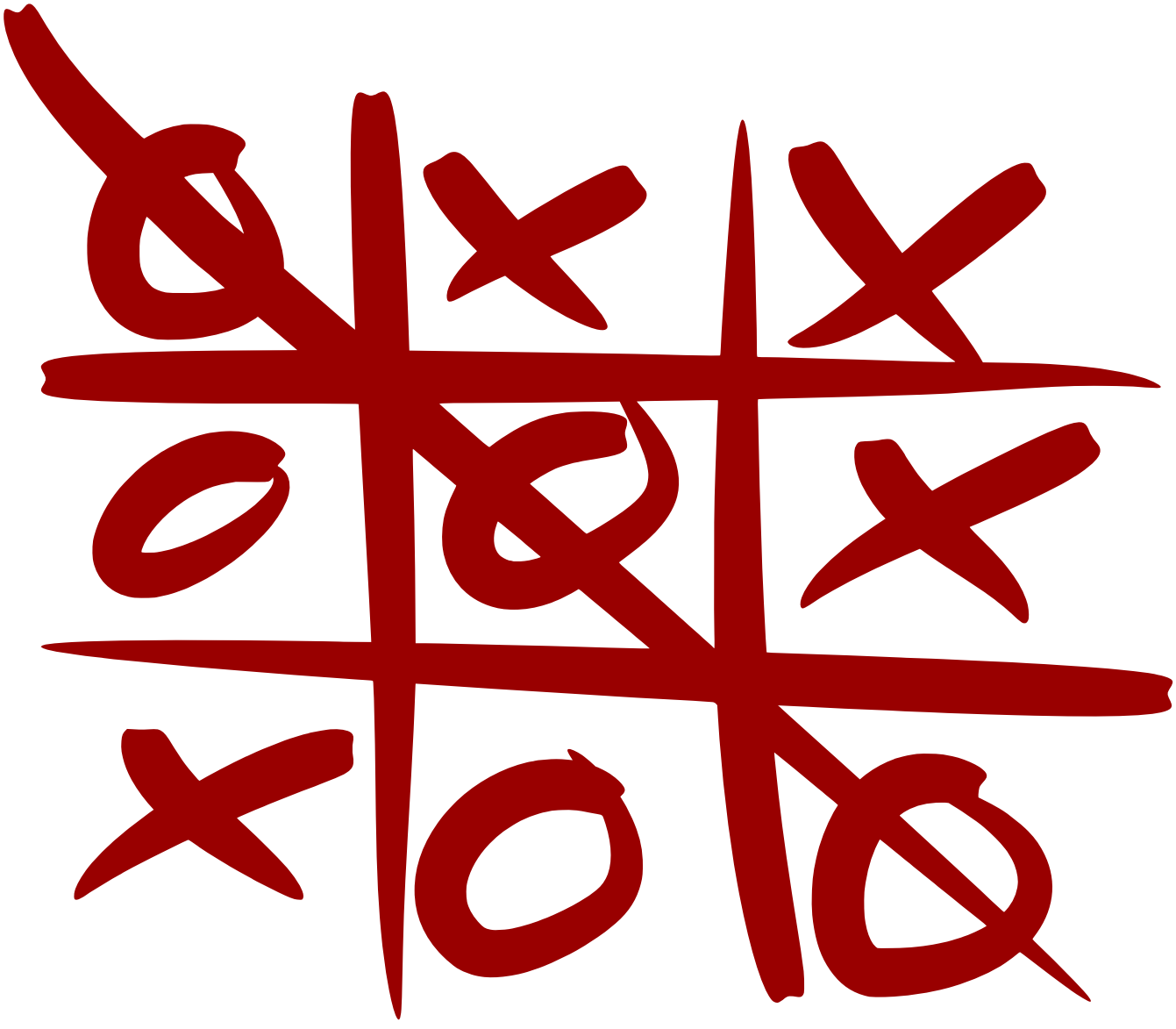
Universidad San Pablo Ceu

Bibliografía

- Russell, Stuart J., and Peter Norvig. Artificial intelligence: a modern approach. Pearson, 2016.

Motivación

Aplicaciones Reales



Aplicaciones Reales

La búsqueda con adversarios se aplica a muchos dominios:

- **Ciberseguridad**: Estrategias atacante vs. defensor
- **Finanzas**: Competiciones de trading algorítmico
- **Subastas**: Estrategias de puja contra competidores
- **Planificación militar**: Toma de decisiones estratégicas
- **Robótica**: Coordinación multi-robot con objetivos en conflicto
- **Juegos**: Damas, Go, Ajedrez, etc.

Fundamentos de Teoría de Juegos

Juegos de Dos Jugadores con Información Perfecta y Suma Cero

Nos centramos en **Juegos de Dos Jugadores con Información Perfecta y Suma Cero** (Two-Player Perfect Information Zero-Sum Games):

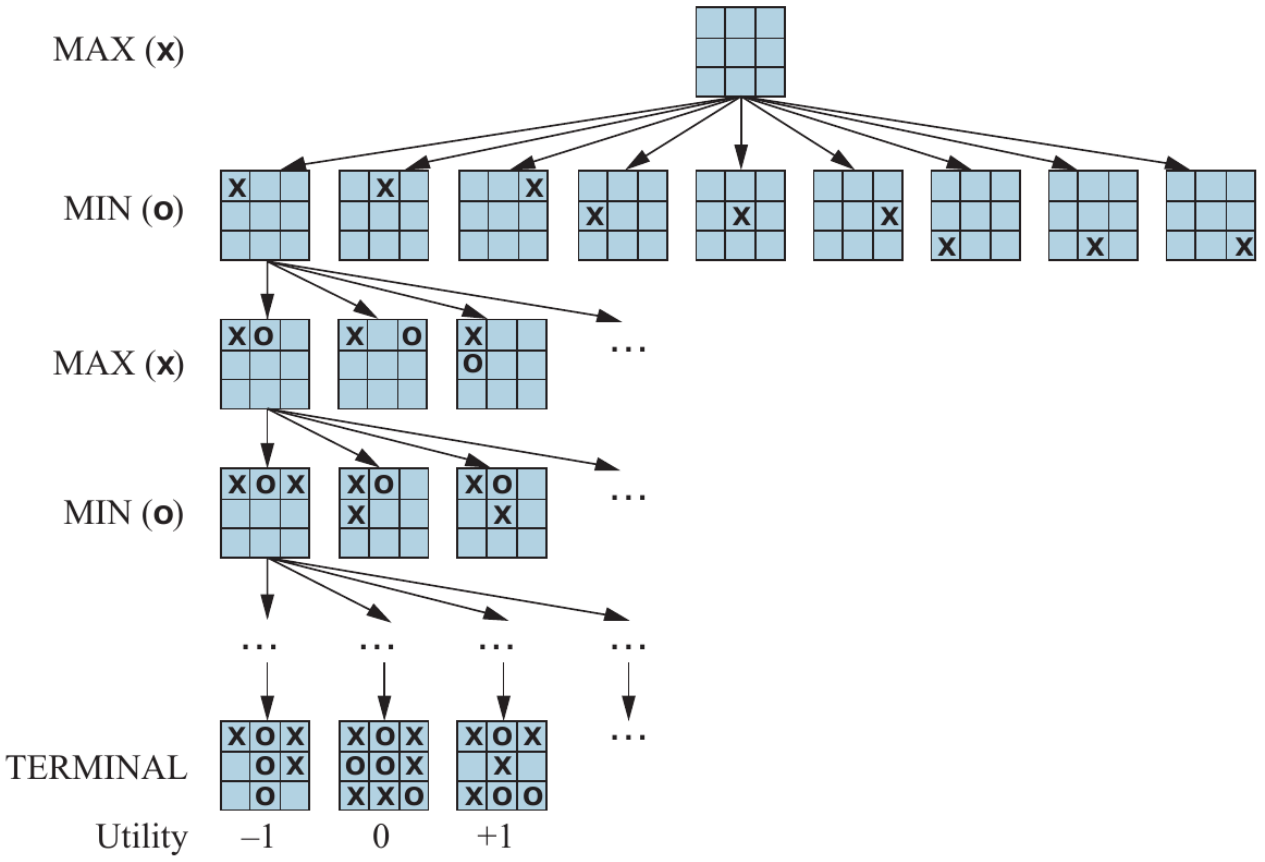
- **Información perfecta**: Todo el estado del juego visible para ambos jugadores
- **Suma cero**: La ganancia de un jugador = pérdida del otro
- **Terminología adicional**:
  - Posición = Estado
  - Semimovimiento (ply) = Un movimiento de un jugador
- Jugadores: MAX y MIN

Modelado del Juego

Un juego consta de:

- **Estado inicial**: Posición de partida
- **TO-MOVE(s)**: Qué jugador mueve en el estado s
- **ACTIONS(s)**: Movimientos legales en el estado s
- **RESULT(s,a)**: Modelo de transición
- **IS-TERMINAL(s)**: Test de terminación
- **UTILITY(s,p)**: Función de utilidad para el jugador p

Árbol de Juego



[1]

[1] Imagen tomada de AIMA

Algoritmos Exactos

Algoritmos Exactos

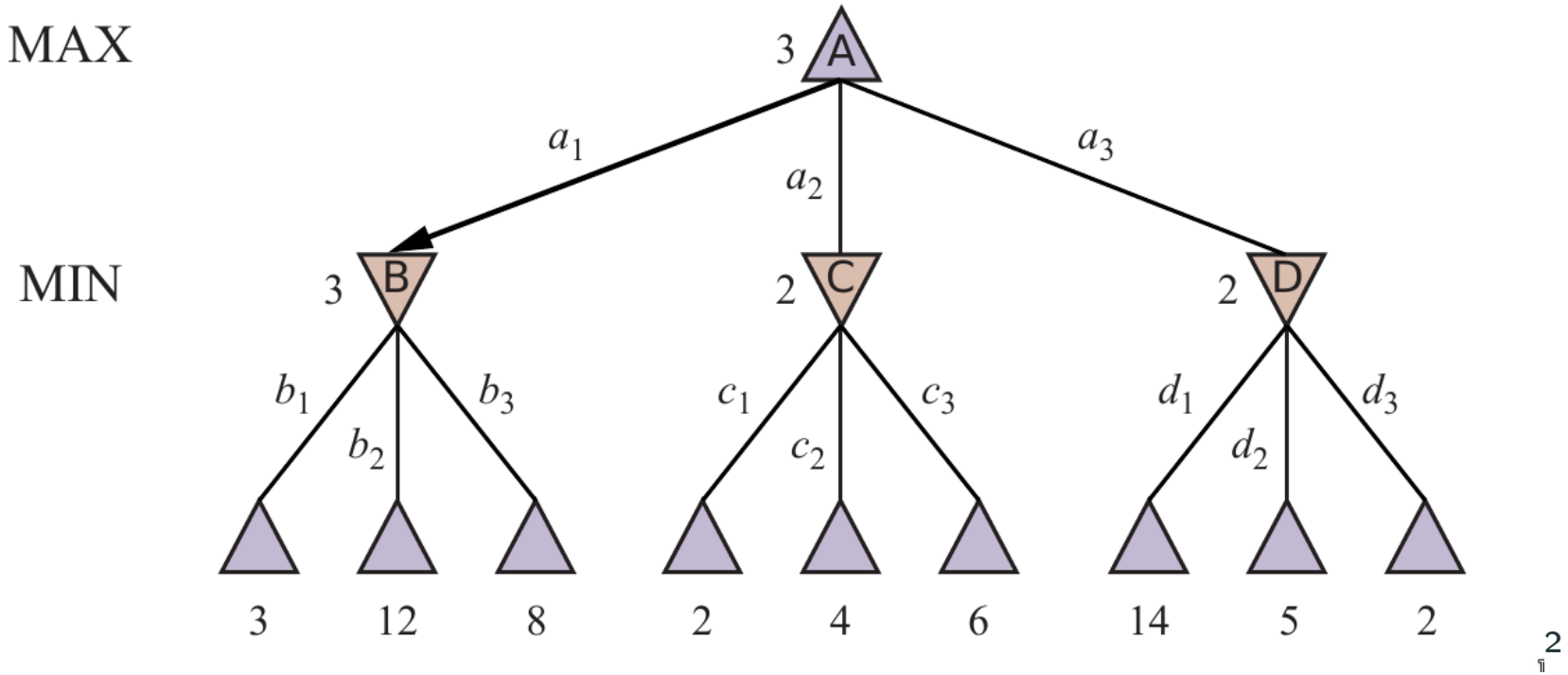
Algoritmo Minimax

Algoritmo Minimax

Idea clave del Minimax

- MAX quiere maximizar la utilidad
- MIN quiere minimizar la utilidad de MAX
- MAX necesita una **estrategia contingente** - plan condicional respondiendo a los movimientos óptimos de MIN

Árbol de juego Minimax



[2]Imagen tomada de AIMA

Valor Minimax

El valor minimax MINIMAX(s) es la utilidad para MAX asumiendo que ambos jugadores juegan óptimamente:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{si IS-TERMINAL}(s) \\ \max_a \text{MINIMAX}(\text{RESULT}(s, a)) & \text{si TO-MOVE}(s) = \text{MAX} \\ \min_a \text{MINIMAX}(\text{RESULT}(s, a)) & \text{si TO-MOVE}(s) = \text{MIN} \end{cases}$$

Algoritmo Minimax

Code Example: Algoritmo Minimax

Propiedades del Algoritmo Minimax

- **Completo**: Sí (árbol finito)
- **Óptimo**: Sí (contra oponente óptimo)
- **Complejidad temporal**: $O(b^m)$
- **Complejidad espacial**: $O(bm)$ (primero en profundidad)

Donde:

- b = factor de ramificación
- m = profundidad máxima

Algoritmos Exactos

Poda Alfa-Beta

Poda Alfa-Beta (Alpha-Beta Pruning)

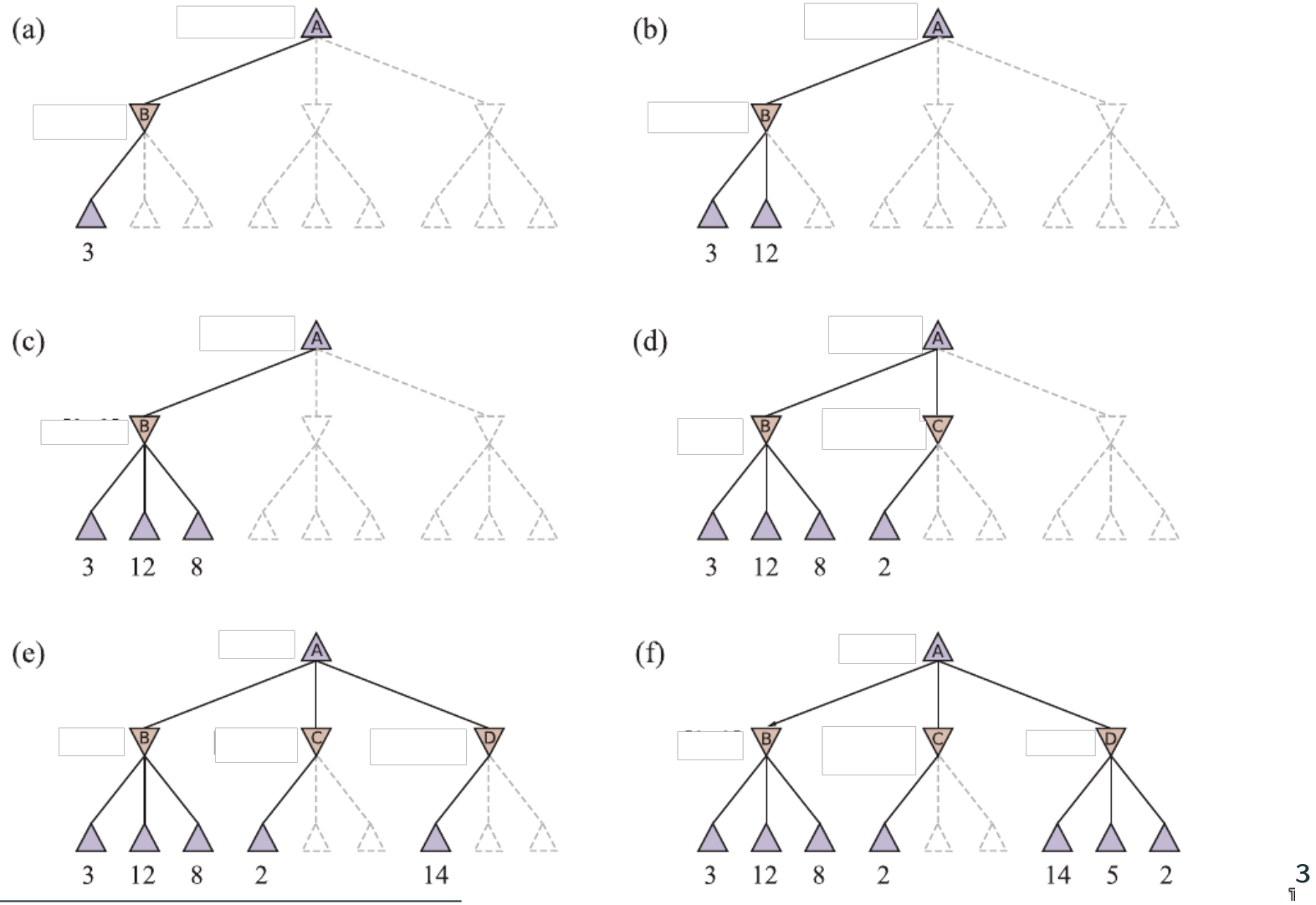
Example: Intuición Alfa-Beta

- **Vendedor** (MAX): No aceptará menos de 10€
- **Comprador** (MIN): No pagará más de 5€
- **Resultado**: ¡La negociación no tiene sentido!
- **Poda**: No necesitamos explorar las ofertas de este comprador

Algoritmo Alfa-Beta

- **Idea clave**: Eliminar ramas que no pueden afectar la decisión final
- **Alfa (α)**: Mejor valor para MAX encontrado hasta ahora ("al menos")
- **Beta (β)**: Mejor valor para MIN encontrado hasta ahora ("como máximo")
- **Condición de poda**: Si $\alpha \geq \beta$, podar ramas restantes

Algoritmo Alfa-Beta



[3]Imagen tomada de AIMA

Algoritmo Alfa-Beta

Code Example: Algoritmo Alpha-Beta

Optimizaciones e Implementaciones

Efectividad de Alfa-Beta

La ordenación de movimientos es fundamental para la efectividad. Si la ordenación pudiera hacerse perfectamente...

- **Mejor caso**: $O(b^{m/2})$ en lugar de $O(b^m)$
- Factor de ramificación efectivo: $\sqrt{b}$ en lugar de b
- Para ajedrez: $\sim$6 en lugar de $\sim$35

Heurísticas de Ordenación de Movimientos:

- **Movimientos asesinos (Killer moves)**: Recordar uno o dos movimientos previos que hayan causado una poda-β a la misma profundidad
- **Heurística de historia**: Generalización. Tabla de movimientos con podas-β en cualquier momento anterior.
- **Específico de ajedrez**: Capturas $>$ jaques $>$ amenazas $>$ movimientos tranquilos

Tablas de Transposición

- **Problema**: Mismas posiciones alcanzadas por diferentes secuencias de movimientos (transposiciones)
- **Solución**: Cachear valores heurísticos de posiciones en tabla hash
- **Beneficio**: Puede duplicar la profundidad de búsqueda alcanzable
- Especialmente efectivo en ajedrez

Ejemplo de Transposición

1. e4 e5, 2. Kf3 Kc6

vs.

1. Kf3 Kc6, 2. e4 e5

¡Misma posición, diferente orden de movimientos!

Otras Optimizaciones Prácticas

Reducción por Simetría:

- Muchas posiciones equivalentes bajo rotaciones/reflexiones
- Tres en raya: 8 versiones simétricas de cada posición

Bases de Datos de Aperturas y Finales:

- **Libros de aperturas**: Mejores movimientos pre-calculados para aperturas comunes
- **Tablas de finales**: Juego perfecto con $\leq$7 piezas
- ¿Por qué buscar miles de millones de nodos para jugar 1.e4 o 1.d4?

Juegos Complejos

Estrategias de Shannon

Incluso con poda alfa-beta y ordenación inteligente de movimientos, minimax no funciona para juegos como el ajedrez y Go...

Ejemplo de complejidad del ajedrez

- Factor de ramificación medio: $b \approx 35$
- Longitud media del juego: $\sim$40 movimientos por jugador ($m = 80$)
- Posiciones totales: $35^{80} \approx 10^{123}$
- A mil millones de posiciones/seg: 10^{114} segundos $\approx 10^{106}$ años!

En 1950, Claude Shannon propuso dos estrategias:

Estrategia Tipo A: Considerar todos los movimientos hasta profundidad fija, usar función de evaluación
Estrategia Tipo B: Ignorar movimientos malos, seguir líneas prometedoras profundamente

Históricamente: Programas de ajedrez $\rightarrow$ Tipo A, programas de Go $\rightarrow$ Tipo B
Actualmente: ¡Ambas se usan simultáneamente!

Juegos Complejos

Estrategia Tipo A: Funciones de Evaluación

Búsqueda Heurística en Árbol Alfa-Beta

Minimax modificado con corte:

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s,\text{MAX}) & \text{si IS-CUTOFF}(s,d) \\ \max_a \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{si TO-MOVE}(s) = \text{MAX} \\ \min_a \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{si TO-MOVE}(s) = \text{MIN} \end{cases}$$

Requisitos de la Función de Evaluación:

- **EVAL(s,p)**: Estimación heurística de utilidad
- Debe coincidir con UTILITY en estados terminales
- Debería estar entre valores de pérdida y victoria
- Cálculo rápido, estimación precisa, correlación con probabilidades de ganar

Cortar la Búsqueda es Complejo (I)

Problema: Profundidad fija $\Rightarrow$ ineficiencia.

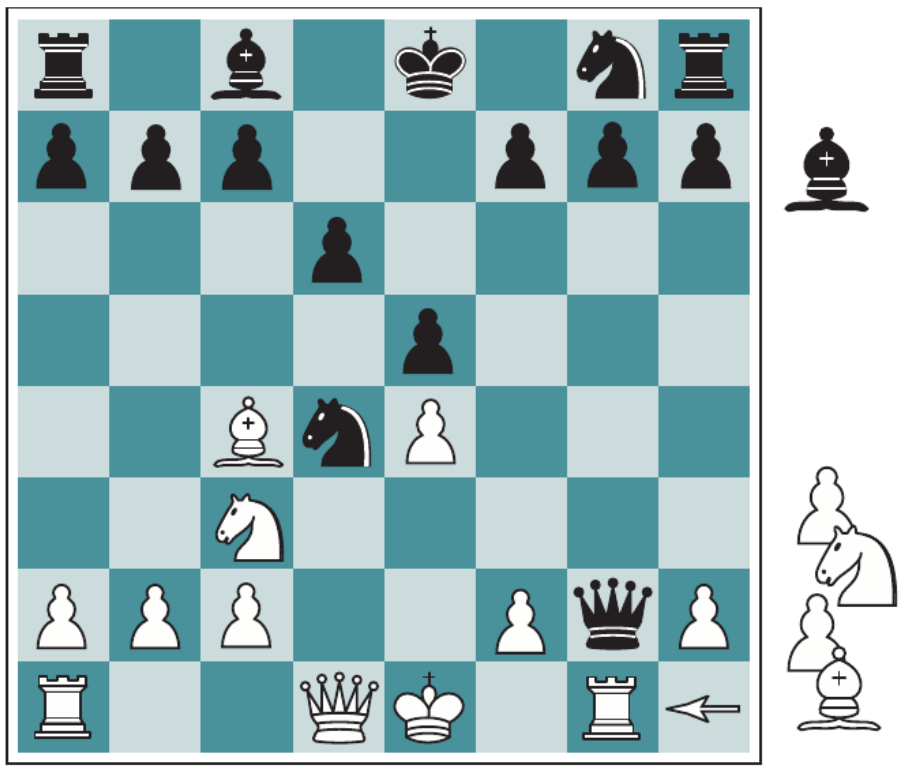
Solución:

- **Iterative Deepening:**
  - Buscar incrementando profundidad en cada iteración.
  - Si se acaba el tiempo, devolver el movimiento de la iteración más profunda completada.

Cortar la Búsqueda es Complejo (II)

Búsqueda de Quiescencia (Quiescence Search):

- Solo evaluar posiciones **quiescentes** (estables)
- Evitar posiciones con capturas/amenazas pendientes
- Continuar búsqueda hasta que la posición se estabilice



(b) White to move

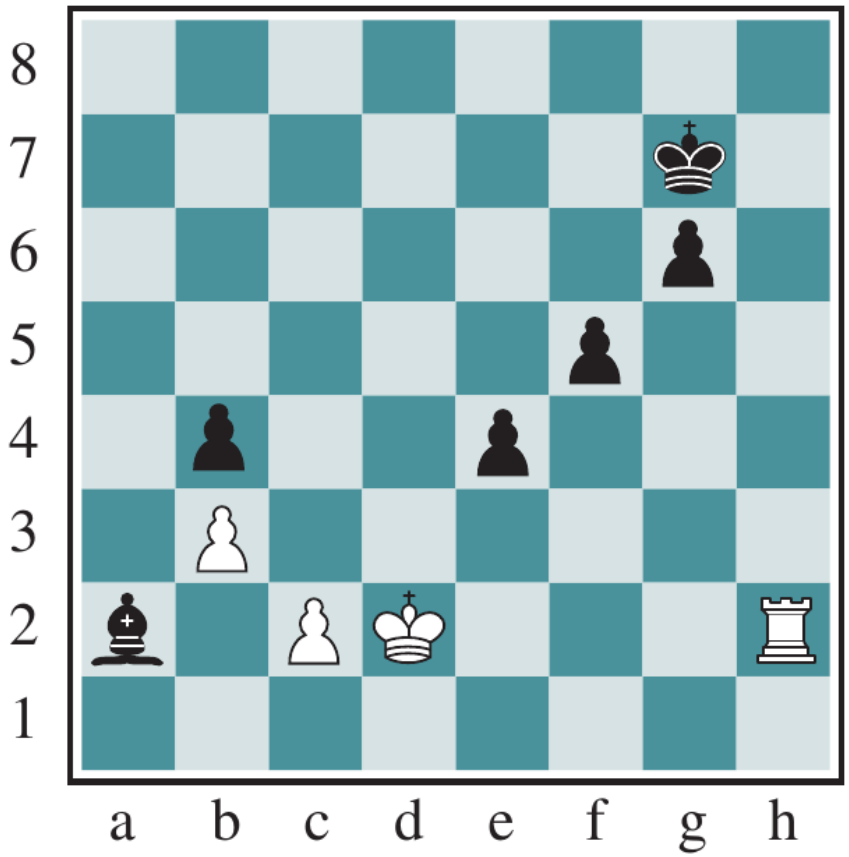
[4]

[4] Imagen tomada de AIMA

Cortar la Búsqueda es Complejo (III)

Efecto Horizonte (Horizon Effect):

- Eventos malos empujados más allá del horizonte de búsqueda
- Mitigación: **Extensiones singulares** para movimientos claramente mejores



Mueven las negras[5]

[5] Imagen tomada de AIMA

Juegos Complejos

Estrategias Tipo B

Poda hacia Adelante (Forward Pruning)

Objetivo: Podar movimientos que parecen pobres (estrategia Tipo B)

- **Búsqueda en haz (Beam search)**: Mantener solo los mejores k movimientos
- **ProbCut**: Usar estadísticas para podar con seguridad
- **Reducción de movimientos tardíos**: Profundidad reducida para movimientos posteriores
- **Riesgo**: Podría podar el mejor movimiento

Monte Carlo Tree Search (MCTS): Visión General

Innovación Clave: Usar simulaciones aleatorias en lugar de funciones de evaluación

Idea Básica:

- Estimar valor del estado mediante **simulación** (partidas hasta el final)
- Promediar utilidad sobre muchos juegos aleatorios
- Enfocar recursos computacionales en partes prometedoras del árbol

¿Por qué MCTS?

- Maneja factores de ramificación altos (Go: 250 movimientos posibles)

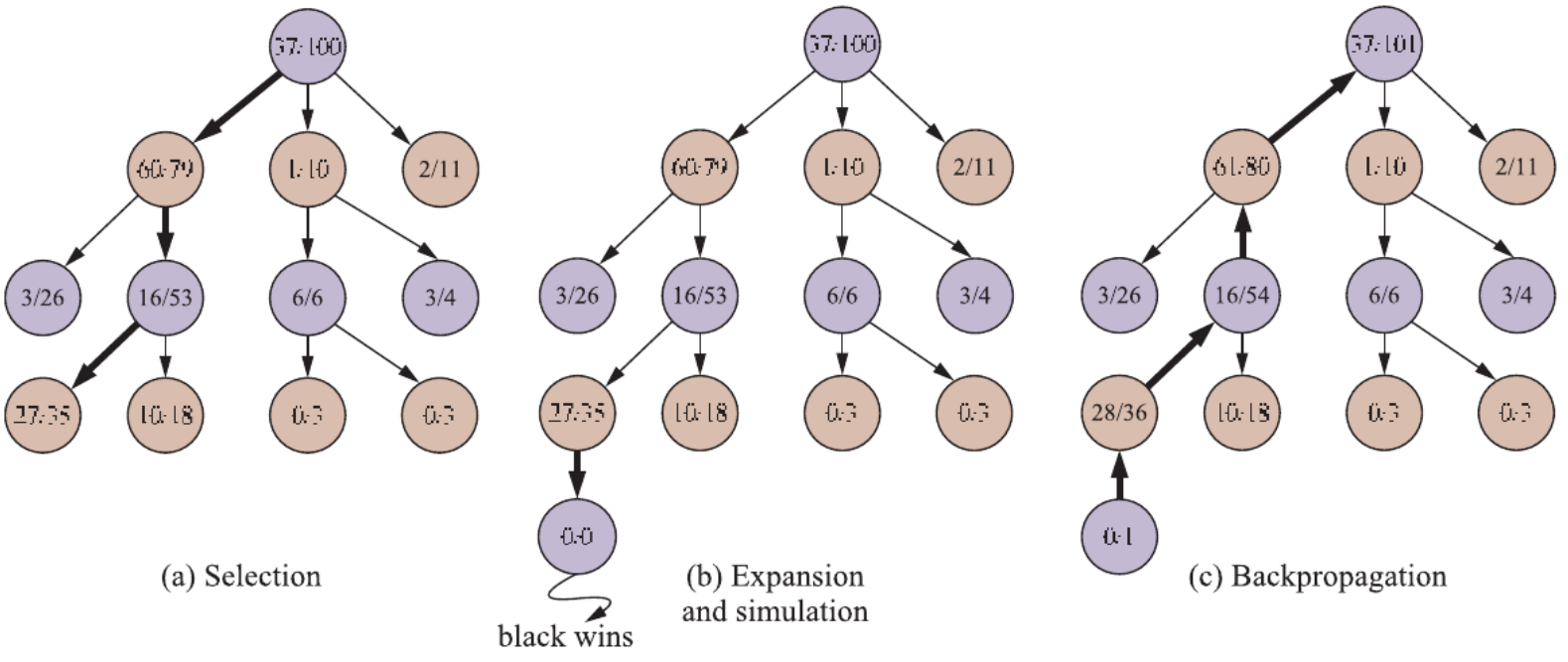
Enfoques Modernos

MCTS: Decisiones Clave de Diseño

Dos decisiones fundamentales:

1. **Política de Simulación (Playout Policy)**:
   - ¿Cómo elegimos movimientos durante las simulaciones?
   - Simulación aleatoria pura $\rightarrow$ "¿mejor movimiento si ambos juegan al azar?"
   - Necesitamos sesgar hacia movimientos buenos para obtener información útil
   - Opciones: heurísticas del dominio, redes neuronales entrenadas
2. **Política de Selección (Selection Policy)**:
   - ¿Desde qué posiciones iniciamos las simulaciones?
   - ¿Cuántas simulaciones asignamos a cada posición?
   - Debe equilibrar:
     - **Exploración**: estados con pocas simulaciones
     - **Explotación**: estados con buenos resultados previos

Monte Carlo Tree Search (MCTS)



[6]

[6] Imagen tomada de AIMA

MCTS: Las Cuatro Fases

1. **Selección**: Navegar el árbol usando política de selección para equilibrar exploración/explotación
2. **Expansión**: Añadir un nuevo nodo hijo al árbol
3. **Simulación**: Ejecutar simulación aleatoria desde el nuevo nodo hasta estado terminal
4. **Retropropagación**: Actualizar estadísticas de todos los nodos en el camino

MCTS: Políticas y Aprendizaje Automático

Integración Moderna con ML (AlphaGo/AlphaZero):

- **Redes neuronales para aprender las políticas (Policy network)**
- **Entrenamiento por refuerzo (Reinforcement Learning)**: El sistema mejora jugando contra sí mismo

Terminación Temprana:

- Detener simulaciones que llevan demasiados movimientos

MCTS: Ventajas y Limitaciones

Ventajas:

- Maneja bien factores de ramificación altos
- Algoritmo anytime (mejora con más tiempo)
- Expansión asimétrica del árbol (se enfoca en variaciones importantes)

Limitaciones:

- Dificultades con estados "trampa"(un movimiento cambia el resultado)
- Puede ser lento reconociendo secuencias forzadas
- Simulaciones aleatorias pueden no reflejar juego realista

Limitaciones y Extensiones

Lo Que No Hemos Cubierto (I)

Nuestro enfoque ha sido en **Juegos de Dos Jugadores con Información Perfecta y Suma Cero**, pero existen muchas extensiones:

Juegos Estocásticos:

- Incluyen nodos de azar (dados, cartas)
- Algoritmo Expectiminimax
- Ejemplos: Backgammon, Póker

Juegos Multi-Jugador:

- Más de 2 jugadores con objetivos potencialmente diferentes
- Formación de coaliciones y equilibrios de Nash
- Ejemplos: Risk, Diplomacy

Lo Que No Hemos Cubierto (II)

Juegos de Suma No Cero:

- Las utilidades de los jugadores no suman cero
- Pueden surgir estrategias cooperativas
- Teoría de juegos y equilibrios de Nash cruciales

Juegos de Información Parcial:

- Los jugadores no pueden ver todo el estado del juego
- Estados de creencia y conjuntos de información
- Ejemplos: Póker, Batalla Naval, ajedrez Kriegspiel

Limitaciones de la Búsqueda (Pura) en Juegos

- **Vulnerabilidad a errores de evaluación**: Pequeños errores pueden propagarse
- **Computación fija**: No se adapta a la complejidad de la posición -> utilidad de una expansión de nodo
- **Sin razonamiento estratégico**: Puramente táctico, movimiento a movimiento
- **Sin aprendizaje**: No mejora con la experiencia

Direcciones Futuras:

- Metarazonamiento sobre la computación
- Planificación estratégica de alto nivel
- Integración de aprendizaje automático (AlphaGo, AlphaZero)