Capítulo — Búsqueda en Árbol de Monte Carlo

Este capítulo extiende el marco abstracto de árbol de juego desarrollado previamente. Habiendo definido **qué** es un árbol de juego, ahora investigamos **cómo** buscar en dicho árbol cuando la enumeración exhaustiva es imposible. Nuestra herramienta es la **Búsqueda en Árbol de Monte Carlo (MCTS)**—un método estadísticamente fundamentado que incrementalmente construye un árbol *parcial* mediante el muestreo de jugadas simuladas.

---

1 Descripción Conceptual

Recordemos la tupla de juego
$\mathcal G = (S, s_0, A, T, P, Z, u)$
del capítulo anterior. Cada *jugada* es una trayectoria
$(s_0,a_0,s_1,a_1,\dots,s_H)$ que termina en una hoja $s_H\in Z$ con vector de utilidad $u(s_H)$.

Idea central. En lugar de evaluar *cada* trayectoria, MCTS *muestrea* muchas jugadas aleatorias (*rollouts*) desde nodos seleccionados, promedia sus utilidades, y usa esas medias empíricas para guiar muestreos posteriores. Con el tiempo, el procedimiento concentra el esfuerzo en ramas prometedoras mientras continúa explorando alternativas.

Visualmente, el algoritmo desarrolla un árbol *poco profundo y frondoso* cerca de la raíz y un subárbol *profundo y estrecho* a lo largo de líneas que parecen fuertes—reflejando el enfoque natural del razonamiento estratégico.

---

2 Fundamentos Estadísticos

2.1 Estimador de Rollout

Fijamos un nodo v (estado s) y una *política predeterminada* estocástica ρ. Un solo rollout produce
$X = u(\text{leaf}(\text{rollout}(s,\rho))).$
Dado que el juego es determinista una vez que se eligen las acciones, la aleatoriedad entra solo a través de ρ (y movimientos de azar, si los hay). Repitiendo el experimento se obtienen muestras i.i.d.
$X_1,\dots,X_n$ con media
$\bar X_n = \frac1n\sum_{i=1}^n X_i.$
Por la **Ley Fuerte de los Grandes Números**
☑ $\quad (n\to\infty).$
Así, los promedios empíricos convergen a la *utilidad esperada* en s.

2.2 Exploración–Explotación vía Límites de Confianza Superiores

Seleccionar qué hijo muestrear es un problema de **bandido multibrazo**. Para un nodo padre con recuento de visitas N y un hijo con visitas n_i y utilidad media actual Q_i, la **puntuación UCT** es
$Q_i + c,\sqrt{\frac{\ln N}{n_i}},$

PROF

donde $c>0$ controla la exploración. La Ecuación (1) se deriva del límite de Hoeffding y produce un arrepentimiento logarítmico.

2.2.1 Formulación Formal de UCT (Upper Confidence Bounds for Trees)

UCT es una adaptación del algoritmo UCB1 (Upper Confidence Bounds 1) al contexto de árboles de búsqueda. Formalmente, UCT se fundamenta en la teoría de bandidos multi-brazo para optimizar la exploración-explotación.

Definición formal: Sea s un nodo en el árbol de juego con hijos ${s_1, s_2, ..., s_k}$. Definimos:

- $Q(s_i)$: El valor estimado del nodo hijo s_i (promedio de recompensas)
- $N(s)$: Número total de visitas al nodo padre s, es decir cuantas veces hemos explorado ese nodo o hemos ejecutado el algoritmo en ese nodo.
- $n(s_i)$: Número de visitas al hijo s_i, cuantas veces hemos visitado ese nodo hije.
- c: Constante de exploración (típicamente $c = \sqrt{2}$ derivado de la teoría UCB)

Para cada iteración, UCT selecciona el hijo s_i que maximiza:

$UCT(s_i) = Q(s_i) + c \sqrt{\frac{\ln N(s)}{n(s_i)}}$

Fundamento matemático: Esta fórmula proviene del límite de confianza superior de Hoeffding para distribuciones acotadas en $[0,1]$. Para cualquier variable aleatoria X con media μ en $[0,1]$, después de n observaciones independientes con media empírica $\bar{X}_n$, tenemos que:

$P(\mu > \bar{X}_n + \sqrt{\frac{2\ln(1/\delta)}{n}}) \leq \delta$

Donde δ es la probabilidad de error. Estableciendo $\delta = N^{-4}$ y resolviendo para el límite, obtenemos el término de exploración $\sqrt{\frac{\ln N}{n}}$.

Elección de $\delta = N^{-4}$: Esta selección específica no es arbitraria, sino que deriva de la teoría de procesos estocásticos y el análisis de confiabilidad. Kocsis & Szepesvári, en su formulación original de UCT (2006), eligieron esta tasa de decaimiento por tres razones fundamentales:

PROF

1. **Convergencia garantizada**: Al establecer $\delta = N^{-4}$, la suma $\sum_{N=1}^{\infty} \delta = \sum_{N=1}^{\infty} N^{-4}$ converge (es la serie zeta de Riemann $\zeta(4) = \pi^4/90 \approx 1.08$). Esto garantiza que, por el lema de Borel-Cantelli, la probabilidad de que ocurran infinitas desviaciones significativas es cero, asegurando la convergencia eventual al valor óptimo.

2. **Balance entre confiabilidad y exploración**: Un exponente mayor (por ejemplo, N^{-5}) generaría un término de exploración más pequeño, reduciendo la exploración. Un exponente menor (por ejemplo, N^{-3}) incrementaría demasiado la exploración a expensas de la explotación. El valor N^{-4} proporciona un equilibrio teóricamente óptimo.

3. **Complejidad computacional**: Esta elección conduce a una tasa de convergencia que optimiza el compromiso entre velocidad de aprendizaje y costo computacional total.

Asimetría del límite: La desigualdad de Hoeffding es intrínsecamente asimétrica y UCT utiliza deliberadamente solo el límite superior (y no el inferior) por varias razones críticas:

1. **Principio de optimismo frente a la incertidumbre**: En el contexto de exploración-explotación, es preferible sobreestimar el potencial de opciones poco exploradas que subestimarlas. Este "optimismo frente a la incertidumbre" es un principio fundamental en teoría de aprendizaje por refuerzo.

2. **Propiedades matemáticas de búsqueda en árbol**: En un árbol de juego minimax, el valor real de un nodo está acotado superiormente por el "mejor caso posible" de sus descendientes. El límite superior captura esta relación natural mejor que un límite simétrico o inferior.

3. **Eficiencia en la poda implícita**: Al considerar solo desviaciones hacia arriba, UCT puede descartar implícitamente ramas del árbol que tienen muy baja probabilidad de superar el valor actual óptimo, sin necesidad de explorarlas exhaustivamente.

4. **Resultado teórico**: Se puede demostrar matemáticamente que esta asimetría lleva a una tasa óptima de arrepentimiento de $O(\sqrt{n \ln n})$, mientras que un enfoque simétrico resultaría en tasas de convergencia significativamente inferiores en el contexto de decisiones secuenciales.

Propiedades teóricas clave:

1. **Balance óptimo**: El primer término (Q_i) favorece la explotación de nodos con alto valor esperado, mientras el segundo término favorece la exploración de nodos poco visitados.

2. **Convergencia asintótica**: A medida que $n_i \to \infty$, el término de exploración tiende a cero, priorizando eventualmente los nodos de mayor valor.

3. **Garantía de exploración**: Dado que el término de exploración es infinito cuando $n_i = 0$, UCT garantiza que todos los nodos serán visitados al menos una vez antes de revisitar cualquier nodo.

4. **Arrepentimiento sublineal**: UCT garantiza un arrepentimiento (diferencia entre la recompensa acumulada óptima y la obtenida) acotado por $O(\sqrt{n \ln n})$.

5. **Invarianza de escala adaptativa**: A diferencia de otros algoritmos, UCT ajusta automáticamente la exploración según la profundidad del árbol y la magnitud de las recompensas, sin necesidad de recalibración manual en diferentes dominios.

PROF

6. **Consistencia teórica**: Se puede demostrar que UCT converge al árbol de decisión óptimo a medida que el número de simulaciones tiende a infinito, convirtiéndolo en un algoritmo asintóticamente óptimo.

7. **Comportamiento no paramétrico**: UCT no hace suposiciones sobre la estructura de recompensas subyacente más allá de estar acotada, haciéndolo robusto frente a distribuciones complejas o multimodales.

Consideraciones prácticas de implementación:

Aunque UCT es matemáticamente elegante, su implementación práctica requiere ajustes adicionales para optimizar su rendimiento:

1. **Selección del parámetro c**: Teóricamente $c = \sqrt{2}$ es óptimo para recompensas en $[0,1]$, pero en la práctica, este valor debe ajustarse según:

    - La escala de las recompensas (valores no normalizados)
    - La profundidad esperada del árbol
    - El factor de ramificación promedio
    - El tiempo computacional disponible

2. **Inicialización de nodos inexplorados**: Para evitar la divisón por cero cuando $n_i = 0$, existen diversas estrategias:

    - Asignar un valor inicial optimista
    - Utilizar una prioridad infinita para nodos no visitados
    - Emplear un pequeño valor ϵ como denominador: $\sqrt{\frac{\ln N}{n_i + \epsilon}}$

3. **Modificaciones para adversarios**: En juegos de suma cero, se aplica una transformación negamax a las recompensas propagadas, invirtiendo valores entre niveles alternos del árbol.

Esta formalización hace de UCT un algoritmo matemáticamente sólido y aplicable a una amplia gama de problemas de decisión secuencial bajo incertidumbre, desde juegos clásicos como tic-tac-toe hasta planificación robótica y simulaciones financieras complejas.

---

3 Algoritmo Canónico de MCTS

3.1 Visión General del Algoritmo

MCTS opera iterativamente sobre un *árbol parcial* $\mathcal T^* \subset \mathcal T$, construyendo y evaluando incrementalmente diferentes ramas del árbol de juego completo. El algoritmo canónico consta de cuatro fases fundamentales que se repiten hasta agotar el presupuesto computacional disponible:

1. **Selección** — Navegación descendente desde la raíz
2. **Expansión** — Añadir nuevos nodos al árbol
3. **Simulación** — Estimar el valor de nuevos nodos
4. **Retropropagación** — Actualizar información en el camino de regreso a la raíz

PROF

Cada una de estas fases cumple un propósito específico en el balance entre exploración y explotación que caracteriza a MCTS.

3.2 Las Cuatro Fases en Detalle

3.2.1 Fase de Selección

Definición formal: Partiendo del nodo raíz s_0, el algoritmo desciende por el árbol seleccionando en cada nivel el nodo hijo s_i que maximiza la fórmula UCT:

$UCT(s_i) = Q(s_i) + c\sqrt{\frac{\ln N(s)}{N(s_i)}}$

donde:

- $Q(s_i)$ es el valor empírico promedio del nodo hijo

- $N(s)$ es el número de visitas al nodo padre
- $N(s_i)$ es el número de visitas al nodo hijo
- c es la constante de exploración (típicamente $\sqrt{2}$ para recompensas en $[0,1]$)

Conceptos clave:

- **Acción expandida:** Una acción $a \in A(s)$ se considera "expandida" si ya se ha creado el nodo hijo correspondiente al estado resultante $s' = T(s, a)$ en el árbol parcial $\mathcal T^*$.
- **Nodo completamente expandido:** Un nodo s está "completamente expandido" cuando todas las acciones legales $A(s)$ han sido expandidas, es decir, cuando todos los posibles estados sucesores inmediatos están representados como nodos hijos en el árbol.
- **Nodo hoja en el árbol parcial:** Un nodo que no tiene hijos en el árbol parcial $\mathcal T^*$, aunque podría tener acciones legales en el juego real.

Proceso detallado:

1. Iniciar en el nodo raíz s_0.
2. Mientras el nodo actual s no sea terminal y tenga todos sus hijos ya expandidos:
    - Para cada hijo s_i de s, calcular su puntuación UCT.
    - Seleccionar el hijo $s_{next} = \arg\max_{s_i} UCT(s_i)$.
    - Actualizar el nodo actual: $s \leftarrow s_{next}$.
3. El proceso continúa hasta encontrar:
    - Un nodo terminal (estado final del juego), o
    - Un nodo con acciones legales no expandidas todavía.
4. Si el nodo resultante s no es terminal y tiene acciones legales no expandidas, el algoritmo pasa a la fase de Expansión con este nodo.
5. Si el nodo resultante s es terminal (estado final del juego), el algoritmo salta la fase de Expansión y Simulación, y procede directamente a la fase de Retropropagación.

Consideraciones técnicas:

- Para nodos nunca visitados ($N(s_i) = 0$), la puntuación UCT se considera infinita, garantizando que serán seleccionados al menos una vez.
- En implementaciones prácticas, los empates en puntuación UCT se resuelven mediante criterios secundarios como heurísticas específicas del dominio o selección aleatoria uniforme.
- El valor de c controla directamente el balance entre exploración y explotación: valores más altos favorecen la exploración de ramas menos visitadas, mientras que valores más bajos priorizan ramas con altos valores empíricos.

PROF

3.2.2 Fase de Expansión

Definición formal: Dado un nodo seleccionado s que no es terminal y no está completamente expandido, se selecciona una acción no explorada $a \in A_{unexpanded}(s)$ y se crea un nuevo nodo hijo $s' = T(s,a)$ que se añade al árbol parcial $\mathcal T^*$.

Proceso detallado:

1. Identificar el conjunto $A_{unexpanded}(s) \subset A(s)$ de acciones legales aún no representadas en el árbol.

- $A(s)$ es el conjunto completo de acciones legales desde el estado s
- $A_{expanded}(s)$ es el subconjunto de acciones que ya tienen nodos hijo creados
- $A_{unexpanded}(s) = A(s) - A_{expanded}(s)$

2. Seleccionar una acción $a \in A_{unexpanded}(s)$, generalmente de manera uniforme aleatoria.
3. Aplicar la función de transición para obtener el nuevo estado: $s' = T(s, a)$.
4. Crear un nuevo nodo para s' con estadísticas iniciales:
   - $N(s') = 0$ (contador de visitas)
   - $W(s') = 0$ (suma acumulada de valores)
5. Establecer la conexión padre-hijo entre s y s' en el árbol parcial $\mathcal T^*$.
6. Añadir a al conjunto $A_{expanded}(s)$ para registrar que esta acción ya ha sido expandida.

Variantes y optimizaciones:

- **Expansión progresiva:** En juegos con factores de ramificación extremadamente grandes (como Go), se puede limitar el número de hijos expandidos proporcionalmente a $N(s)^\alpha$ para algún $\alpha \in (0,1)$.
- **Expansión por lotes:** Expandir múltiples nodos hijo en una sola iteración para mejorar la eficiencia en arquitecturas paralelas.
- **Expansión priorizada:** Utilizar conocimiento del dominio para expandir primero las acciones más prometedoras.

3.2.3 Fase de Simulación (Rollout)

Definición formal: Partiendo del nodo recién expandido s', se simula una partida completa hasta alcanzar un estado terminal $z \in Z$ siguiendo una política de simulación ρ, y se obtiene el vector de utilidad $u(z)$.

Proceso detallado:

1. Iniciar con el estado s' del nuevo nodo.
2. Repetir hasta alcanzar un estado terminal:
   - Determinar el conjunto de acciones legales $A(s')$.
   - Seleccionar una acción $a \in A(s')$ según la política de simulación ρ.
   - Actualizar el estado: $s' \leftarrow T(s', a)$.
3. Una vez alcanzado un estado terminal $z \in Z$, evaluar y retornar la utilidad $u(z)$.

Políticas de simulación ρ:

- **Aleatoria uniforme:** La más simple y computacionalmente eficiente, selecciona cualquier acción legal con igual probabilidad.
- **Heurística ligera:** Incorpora conocimiento del dominio para guiar las simulaciones:
  - **Capturas rápidas** en juegos de tablero como Go o Ajedrez
  - **Patrones conocidos** como respuestas a aperturas
  - **Reglas de sentido común** específicas del dominio
- **Políticas aprendidas:** Entrenadas mediante técnicas de aprendizaje automático a partir de partidas anteriores o auto-juego.

El balance crítico en esta fase es entre:

- **Calidad de simulación:** Qué tan bien la política ρ aproxima el juego óptimo
- **Velocidad de ejecución:** Cuántas simulaciones pueden realizarse por unidad de tiempo
- **Sesgo-varianza:** Políticas más informadas reducen la varianza pero pueden introducir sesgos

3.2.4 Fase de Retropropagación

Definición formal: Para cada nodo s en el camino desde el nodo expandido hasta la raíz, actualizar las estadísticas:

- $N(s) \leftarrow N(s) + 1$ (incrementar contador de visitas)
- $W(s) \leftarrow W(s) + u(z)$ (acumular la utilidad obtenida)
- $Q(s) = W(s)/N(s)$ (recalcular el valor empírico)

Proceso detallado:

1. A partir del nodo hoja (el último visitado en esta iteración), recorrer hacia arriba el camino hasta la raíz.
2. Para cada nodo s en el camino:
    - Incrementar su contador de visitas: $N(s) \leftarrow N(s) + 1$.
    - Actualizar su valor acumulado: $W(s) \leftarrow W(s) + u(z)$.
    - Recalcular su valor empírico: $Q(s) \leftarrow W(s) / N(s)$.
3. Para juegos de suma cero con dos jugadores, invertir la utilidad en cada nivel alternativo del árbol (perspectiva Negamax).

Variantes avanzadas:

- **Retropropagación con decaimiento temporal:** $W(s) \leftarrow \gamma \cdot W(s) + u(z)$ con $\gamma < 1$, dando menos peso a las simulaciones antiguas.
- **Actualización RAVE (Rapid Action Value Estimation):** Mantener estadísticas adicionales para movimientos equivalentes vistos en diferentes contextos.
- **Retropropagación ponderada:** Asignar diferentes pesos a las simulaciones según alguna medida de calidad o confianza.

PROF

3.3 Pseudocódigo del Algoritmo MCTS Completo

```
función MCTS(estado_inicial, presupuesto):
    crear nodo_raíz con estado = estado_inicial, N = 0, W = 0

    mientras no se agote presupuesto:
        // Fase 1: Selección
        nodo = nodo_raíz
        mientras nodo está completamente expandido y no es terminal:
            nodo = SeleccionarMejorHijo(nodo)

        // Fase 2: Expansión (si corresponde)
        si nodo no es terminal:
            nodo = ExpandirNodo(nodo)

        // Fase 3: Simulación
```

```
        resultado = SimularPartida(nodo.estado)

        // Fase 4: Retropropagación
        RetropropagarResultado(nodo, resultado)

    // Selección de acción final
    retornar acción que lleva al hijo con mayor N (o mayor Q) de
nodo_raíz

función SeleccionarMejorHijo(nodo):
    mejor_puntuación = -∞
    mejor_hijo = null

    para cada hijo de nodo:
        si N(hijo) = 0:
            retornar hijo  // Primera visita tiene prioridad

        puntuación_UCT = Q(hijo) + c·√(ln N(nodo) / N(hijo))

        si puntuación_UCT > mejor_puntuación:
            mejor_puntuación = puntuación_UCT
            mejor_hijo = hijo

    retornar mejor_hijo

función ExpandirNodo(nodo):
    acciones_posibles = AccionesLegales(nodo.estado) -
AccionesExpandidas(nodo)
    acción = SeleccionarAleatoriamente(acciones_posibles)

    nuevo_estado = AplicarAcción(nodo.estado, acción)
    nuevo_nodo = CrearNodo(estado = nuevo_estado, N = 0, W = 0, padre =
nodo)

    AñadirHijo(nodo, nuevo_nodo, acción)
    retornar nuevo_nodo

función SimularPartida(estado):
    estado_actual = CopiarEstado(estado)

    mientras estado_actual no es terminal:
        acciones_legales = AccionesLegales(estado_actual)
        acción = PolíticaSimulación(estado_actual, acciones_legales)
        estado_actual = AplicarAcción(estado_actual, acción)

    retornar EvaluarUtilidad(estado_actual)

función RetropropagarResultado(nodo, resultado):
    valor_actual = resultado
    nodo_actual = nodo

    mientras nodo_actual no es null:
        nodo_actual.N += 1
```

PROF

```
        nodo_actual.W += valor_actual

        // Para juegos de suma cero alternantes
        si EsJuegoSumaZero():
            valor_actual = -valor_actual

        nodo_actual = nodo_actual.padre
```

3.4 Criterios de Terminación y Selección Final

El algoritmo MCTS es un método "anytime", lo que significa que puede detenerse en cualquier momento y proporcionar una solución. Los criterios comunes para finalizar la búsqueda incluyen:

Criterios de terminación:

1. **Tiempo fijo:** Ejecutar durante un periodo predeterminado (e.g., 1 segundo).
2. **Número de iteraciones:** Realizar un número específico de simulaciones (e.g., 10,000).
3. **Umbral de confianza:** Continuar hasta que la diferencia entre las mejores acciones sea estadísticamente significativa.

Estrategias para la selección final de la acción:

1. **Mayor conteo de visitas ($\max_i N(s_i)$):** Esta es la estrategia más robusta, ya que:
    - Es menos sensible a valores atípicos y fluctuaciones estadísticas
    - Correlaciona directamente con la convergencia asintótica del algoritmo
    - Proporciona mejor desempeño empírico en la mayoría de los dominios
2. **Mayor valor empírico ($\max_i Q(s_i)$):**
    - Puede ser preferible cuando el presupuesto computacional es extremadamente limitado
    - Más sensible a la varianza de las simulaciones
    - Útil cuando la política de simulación tiene alta calidad
3. **Combinación ponderada:** $\arg\max_i [Q(s_i) + \lambda \cdot N(s_i)/N(s_0)]$
    - Equilibra el valor esperado con la frecuencia de visitas
    - El parámetro λ controla la importancia relativa de cada componente

PROF

El fundamento teórico para preferir el conteo de visitas radica en el hecho de que, asintóticamente, MCTS visitará con mayor frecuencia las acciones óptimas, haciendo que el conteo sea un estimador más robusto que el valor promedio, especialmente para distribuciones de recompensa con alta varianza.

3.5 Complejidad Computacional

La complejidad práctica del algoritmo MCTS depende principalmente de:

Por iteración:

- **Tiempo:** $O(D)$ donde D es la profundidad promedio de las simulaciones
- **Espacio:** $O(1)$ incremental por iteración (un nuevo nodo)

Total (después de I iteraciones):

- **Tiempo:** $O(I \cdot D)$
- **Espacio:** $O(I)$ en el peor caso, pero típicamente mucho menor ya que muchas iteraciones visitarán nodos existentes

La eficiencia del algoritmo se debe a su naturaleza asimétrica: concentra recursos computacionales en regiones prometedoras del árbol mientras mantiene una exploración suficiente de alternativas.

---

4 Ejemplo Desarrollado: Tic-tac-toe

4.1 Configuración

- **Política predeterminada** ρ = movimiento legal aleatorio uniforme.
- **Constante de exploración** $c=\sqrt2$.

4.2 Primeras Seis Jugadas (ilustrativo)

Jugada	Trayectoria seleccionada	Ganador del rollout	Estadísticas actualizadas de la raíz (N, Q para centro)
1	centro	empate	$(1, 0)$
2	esquina – centro	X	$(2, +0.5)$
3	esquina	X	$(3, +0.67)$
4	borde	O	sin cambios (centro no visitado)
5	centro	empate	$(4, +0.5)$
6	esquina	X	$(5, +0.6)$

Después de unos cientos de jugadas, el centro se convierte en la apertura preferida de manera abrumadora, coincidiendo con el juego óptimo. Incluso este pequeño juego revela la dinámica de exploración/explotación: los movimientos con éxito temprano atraen más muestras, pero el bonus de raíz cuadrada garantiza que cada movimiento se muestrea infinitas veces.
Visualizacion de MTC tic-tac-toe

PROF

5 Mejoras y Variantes

Categoría	Técnica	Idea	Beneficio Típico
Política de árbol	RAVE / AMAF	Compartir estadísticas entre movimientos equivalentes vistos más tarde en los rollouts.	Acelera el aprendizaje temprano (Go).

Categoría	Técnica	Idea	Beneficio Típico
	PUCT	Añadir probabilidad previa p_i al término UCT: $Q_i + c,p_i,\frac{\sqrt N}{1+n_i}$.	Integra redes neuronales (AlphaZero).
Política de jugada	ρ heurística	Sesgar movimientos aleatorios hacia heurísticas del dominio.	Reduce la varianza.
Expansión	Ensanchamiento progresivo	Retrasar la adición de hijos cuando $	A(s)
Paralelismo	Pérdida virtual	Restar temporalmente recompensa a lo largo de una trayectoria siendo explorada por otro hilo.	Decorrelación de hilos.

6 Propiedades Teóricas

- **Solidez.** Para juegos deterministas y finitos, y cualquier desempate que garantice exploración infinita, $Q(s) \to V(s)$ (el valor minimax) casi seguramente cuando rollouts $\to\infty$ (Kocsis & Szepesvári 2006).
- **Límite de arrepentimiento.** La regla UCT simple produce un arrepentimiento simple esperado de $\tilde O(\sqrt{\ln N / n})$ por nodo.
- **Comportamiento en cualquier momento.** El error empírico disminuye monótonamente con el recuento de jugadas, por lo que el algoritmo puede detenerse en cualquier momento.

---

7 Resumen de Complejidad

Recurso	Por rollout	Después de n rollouts
Tiempo	$O($profundidad promedio$)$	$O(n,$profundidad$)$
Memoria	$O($nodos expandidos$)$	Hasta $O(n)$ (típicamente mucho menos)

PROF

Para juegos pequeños como tic-tac-toe, el árbol completo cabe en memoria; para Go, la memoria se convierte en el recurso limitante antes que la CPU.

> **Mensaje clave.** *La Búsqueda en Árbol de Monte Carlo navega el árbol de juego aprovechando la aleatoriedad: los retornos empíricos impulsan las estimaciones de valor; los bonos de confianza dirigen la exploración; y, con suficientes simulaciones, el algoritmo converge al juego óptimo sin ver jamás la mayor parte del árbol.*

8 Expansiones

1. Que pasa en el ajedrez? No puede ser un arbol o dag, pues se pueden repetir estados desde otros estados o se puede regresar al estado anterior. Que pasa?

2. En el ajedrez y otros juegos jamas vemos el nodo terminal, entonces como calculamos Q? De donde viene Q?
3. Que pasa en juegos como el poker donde existe incertidumbre de estados?