MCTS aplicado al juego de Hex

Pablo Gómez Vidal

1. Introducción

Monte Carlo Tree Search (MCTS) es un enfoque poderoso para diseñar bots que jueguen juegos o para resolver problemas de decisión secuenciales. El método se basa en una búsqueda en árbol que equilibra exploración y explotación. MCTS realiza muestreos aleatorios en forma de simulaciones y almacena estadísticas de acciones para tomar decisiones más informadas en cada iteración subsiguiente.

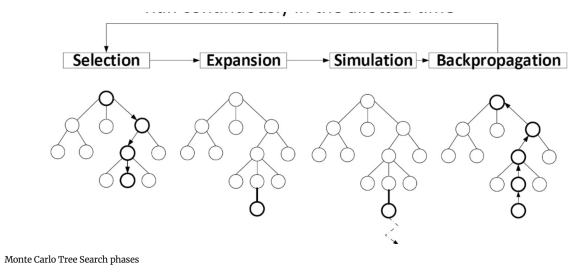
En problemas no triviales, el espacio de estados (por ejemplo, un árbol del juego de Hex) no puede ser explorado por completo. En aplicaciones prácticas, a MCTS se le asigna un presupuesto computacional que puede especificarse por el número de iteraciones(es lo que se implementó en este caso) o el tiempo disponible para tomar una decisión. MCTS es un algoritmo *anytime*. Esta propiedad significa que puede detenerse en cualquier momento y proporcionar la mejor acción actual.

1.1. ¿Por qué este algoritmo?

La principal causa por la que se decide usar **MCTS** en vez del Minimax con poda alfa-beta, es porque computacionalmente es menos costoso y al comparar una versión **mvp** de ambos algoritmos, este resultó no solo ser más rápido sino también más efectivo, lo cual introduce la segunda razón por la que se decide usar MCTS, la falta de una heurística fuerte para este juego, no me fue posible encontrar una heurística .ªbsoluta"debido a la naturaleza abstracta del juego, por lo que al usar Minimax era necesario unir diferentes heurísticas en diferentes etapas del juego para su mejora, mientras que MCTS no necesita una heurística explícita, ya que evalúa movimientos mediante simulaciones aleatorias hasta el final del juego. Esto evita sesgos o errores derivados de una heurística mal diseñada.

2. Fases del Algoritmo

El algoritmo MCTS consta de cuatro fases principales:



Monte Carlo Tree Search phases

2.1. 1. Selección

El algoritmo busca la parte del árbol que ya ha sido representada en la memoria. La selección siempre comienza desde el nodo raíz y, en cada nivel, selecciona el siguiente nodo según la

política de selección. Esta fase termina cuando se visita un nodo hoja y el siguiente nodo no está representado en la memoria aún o no se ha alcanzado un estado terminal del juego.

2.2. 2. Expansión

A menos que la selección haya alcanzado un estado terminal, la expansión agrega al menos un nuevo nodo hijo al árbol representado en la memoria. El nuevo nodo corresponde a un estado que se alcanza realizando la última acción en la fase de selección. Si la expansión alcanza un estado terminal, la iteración actual pasa directamente a la retropropagación.

2.3. 3. Simulación

Realiza una simulación aleatoria completa del juego, es decir, alcanza un estado terminal y obtiene las recompensas. Esta es la parte "Monte Carlo"del algoritmo.

2.4. 4. Retropropagación

Actualiza las recompensas (+1 en caso de victoria), para todos los agentes modelados en el juego, de regreso a todos los nodos a lo largo de la ruta desde el último nodo visitado en el árbol (la hoja) hasta la raíz.

3. Política de Selección UCT

El objetivo de la política de selección es mantener un equilibrio adecuado entre la **exploración** (acciones no bien probadas) y la **explotación** (mejores acciones identificadas hasta el momento). Es el algoritmo más común que se implementa en MCTS

$$UCT = \frac{Q_i}{N_i} + C\sqrt{\frac{\ln N_p}{N_i}}$$

Donde:

- Q_i: recompensa total del nodo i
- N_i: número de visitas al nodo i
- N_p: número de visitas al nodo padre
- C: constante de exploración (valor recomendado: $\sqrt{2}$ aunque termino usando 1,4)

4. ¿Cómo escogemos el nodo (jugada) que vamos a usar?

Se probaron dos métodos, luego de realizar las n iteraciones se probó devolver el nodo más visitado y devolver el nodo con más victorias acumuladas, se desarrolló una simulación donde se puso ambos agentes a competir y la mayoría de las veces el ganador fue el que devolvía el nodo con más visitas.

5. Deficiencia y Solución

5.1. La Deficiencia

La principal deficiencia de esta implementación es en tableros medianamente grandes como por ejemplo un $11x11$, y sobre todo al enfrentarse a algoritmos greedys o con eurísticas enfocadas solo en maximizar la ganancia del oponente, esto es debido a que en la fase inicial del juego

nuestro algoritmo aún no ha hecho muchas simulaciones para tener buenas estadísticas en cada acción, por lo que en la primera parte del juego las acciones son más random ya que acciones que pueden ser buenas en muchos contextos (como colocar una ficha en el centro, u obstruir el camino del contrario en un puente) no se aprovechan hasta que se visitan muchas veces esos contextos específicos, y en varios casos es muy tarde cuando el algoritmo se da cuenta

5.2. La Solución

Implementar el concepto de Rapid Action Value Estimation (RAVE), el cual es muy útil para la parte inicial del juego, ya que ahora en cada simulación que realicemos no solo guardaremos la jugada inmediata en el playout, sino que guardaremos todas las jugadas en ese playout, ya que aunque no sean la jugada inmediata, pueden tener más peso en la victoria que esta. RAVE introduce la idea de reusar información de jugadas que ocurrieron después en una simulación, incluso si no fueron la jugada inmediata. Un ejemplo simple: Imaginemos que en una simulación desde una posición A, jugamos las acciones en este orden:

A $\rightarrow$ jugada J1 $\rightarrow$ J2 $\rightarrow$ J3 $\rightarrow$ ... $\rightarrow$ resultado

En el MCTS clásico, solo se actualiza la estadística de J1 en el nodo A.

Con RAVE, también se actualizan las estadísticas de J2, J3, etc. en A, porque esas jugadas aparecieron en esa simulación, aunque no fueron la primera jugada desde A.

Estas estadísticas adicionales se llaman AMAF (All Moves As First). Se calcula para cada acción cómo le ha ido cuando ha aparecido en la simulación, sin importar el orden.

La idea intuitiva que plantea RAVE es la siguiente:

"No he probado esta jugada como primera, pero cada vez que apareció más tarde en la partida, gané"

5.3. Integración de RAVE a nuestro MCTS

Basta con mantener una constancia en el método de:

```
simulate_random_playout
```

de todos los movimientos hechos en la simulación y actualizar sus valores en la retropropagación. El otro cambio importante ocurre en nuestra función UCT la cual ahora cambia a:

$$\text{Score}(i) = (1 - \beta_i) \cdot \frac{Q_i}{N_i} + \beta_i \cdot \frac{Q_i^{\text{RAVE}}}{N_i^{\text{RAVE}}} + c \cdot \sqrt{\frac{\ln N_p}{N_i}}$$

Donde:

- $\frac{Q_i^{\text{RAVE}}}{N_i^{\text{RAVE}}}$: tasa de victoria RAVE del movimiento correspondiente a i (los nuevos valores que ahora tenemos en cuenta).
- $\beta_i = \sqrt{\frac{b}{b+3n_i}}$: peso que mezcla la información RAVE con la UCT tradicional
- b: constante de suavizado de RAVE (en este caso usé 314)

6. Conclusión

MCTS + RAVE es un algoritmo versátil que ha demostrado gran eficacia en diversos dominios, especialmente en juegos. Su capacidad para equilibrar exploración y explotación, junto con su naturaleza *anytime*, y la mejora del RAVE lo hacen particularmente útil en problemas con espacios de estado grandes y complejos como el caso del juego de Hex.