Universidad Simón Bolívar

Dpto. de Computación y Tecnología de la Información

Inteligencia Artificial I

Nathalia Silvera 12-10921

Carolina Rivas 13-11209

Informe Proyecto II

El objetivo de este proyecto es adquirir conocimientos sobre el modelo de árboles de juego y explorar algoritmos básicos de solución en el contexto del juego de Othello. Se ha utilizado una versión reducida del juego para evaluar y comparar el rendimiento de diferentes algoritmos.

Los algoritmos implementados y analizados en este proyecto son los siguientes:

1. **Negamax con poda alpha-beta:** Este algoritmo utiliza la técnica de búsqueda Negamax en combinación con la poda alpha-beta para mejorar la eficiencia de exploración del árbol de juego. La poda alpha-beta permite descartar movimientos que no afectarán el resultado final, reduciendo así el número de nodos a evaluar.

2. **Negamax sin poda alpha-beta:** Esta versión del algoritmo Negamax no utiliza la poda alpha-beta. Aunque puede ser menos eficiente en términos de tiempo de ejecución, nos permite evaluar su desempeño en comparación con la versión mejorada.

3. **Scout**: El algoritmo Scout, también conocido como Poda de Mejor-Primero, se basa en la idea de realizar una búsqueda selectiva en el árbol de juego. En lugar de explorar todos los movimientos posibles, Scout se enfoca en los movimientos más prometedores en función de la evaluación actual. Esto reduce significativamente el número de nodos evaluados, lo que puede llevar a una mayor eficiencia en la búsqueda.

4. **Negascout**: Este algoritmo, también conocido como Principal Variation Search (PVS), es una mejora del algoritmo Scout. Utiliza una variante de la búsqueda alpha-beta que realiza una exploración más precisa de las posiciones principales en el árbol de juego. Negascout sigue una estrategia de búsqueda en ventana estrecha, lo que significa que realiza múltiples búsquedas con intervalos cada vez más pequeños para refinar su estimación del valor de un movimiento.

El objetivo de este proyecto es comparar el rendimiento y la eficiencia de estos algoritmos en el contexto del juego de Othello y analizar cómo influyen en la toma de decisiones y en el resultado final del juego.

Algoritmos de recorrido en árboles de juego

Se ha eliminado el parámetro "use_tt" en todos los algoritmos mencionados anteriormente, ya que las tablas de transposición no se implementaron en el proyecto.

Al eliminar las tablas de transposición, se simplifican los algoritmos y se reduce la complejidad del código. Sin embargo, es importante tener en cuenta que la ausencia de tablas de transposición puede afectar el rendimiento y eficiencia de los algoritmos de búsqueda en juegos. Las tablas de transposición suelen ser útiles para evitar la repetición de cálculos y reducir la cantidad de exploración en la búsqueda del árbol de juego. Sin ellas, es posible que se realicen más cálculos y se recorran caminos redundantes en la búsqueda, lo que podría afectar la velocidad y eficacia del algoritmo.

En resumen, aunque la falta de tablas de transposición puede simplificar el código, también puede afectar el rendimiento de los algoritmos.

El tiempo que se estableció para la ejecución de cada algoritmo fue de dos horas, y los resultados se exponen a continuación, tomando en cuenta la profundidad que se alcanzó en cada algoritmo.

Algoritmos	**Profundidad nodo inicial**
Negamax	18
Negamax alpha-beta	12
Scout	10
Negascout	11

En la siguiente tabla se visualizan los resultados de las corridas para los algoritmos :

Algoritmos	**Nodos expandidos**	**Nodos generados**	**Nodos Generados por seg**	**Tiempo de ejecución**
Negamax	625084814	876269598	1.03098e+06	849.94
Negamax alpha-beta	2230058150	2931981147	869021	3373.89
Scout	1362837972	3132439617	1.0279e+06	3047.4,
Negascout	2569420749	3362309726	836177	4021.05

***Para ver los resultados completos de todos los algoritmos, ver Resultados.txt**

Como se puede observar

El algoritmo de Negamax sin poda solo fue capaz de lograr llegar a una profundidad de 18, mientras que el algoritmo Negamax con poda tuvo un mejor desempeño, logrando alcanzar una mayor profundidad.

Al comparar los resultados de los algoritmos de Scout y Negascout, podemos notar que Scout tuvo un mejor desempeño en términos de tiempo de ejecución. Esto puede deberse a que , el algoritmo Scout logró encontrar y aprovechar las ventajas de los movimientos disponibles para ambos jugadores, lo que permitió tomar decisiones más rápidamente y expandir menos nodos en la búsqueda. Por otro lado, el algoritmo Negascout , no encontró posiciones donde existiera ventajas para ninguno de los jugadores , lo que llevó a una mayor exploración de nodos y, en consecuencia, a un mayor tiempo de ejecución.

Los resultados obtenidos en las pruebas confirman las expectativas, ya que los algoritmos Negascout y Scout demostraron ser muy eficientes en términos de rendimiento. Específicamente, Scout se destacó como el más eficiente entre todos.

En contraste, el algoritmo Negamax mostró un desempeño menos eficiente, llegando incluso a ser considerado ineficiente en comparación con los otros algoritmos evaluados. Esto indica la importancia de aplicar técnicas de poda, como el algoritmo alpha-beta, para reducir la cantidad de nodos explorados y mejorar el rendimiento en la búsqueda.

Entre los algoritmos evaluados, Scout se destacó como el más eficiente en términos de tiempo de ejecución. Este resultado puede atribuirse al hecho de que durante la ejecución del algoritmo se presentaron ventajas de movimientos para ambos jugadores, lo que permitió tomar decisiones más rápidas y expandir menos nodos en la búsqueda.

En resumen, los resultados de las pruebas respaldan la eficiencia y efectividad de los algoritmos Negascout y Scout, demostrando que son opciones sólidas para la búsqueda en juegos y problemas similares. Por otro lado, el algoritmo Negamax sin poda se mostró menos eficiente y podría beneficiarse de técnicas de poda para mejorar su rendimiento.