Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Inteligencia Artificial I (CI5437)

Proyecto #2
Othello

Integrantes:
Andreina García 08-10406
José Antonio Rodríguez 08-10971
Pedro Romero 08-10992

Sartenejas, 6 de noviembre de 2012

El objetivo de este proyecto es probar el desempeño, entre los algoritmos de búsqueda en grafos conocidos como minimax, alpha-beta pruning y negascout, para resolver el juego de Othello de forma tal que ambos jugadores (MIN y MAX) muevan sus piezas de la mejor forma posible.

Othello es un juego de estrategia para dos jugadores en un tablero de 6x6 donde cada jugador por turnos coloca una ficha por su cara blanca o negra según el color del jugador. En el juego cada vez q se coloque una pieza se voltearán todas las piezas adyacentes a ésta que sean del otro color hasta llegar a una del color de la pieza jugada. Si no hay otra pieza del mismo color de la pieza a colocar en la fila, columna o alguna de las diagonales de la posición donde se quiere jugar entonces esa jugada no es posible. Gana el jugador que, al no quedar espacios libres en el tablero, tenga la mayor cantidad de fichas colocadas.

Los puntos del juego serán llevados en base al jugador MAX (en este caso las fichas negras). Esto quiere decir que si el jugador MAX gana el juego la puntuación será positiva, si el jugador MIN (fichas blancas) gana será negativo y si ocurre un empate será cero. En el juego perfecto el resultado es -4 lo que significa que siempre gana el jugador MIN con 4 puntos de ventaja. El tablero inicial contiene 4 fichas colocadas, dos negras en las posiciones 1 y 2 y dos blancas en las posiciones 0 y 3. El jugador MAX comienza el juego. El juego perfecto viene dado por la secuencia de movimientos PV = { 12, 21, 26, 13, 22, 18, 7, 6, 5, 27, 33, 23, 17, 11, 19, 15, 14, 31, 20, 32, 30, 10, 25, 24, 34, 28, 16, 4, 29, 35, 36, 8, 9, -1 } donde las casillas están organizadas de la siguiente manera:

4	5	6	7	8	9
10	11	12	13	14	15
16	17	0	1	18	19
20	21	2	3	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Minimax es un algoritmo de juego recursivo más simple y usado entre dos oponentes perfectos, es decir dos jugadores expertos en el juego; en el que se representa a ambos jugadores como Max, el cual representa las jugadas a favor; y Min, el cual representa las jugadas negativas realizadas por el oponente. El árbol de juego utilizado por el algoritmo inicia con una jugada de Max (nodo raíz), y se realiza una búsqueda limitada en profundidad en el árbol para poder calcular todas las posibles jugadas dentro del límite establecido alternando las jugadas de los jugadores al ir descendiendo en la profundidad del árbol. El tiempo empleado para explorar el árbol es aproximadamente igual al número de ramificaciones o posibles jugadas que tenga en el momento cada jugador exponencial a la profundidad de la búsqueda.

Alpha-beta pruning es un algoritmo de búsqueda recursivo que intenta disminuir el número de nodos expandidos en el árbol de juego por el algoritmo minimax al encontrar el valor del juego perfecto de Othello. Este algoritmo mejora el tiempo del algoritmo minimax al no explorar ramas del árbol de juego que el “sabe”, gracias al control de los valores alpha y

beta, no aportarán nada al resultado del juego.

Negascout es un algoritmo de búsqueda, variante del Minimax, que disminuye el número de nodos expandidos usando la ventana nula. Este algoritmo mejora los tiempos del Minimax al no explorar ramas que no darán una mejor solución, esto se sabe gracias a los valores de alpha y beta.

Dichos algoritmos utilizan una heurística que permite optimizarlos y así obtener buenos resultados en buenos tiempos de ejecución. La heurística utilizada en este proyecto es el valor del juego, es decir, la resta entre fichas que posee el jugador MAX y fichas que posee el jugador MIN.

Para implementar dicho proyecto utilizamos el lenguaje C++. Contamos con el archivo othello_cut.h que define la clase state_t que representan los estados del tablero de juego y se definen funciones sobre estos que utilizarán los algoritmos de búsqueda. Por el diseño trabajado en sí de los tres algoritmos, no utilizamos un tipo de estructura en especial para almacenar los diferentes estados state_t generados y expandidos, lo cual permitió que el uso de memoria principal sea nulo. También se cuenta con el archivo main.cc que realiza la ejecución de los algoritmos; sus instrucciones de uso se encuentran en el archivo README.

A continuación presentamos unas tablas comparativa con datos recopilados de la ejecución de cada algoritmo sobre árboles de juego de Othello de diferentes tamaños. En cada tabla, la columna de pasos previos se refiere al número de jugadas realizadas en la variante principal PV que generan el estado state_t que es pasado al algoritmo elegido que completará la ejecución hasta conseguir el mejor valor del juego. Para cada algoritmo se presentan los nodos creados y expandidos por el mismo, lo que resulta un factor muy influyente en los tiempos alcanzados. Todos los algoritmos encontraron el valor del juego perfecto igual a -4, donde gana el jugador MIN por 4 puntos.

Minimax			
Pasos previos	Tiempo	Expandidos	Creados
30	0m0.003s	3	5
28	0m0.004s	9	13
26	0m0.003s	111	163
24	0m0.015s	2701	4031
22	0m0.174s	44989	66688
20	0m7.135s	2062071	3015557
18	3m10.652s	53382964	78915640
16	163m8.201s	int overflow	int overflow
14	más de 10 horas	-	-
12	-	-	-
10	-	-	-

Alpha-beta pruning			
Pasos previos	Tiempo	Expandidos	Creados
30	0m0.004s	3	5
28	0m0.004s	9	13
26	0m0.005s	55	75
24	0m0.011s	661	895
22	0m0.021s	2679	3579
20	0m0.234s	64550	87284
18	0m0.612s	171682	234269
16	0m4.616s	13291136	1811275
14	1m36.845s	28813457	39439081
12	17m43.920s	277692464	378528258
10	235m49.590s	int overflow	int overflow

Negascout			
Pasos previos	Tiempo	Expandidos	Creados
30	0m0.004s	3	5
28	0m0.004s	13	17
26	0m0.005s	92	121
24	0m0.023s	2153	2806
22	0m0.038s	9674	12579
20	0m0.496s	149467	194160
18	0m2.490s	749273	982272
16	0m26.176s	7878343	10436043
14	22m50.562s	401714181	535296706
12	289m44.954s	1206704601	int overflow
10	más de 5 horas	-	-

De estas tablas se puede inferir que el algoritmo minimax está generando un gran número de nodos ya que este explora todo el árbol de juego antes de tomar la decisión de cuál es el valor de la mejor jugada. A medida que se explora un mayor árbol, minimax genera un número exponencial de nodos. En cambio alpha-beta pruning, mediante comparaciones entre los valores alpha y beta, hace podas del árbol y evita visitar aquellas ramas que sabe no agregarán información al resultado obtenido hasta ese momento; por eso este algoritmo genera menos nodos de una manera más pausada y es mucho más rápido que minimax sin generar errores ni perder ninguna información a la hora de calcular el valor del mejor juego.

Por otro lado el algoritmo negascout, que es una combinación de los dos anteriores, mejora los tiempos de minimax al hacer podas enviando ventanas vacías y controlando los valores alpha y beta, pero aun así no mejora los tiempos de alpha-beta pruning porque se apega al principio de minimax y termina evaluando más nodos de los estrictamente

necesarios. Negascout podría mejorar los tiempos de alpha-beta pruning si visitara los nodos en orden de acuerdo a su heurística, lo que permitiría que se dé cuenta de cuales ramas deben podarse con mayor rapidez.

Referencias:

http://ldc.usb.ve/~bonet/courses/ci5437/othello/
http://www.cs.cornell.edu/~yuli/othello/othello.html