Inteligencia Artificial I-Proyecto II
Othello

Fabio Castro 10-10132, María Jorge 11-10495, Jorge Marcano 11-10566

08/06/2015

1. Introducción

En teoría de juego, un árbol de juego es un grafo dirigido de tipo árbol cuyos nodos representan posiciones en el juego y cuyas aristas representan movimientos de los jugadores. Cualquier sucesión de jugadas puede representarse por un camino conexo dentro del árbol de juego. Si el juego acaba siempre después de un número finito de pasos, entonces el árbol tiene un número finito de nodos.

Sabemos que existen juegos de dos personas en donde cada persona juega por un turno y el objetivo es jugar de la mejor manera posible. Este tipo de juegos se pueden representar con un árbol de decisión, especificamente en uno llamado "And Or", que está constituido en cada nivel del árbol solamente por nodos tipo "And" o solamente tiene nodos tipo "Or".

La Computación toma su papel en estos juegos a la hora de calcular la mejor manera posible de jugar para cada jugador. En el caso de un juego de ajedrez, podemos calcular la mejor manera de jugar buscando hacer el jaque mate lo más rápido posible, o simplemente asegurar un jaque mate en una jugada futura.

En nuestro caso haremos uso del juego Othello, el cual 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 que le corresponda a cada jugador. En el juego cada vez que 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. De manera intuitiva también podemos decir que muchas veces es mejor buscar una solución cercana a la óptima, que encontrar la solución óptima, ya que recortamos el espacio de búsqueda.

El objetivo de este proyecto es probar el desempeño, entre los algoritmos de búsqueda en grafos conocidos como Negamax, Negamax alpha-beta, Scout 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.

2. Algoritmos

Negamax es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario. Este cálculo se hace de forma recursiva. El funcionamiento de Negamax puede resumirse en cómo elegir el movimiento que más te favorezca, suponiendo que tu contrincante escogerá el peor para ti. 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.

El problema de la búsqueda Negamax es que el número de estados a explorar es exponencial al número de movimientos. Partiendo de este hecho, la técnica de poda alfa-beta trata de eliminar partes grandes del árbol, aplicándolo a un árbol Negamax estándar, de forma que se devuelva el mismo movimiento que devolvería este, gracias a que la poda de dichas ramas no influye en la decisión final

Negascout es un algoritmo de búsqueda, variante de 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.

Por su parte Scout es un algoritmo que busca recortar el espacio de búsqueda haciendo uso de una función auxiliar llamada TEST, y recordando los parientes de un nodo del árbol para su beneficio.

Todos los algoritmos utilizan heurísticas que permite optimizarlos y así obtener buenos resultados en buenos tiempos de ejecución. En este proyecto utilizaremos el valor del juego como heurística, es decir, la resta entre fichas que posee el jugador MAX y fichas que posee el jugador MIN.

3. Implementación

Para la implementación utilizamos el lenguaje C++. Contamos con el archivo othello_cut.h, donde se define la clase state_t, que representa los estados del tablero de juego. Además se definen funciones sobre estos necesarias para el funcionamiento de los algoritmos de búsqueda.

Para la implementación de los algoritmos realizamos un archivo para cada uno de ellos y utilizamos el contenido del archivo main.cc para poder realizar las corridas desde estados de la variación principal, es decir, dependiendo de la profundidad que se deseara probar, avanzábamos en la variación principal hasta llegar al estado ïnicial"desde donde empezaría la ejecución de alguno de los algoritmos. Además, la salida por pantalla indica el tiempo de ejecución de la corrida, los nodos generados y los nodos finales alcanzados.

A continuación presentamos tablas comparativas con datos recopilados de la ejecución de cada algoritmo sobre árboles de juego de Othello de diferentes tamaños. Para cada algoritmo se presentan los nodos generados, nodos finales alcanzados y el tiempo de ejecución. Todos los algoritmos encontraron el valor del juego perfecto igual a 4, donde gana el jugador MIN por 4 puntos. Para la realización de las tablas se tomó como cota de tiempo máximo 10 minutos, por tanto, si una corrida no terminaba en esa duración de tiempo, ninguna otra que estuviese más cerca de la raíz de la variación principal lo haría. Por esta razón tomamos el último nivel en el que se pudo resolver el problema.

4. Resultados

		NegaMax			NegaMaxAB		
nivel	val	#gen	#fin	tiempo	#gen	#fin	tiempo
30	-4	5	2	0.000012s	1	1	0.000005s
29	4	12	4	0.000028s	4	2	0.000011s
28	-4	13	4	0.000031s	5	2	0.000012s
27	4	90	27	0.000180s	12	4	0.000029s
26	-4	176	52	0.000350s	13	4	0.000031s
25	4	1048	305	0.002176s	26	6	0.000058s
24	-4	4497	1330	0.009451s	81	20	0.000173s
23	4	11977	3381	0.024579s	237	52	0.000534s
22	-4	76825	21699	0.163949s	1002	234	0.002261s
21	4	428401	119923	0.908753s	1501	350	0.003440s
20	-4	3478734	953486	7.392386s	4067	900	0.009746s
19	4	13078932	3619363	27.637522s	9129	2099	0.023140s
18	-4	90647894	25526376	189.487928s	98754	22734	0.230288s
17	4	NT	NT	NT	127643	29515	0.303847s
16	-4	NT	NT	NT	267603	62587	0.629561s
15	4	NT	NT	NT	1259429	299087	2.973716s
14	-4	NT	NT	NT	2031923	482139	4.846172s
13	4	NT	NT	NT	29501797	7176690	67.372291s
12	-4	NT	NT	NT	43574642	10625624	100.456035s
11	4	NT	NT	NT	107642870	25626713	247.318001s
10	-4	NT	NT	NT	NT	NT	NT
9	NT	NT	NT	NT	NT	NT	NT
8	NT	NT	NT	NT	NT	NT	NT

		Scout			NegaScout		
nivel	val	#gen	#fin	tiempo	#gen	#fin	tiempo
30	-4	1	1	0.000006s	1	1	0.000006s
29	4	4	2	0.000011s	4	2	0.000011s
28	-4	5	2	0.000011s	5	2	0.000012s
27	4	7	6	0.000040s	17	6	0.000040s
26	-4	18	6	0.000042s	18	6	0.000042s
25	4	31	8	0.000070s	31	8	0.000070s
24	-4	81	20	0.000179s	81	20	0.000169s
23	4	389	88	0.000871s	389	86	0.000859s
22	-4	1804	445	0.004049s	1631	393	0.003588s
21	4	2717	655	0.006005s	2421	571	0.005433s
20	-4	4229	955	0.009825s	3843	847	0.008934s
19	4	14900	3544	0.033689s	11979	2756	0.027654s
18	-4	78268	17770	0.182093s	48477	10728	0.113865s
17	4	202034	46661	0.467732s	81631	18326	0.190300s
16	-4	314746	72717	0.731536s	184144	41839	0.431409s
15	4	850441	199368	1.972837s	605947	140374	1.419510s
14	-4	1407448	331087	3.264474s	1134288	264924	2.643214s
13	4	8367183	1995588	19.235866s	7221811	1706005	16.623680s
12	-4	42907223	10308619	97.288906s	25831374	6129025	59.355238s
11	4	83581446	19674433	189.939757s	62051335	14453944	141.531664s
10	-4	NT	NT	NT	242584981	57293575	541.282530s
9	NT	NT	NT	NT	NT	NT	NT
8	NT	NT	NT	NT	NT	NT	NT

5. Conclusiones

De las tablas presentadas de las ejecuciones de los algoritmos podemos inferir que el algoritmo Negamax es el más ineficiente, esto se debe a que el espacio de búsqueda es demasiado grande, ya que este recorre todo el árbol de juego antes de tomar una decisión. Con la cota de tiempo establecida, fue el algoritmo que menos niveles pudo resolver, llegando al nivel 18.

Lo ideal es usar algoritmos que realicen podas sobre el espacio de búsqueda como lo son los algoritmos Negamax con poda alpha beta, Scout o Negascout, que logran podar significativamente el espacio de búsqueda. El mejor algoritmo de los utilizados, fue Negascout, seguido de Scout, NegaMax Alpha beta y por último Negamax. Finalmente, el secreto de Negascout radica en que asume que el nodo actual que está abierto es el mejor y que se encuentra en la variación principal, buscando así si es cierto con una ventana nula de alpha beta, lo que termina siendo mucho más rápido que una busqueda con alpha beta normal.

Referencias

http://ldc.usb.ve/ bonet/courses/ci5437/othello/
http://www.cs.cornell.edu/ yuli/othello/othello.html
http://elvex.ugr.es/decsai/algorithms/slides/problems/Games.pdf
http://profesores.elo.utfsm.cl/ tarredondo/info/soft-comp/Arboles%20de%20Decision.pdf
http://es.wikipedia.org/wiki/Negascout