Universidad Simón Bolívar.
Departamento de Computación y Tecnología de la Información.
Asignatura: CI5437 - Inteligencia Artificial I.
Profesor: Blai Bonet

Informe Proyecto 2
(Árboles de Juego - Othello)

Realizado por:
José Antonio Cipagauta (05-38040)
Ricardo Lira (08-10616)
Daniela Ortiz (10-10517)

Sartenejas, 2016.

Descripción de las Actividades Realizadas, Conclusiones y Análisis de los Resultados

Descripción del Juego Othello:

Othello es un juego entre dos personas (un jugador con fichas blancas y otro con fichas negras), que van colocando por turnos sus fichas en el tablero, con la intención de cambiar el color de las fichas del adversario, para así, obtener la mayor cantidad de fichas de su color sobre el tablero.

La representación de este juego viene dada por 4 matrices, que representan las filas, columnas, diagonal principal y diagonal secundaria del tablero. Para nuestro caso, el cual fue un tablero de 6x6, cada matriz contiene 32 elementos, ya que al inicio del juego se colocan 4 fichas (2 blancas y 2 negras) en el centro del tablero. Las posiciones dentro del tablero, están numeradas de la siguiente forma:

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

Las posiciones fuera del tablero se representan con -1.

Actividad 1:

Para la realización de la actividad 1 únicamente se completó la representación del juego. Para ello, se completó la función move para que cuando se hiciera una jugada, se modificaran las diagonales al color correspondiente, y outflanked para que se verificaran las diagonales y determinar si es posible realizar la jugada en una posición determinada gracias a la distribución de colores en las diagonales.

Actividad 2:

Se realizó la implementación de los 5 algoritmos pedidos:

- Min-max/Max-min doblemente recursivo.
- Versión Negamax de Min-max/Max-min.
- Versión Negamax de Min-max/Max-min con poda alpha-beta.
- Scout.
- Negascout (Negamax con poda alpha-beta + Scout).

Además, se realizaron corridas de 10 minutos con cada uno de ellos. Los resultados obtenidos se encuentran respectivamente en los archivos:

- minmax_results.txt
- negamax_results.txt
- negamax_abp_results.txt
- scout_results.txt
- negascout_results.txt

Actividad 3:

Análisis de los Algoritmos

Todos los algoritmos recorrieron la variación principal con un resultado de -4, mostrando así que al final gana el jugador 2, es decir, el jugador blanco. Los algoritmos siempre reportaron -NaN como número de nodos generados en la primera iteración, este error se debe a la forma en como están calculados los segundos, ya que al usar la función de utils.h, el tiempo no es preciso, y sólo está en segundos. Por su parte, la primera iteración donde no se genera ningún nodo reporta 0/0, lo cual es NaN. Las siguientes iteraciones al sólo tomar tiempo poco significativo, se reportan como 0 segundos, y al hacer la división de n/0, muestra infinitos nodos generados por segundo. Sin embargo, luego en alguna iteración con tiempo más significativo, entonces el reporte de nodos generados por segundo toma así alguna medida razonable.

De los algoritmos usados el más exitoso fue Negascout, logrando llegar hasta el paso 12 de la variación principal, seguido de Scout llegando a 12 también pero tomando más tiempo y generando más nodos, luego Negamax con alpha-beta pruning llegando a 13, seguido por Negamax(minmax) llegando sólo al paso 18 y finalmente Minmax-Maxmin tambien llegando sólo al paso 18, pero de nuevo tomando más tiempo y generando la misma cantidad de nodos.

El algoritmo Negascout es de esperarse que tenga el mayor éxito debido a que posee un factor de ramificación mucho menor, y por lo tanto no recorre nodos innecesarios, que no sean jugadas óptimas. Scout al poseer las mismas características, tiene gran éxito pero no tanto como Negascout. Pudimos notar, que si los algoritmos fuesen ejecutados por una mayor cantidad de tiempo, no sólo 10 minutos, se podría apreciar una mayor diferencia entre ambos algoritmos. Negamax con alpha-beta pruning les sigue de cerca a los dos anteriores, ya que éste también realiza una poda de una gran cantidad de ramas innecesarias a explorar. Es de esperarse que los otros algoritmos que no realizan ninguna poda (Negamax Versión Min-max/Max-Min y Min-max/Max-Min), lleguen a mucha menos profundidad, y que además, por esta razón aumente exponencialmente la cantidad de nodos generados y expandidos por estos algoritmos.

También era de esperarse que el algoritmo Minmax-Maxmin fuese el de menor éxito, no sólo porque no realiza podas, sino porque este algoritmo compara los valores recibidos, dependiendo de si es un máximo o un mínimo, mediante una evaluación doblemente recursiva, la cual toma más tiempo. De igual forma, es evidente que el algoritmo Negamax(minmax) sería un poco mejor que Minmax-Maxmin (pero igualmente poco exitoso), ya que realiza la misma comparación anterior pero valiéndose de la propiedad Max{a,b} == -Min{-a,-b}, para eliminar la doble recursión.

Entre los algoritmos Negamax con alpha-beta pruning y Scout, la diferencia en la poda radica en que Negamax compara entre los valores (máximo y mínimo) de ambos jugadores

para saber cuando puede realizar una poda, mientras que Scout compara con los valores de todos los hijos para seleccionar el mejor, por lo que la poda de este algoritmo resulta más grande (lo cual influye directamente en el tiempo y cantidad de nodos generados y expandidos por el mismo). Sin embargo, como este algoritmo además debe hacer la evaluación y verificación de estados correspondiente a la función Test, toma más tiempo y genera y expande más nodos que el algoritmo Negascout, el cual aprovecha las ventajas combinadas de Negamax con alpha-beta pruning y Scout.

Es interesante notar la cantidad de nodos generados. Todos los algoritmos generan aproximadamente la misma cantidad de nodos por segundo, el éxito de los algoritmos radica principalmente en la eliminación de duplicados, y/o ramas innecesarias a explorar, en pocas palabras el que hace una búsqueda más inteligente es el que tiene mayor éxito.

Los algoritmos que no podan alguna rama generan nodos de una forma sólo generan un poco más rápida que los otros, pero necesitan una gran cantidad de nodos generados y expandidos para llegar a algún resultado. Con estas dos observaciones se puede concluir que lo mejor es simplemente explorar de la forma más eficientemente posible, y no explorar la mayor cantidad de nodos, debido a que esto al final nos daría un resultado útil, en una fracción de tiempo útil.