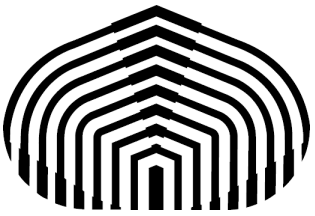


UNIVERSIDAD SIMÓN BOLÍVAR

CI5437

Informe Proyecto #2

Estudiantes:

Pietro Iaia 15-10718

Diego Peña 15-11095

Sartenejas, Junio 2021

Introducción

Uno de los problemas más estudiados en el campo de la Inteligencia Artificial es el desarrollo de algoritmos que sean capaces de desarrollar estrategias óptimas para juegos. Los juegos por turno, particularmente los conocidos como “zero-sum games”, han sido objeto de muchos trabajos que han propuesto distintos algoritmos con distintos enfoques para obtener estrategias óptimas de la manera más eficiente posible.

El objetivo de este informe es comparar los resultados de cuatro algoritmos distintos para búsqueda en árboles de juego sobre un mismo juego: Othello, en su versión 6x6

Metodología

Se utilizaron cuatro algoritmos de búsqueda en árboles de juego sobre el juego Othello 6x6:

- Negamax
- Negamax con poda alfa-beta
- Scout
- Negascout

Para cada algoritmo se utilizó de referencia el pseudocódigo dado en clase para cada uno y se hicieron las adaptaciones correspondientes para poder implementarlos en C++. La implementación de Othello en su mayoría fue suministrada por el profesor, con algunos puntos que requerían completarse. Para cada algoritmo se midió:

- Número de casos resueltos
- Cantidad de nodos expandidos
- Cantidad de nodos generados
- Tiempo
- Tasa de nodos generados por segundo

El valor de la heurística que se asignaba a un estado se determinaba de la siguiente manera: Se sumaba un punto por cada ficha negra y se restaba uno por cada ficha blanca. Esto acotaba el valor de la heurística entre -36 y 46.

Para los casos de prueba, se tomó como referencia la variación principal del juego, propuesta por Sakuta & Iida, 2001. La misma es una partida de Othello en la cual cada

jugador, en cada turno hace la mejor jugada posible. Esto genera 34 estados del juego distintos, contando el estado inicial y final. El caso final tiene identificado 34 y el inicial 1. Como cada jugada de la partida es óptima, al correr estos algoritmos sobre cada uno de estos tableros, se debe llegar al mismo estado final. En el mismo, las fichas blancas ganan por cuatro, por lo que el valor final para cada ejecución debe ser -4. Se asocia el signo positivo con las piezas negras y negativo con las piezas blancas.

La mayoría de los algoritmos (Los tres que empiezan con el prefijo "nega") están diseñados para maximizar siempre el valor heurístico independientemente de quien juegue. El detalle está en que cuando se analizan las posibles jugadas del oponente, el valor heurístico que se maximiza tiene el signo cambiado, lo que es equivalente a minimizar utilizando los valores con su signo sin modificar. Esta inversión genera que los algoritmos devuelvan 4 o -4 dependiendo de a quién le tocaba jugar cuando se hizo la primera llamada. Para evitar que en el output se vieran reflejadas estas inversiones de signo, el código suministrado por el profesor para el output multiplica este valor por el signo correspondiente a ese color. Eso causó algunos problemas con Scout, que en su versión original no invertía signos (Scout maximiza o minimiza el valor para cada jugador de la manera tradicional, dependiendo de quién lo llamó originalmente), por lo que siempre devolvía -4 y al multiplicar por el signo no se obtenía el efecto deseado. Para que no hubiera variación con respecto al comportamiento de los demás algoritmos, se hicieron algunas modificaciones en los signos de la variable score con la que trabaja Scout. Para evitar que el procedimiento Test (Se usa para podar nodos en Scout) utilizara valores con signo invertido, lo cual daba errores, en las llamadas a Test que se hacen directamente dentro de Scout, se multiplico el valor de "score" por el signo correspondiente al jugador. Esto hace que Test siempre use el valor original de la heurística sin alteraciones.

Cada algoritmo se ejecutó partiendo del caso 34 (Estado final resuelto) y fue retrocediendo hasta llegar a un caso en el que pasara media hora y no pudiera resolverlo. En este punto, se detenía al algoritmo y se reinciciaba el experimento con otro algoritmo hasta que no quedaran más. Se colocó profundidad máxima 70 o más a todos los algoritmos ya que es imposible que un juego de Othello tenga más jugadas: AL principio de cada partido quedan 32 casillas, suponiendo que la mitad de las jugadas es "pasar" y la otra mitad es poner una ficha, esto da 64 movimientos. Este caso probablemente no puede ocurrir, porque solo se puede pasar cuando no hay otra opción, pero es una cota superior a cualquier caso posible.

Recursos

- Google Colab, con 12 GB de RAM (Para Scout y Negascout)
- Laptop Intel Core I7 4ta generación con 2.40GHz, 8GB de RAM (Los dos Negamax)

Resultados

A continuación se presentan los resultados de la ejecución de los cuatro algoritmos sobre distintos estados que se presentan a lo largo de la variación principal de Othello. Las tablas tienen el mismo formato: La columna “npv” indica el número de estado de la variación principal del que se parte, la columna “Turno” indica turno a qué color de fichas le toca jugar, la de “valor” calcula el valor del tablero de acuerdo a los criterios estipulados en la sección de implementación, “nodos expandidos” indica los nodos a los cuales se les calcularon sus sucesores para continuar la búsqueda, “nodos generados” son todos aquellos que aparecieron en algún punto de la búsqueda, independientemente de si fueron podados o no, “tiempo” es el tiempo que tardó la ejecución en segundos y “generados/segundos” es la cantidad de nodos generados por segundo.

Nótese que si a un nodo solo se le calcula el valor correspondiente al estado y no se le calculan sus sucesores, cuenta como generado pero no como expandido, es por eso que el caso 34 (El juego terminado) tiene 0 nodos expandidos pero 1 generado. Adicionalmente, en scout se cuentan los nodos generados/expandidos por el procedimiento auxiliar test.

Negamax

npv	Turno	Valor	Nodos expandidos	Nodos generados	Tiempo (Segundos)	Generados/segundo
34	Blancas	-4	0	1	0.00	inf
33	Negras	-4	1	2	0.00	inf
32	Blancas	-4	3	5	0.00	inf
31	Negras	-4	4	6	0.00	inf
30	Blancas	-4	9	13	0.00	inf
29	Negras	-4	10	14	0.00	inf
28	Blancas	-4	64	91	0.00	inf
27	Negras	-4	125	177	0.00	inf
26	Blancas	-4	744	1049	0.00	inf
25	Negras	-4	3168	4498	0.00	inf

24	Blancas	-4	8597	11978	0.00	inf
23	Negras	-4	55127	76826	0.05	1.63E+10
22	Blancas	-4	308479	428402	0.25	1.71E+11
21	Negras	-4	2525249	3478735	1797.00	1.94E+11
20	Blancas	-4	9459570	13078933	7094.00	1.84E+11
19	Negras	-4	65121519	90647895	49859.00	1.82E+11
18	Blancas	-4	625084814	876269598	552328.00	1.59E+10

Tabla 1: Resultados para Negamax.

Nótese que en la columna correspondiente a la tasa de nodos generados por segundos se puede apreciar que algunos casos tienen valor infinito (Inf). Esto se debe a que el tiempo que tardó se aproximó tanto a 0, que para efectos del computador el resultado tiende a infinito.

Negamax (Con poda alfa-Beta)

npv	Turno	Valor	Nodos expandidos	Nodos generados	Tiempos (Segundos)	Generados/segundo
34	Blancas	-4	0	1	0.00	inf
33	Negras	-4	1	2	0.00	inf
32	Blancas	-4	3	5	0.00	inf
31	Negras	-4	4	6	0.00	inf
30	Blancas	-4	9	13	0.00	inf
29	Negras	-4	10	14	0.00	inf
28	Blancas	-4	41	63	0.00	inf
27	Negras	-4	72	112	0.00	inf
26	Blancas	-4	242	378	0.00	inf
25	Negras	-4	579	939	0.00	inf
24	Blancas	-4	820	1316	0.00	inf
23	Negras	-4	10417	15765	0.02	1.05E+09
22	Blancas	-4	13684	21667	0.00	inf
21	Negras	-4	37567	59799	0.03	1.93E+09
20	Blancas	-4	182043	293139	0.13	2.35E+11
19	Negras	-4	426115	707009	0.30	2.38E+10
18	Blancas	-4	4825531	7888678	3.44	2.29E+11
17	Negras	-4	7789838	13053386	5.53	2.36E+11
16	Blancas	-4	52946700	88310040	37.23	2.37E+11

15	Negras	-4	176309609	291752728	122.52	2.38E+11

Tabla 2: Resultados para negamax con alfa beta

Nótese que en la columna correspondiente a la tasa de nodos generados por segundos se puede apreciar que algunos casos tienen valor infinito (Inf). Esto se debe a que el tiempo que tardó se aproximó tanto a 0, que para efectos del computador el resultado tiende a infinito.

Scout

npv	Turno	Valor	Nodos expandidos	Nodos generados	Tiempo (Segundos)	Generados/segundo
34	Blancas	-4	0	1	1.00024E-06	999760
33	Negras	-4	1	2	9.99775E-07	2.00E+11
32	Blancas	-4	4	6	4.00003E-06	1.50E+11
31	Negras	-4	5	7	2.00002E-06	3.50E+11
30	Blancas	-4	10	14	4.99981E-06	2.80E+11
29	Negras	-4	11	15	4.99981E-06	3.00E+11
28	Blancas	-4	58	86	2.40002E-05	3.58E+11
27	Negras	-4	120	184	4.30001E-05	4.28E+11
26	Blancas	-4	284	444	9.90001E-05	4.48E+11
25	Negras	-4	483	793	4.15E-04	1.91E+11
24	Blancas	-4	695	1134	5.59E-04	2.03E+11
23	Negras	-4	4404	7199	2.91E-03	2.47E+11
22	Blancas	-4	15038	24713	8.44E-03	2.93E+11
21	Negras	-4	31175	51333	1.70E-02	3.01E+11
20	Blancas	-4	102232	171916	6.15E-02	2.80E+11
19	Negras	-4	451345	766867	2.55E-01	3.01E+10
18	Blancas	-4	3569095	6107035	1.98E+00	3.08E+11
17	Negras	-4	9968122	16977632	5.44E+00	3.12E+11
16	Blancas	-4	39677477	67439887	2.15E+01	3.14E+11
15	Negras	-4	120547310	204339503	6.46E+01	3.16E+11
14	Blancas	-4	2246547157	3798595138	1.20E+03	3.18E+11

Tabla 3: Resultados para scout

Negascout

npv	Turno	Valor	Nodos expandidos	Nodos generados	Tiempo (Segundos)	Generados/segundo
34	Blancas	-4	0	1	1.00001E-06	999992
33	Negras	-4	1	2	1.00001E-06	2.00E+11
32	Blancas	-4	4	6	3.00002E-06	2.00E+11
31	Negras	-4	5	7	2.00002E-06	3.50E+11
30	Blancas	-4	10	14	3.00002E-06	4.67E+11
29	Negras	-4	11	15	3.00002E-06	5.00E+11
28	Blancas	-4	58	86	1.50001E-05	5.73E+11
27	Negras	-4	120	184	3.29998e-05	5.58E+11
26	Blancas	-4	284	444	8.19997E-05	5.41E+11
25	Negras	-4	483	793	1.42E-04	5.58E+11
24	Blancas	-4	695	1134	5.63E-04	2.01E+11
23	Negras	-4	4382	7168	2.54E-03	2.82E+11
22	Blancas	-4	13838	22827	7.59E-03	3.01E+10
21	Negras	-4	25043	41520	1.36E-02	3.05E+11
20	Blancas	-4	96100	162103	5.37E-02	3.02E+11
19	Negras	-4	419503	717074	2.37E-01	3.02E+11
18	Blancas	-4	3215996	5513929	1.78E+00	3.10E+11
17	Negras	-4	8901812	15208392	4.84E+00	3.14E+11
16	Blancas	-4	37045642	63032052	2.00E+01	3.15E+11
15	Negras	-4	109301007	185632378	5.86E+01	3.17E+11
14	Blancas	-4	2194183932	3713452198	1.16E+03	3.19E+10

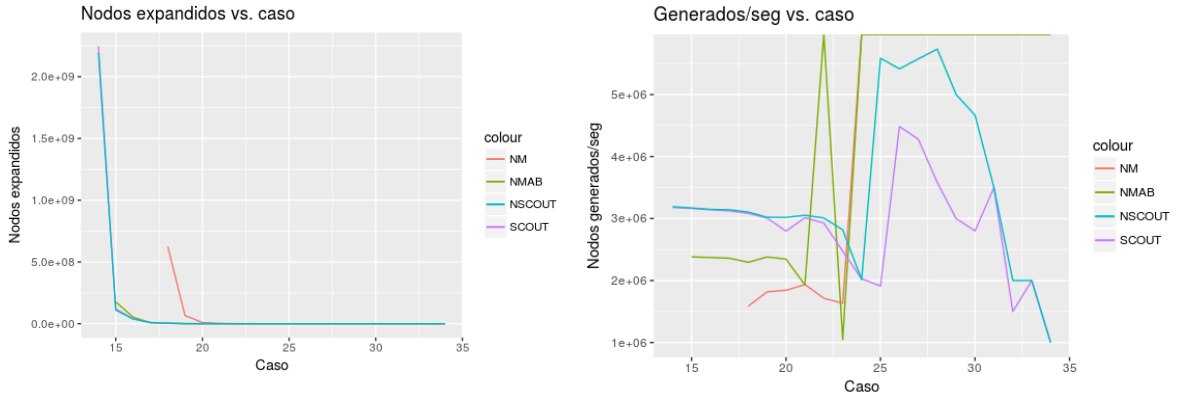
Tabla 4: Resultados para negascout

Análisis de resultados

En las siguientes páginas se presentan tablas que representan el comportamiento promedio de cada algoritmo. La tabla incluye el número de casos resueltos por cada algoritmo en el tiempo estipulado, los nodos expandidos en promedio, los nodos generales en promedio, el tiempo promedio y la tasa de nodos generados por segundo promedio. Adicionalmente, se incluyen gráficas que ilustran el comportamiento de cada uno en determinados aspectos.

Algoritmo	Casos resueltos	Nodos expandidos	Nodos generados	Tiempo (Segundos)	Generados/segundo
Negamax	17	41327499.00	57882248.41	35.9632	1755385.00
Negamax (con alfa-beta)	20	12127164.25	20105253.00	8.4594	2139162.50
Scout	21	115281667.90	194975424.24	61.3760	2857026.19
Negascout	21	112057567.90	189704396.52	59.4775	3484001.05

Tabla 5: Resultados promedio para cada algoritmo. En generados/segundos se omiten los valore "inf" para el cálculo del promedio



Gráficos 1a y 1b: Gráficos que ilustran el tiempo para cada caso en cada algoritmo y la tasa de nodos generados/seg

En primer lugar, se puede observar que la cantidad de nodos expandidos es inversamente proporcional al número que identifica al caso (Gráfico 1a). Esto se debe a la naturaleza de los casos utilizados: Como se explicó anteriormente, cada uno representa el estado del juego durante una partida denominada variación principal donde el 1 es el estado inicial y el 34 el estado final. Por lo tanto, un caso con un identificador pequeño tiene una menor cantidad de piezas en el tablero que otro con identificador mayor, y en consecuencia, hace falta un árbol más de mayor profundidad para alcanzar nodos terminales del juego. Luego, es normal que los casos con un número más pequeño sean los que mayor cantidad de nodos generados/expandidos y tiempo requieran. Solo se incluyen el gráfico de expandidos vs caso y no los de tiempo vs caso, o generados vs. caso porque tienen comportamientos parecidos

Nótese que en el gráfico 1b, cuando la tasa de generación de casos resulta infinita, esta se ve representada en el tope del cuadro que contiene las gráficas. Otro aspecto interesante, es el hecho de cómo la curva nodos generados por segundo empieza a presentar

oscilaciones menos pronunciadas, inclusive casi nulas, en los casos con menor número identificador, particularmente en el caso de negascout. Esto parece indicar que la proporción entre ambos valores tiende a un valor específico a medida que la profundidad crece.

También es importante destacar que la línea correspondiente Scout en el gráfico 1a se ve opacada por la de negascout debido a que en esta escala, los resultados son muy parecidos, lo cual causa que una línea se superponga a la otra. Este fenómeno se observa también en otras gráficas que se mostrarán más adelante.

A nivel individual, al observar el gráfico 1a, se hace obvio el hecho de que la cantidad de nodos expandidos de ejecución empieza a crecer de forma notable entre los casos 20 y 18 (Nota: También podría utilizarse el término “decrecer”, si se parte desde el 18, pero como en los experimentos y las tablas se parte del caso mayor al caso menor, pareció más conveniente usar el término “crecer) mientras que este mismo hecho se observa en los demás algoritmos entre los casos 14 y 16, lo cual evidencia una clara ventaja de los demás en una comparación individual caso por caso. Esto se debe a que el negamax simple no implementa ningún mecanismo de poda, más allá de establecer un límite de profundidad, cosa que todos los cuatro algoritmos hacen. Negamax con alfa-beta implementa el tipo de poda que indica su nombre, scout implementa el procedimiento test para comprobar la existencia (O inexistencia) de nodos con mejores valores, y negascout una mezcla de ambas.

Ahora, en términos generales, si se observa solamente la tabla 5, a simple vista puede parecer no solo que negamax con alfa-beta es mejor, sino que scout y negascout resultan bastante malos en comparación con negamax sin poda, tanto en tiempo como en nodos generados y expandidos. Sin embargo, al observar el gráfico 1a o, alternativamente, las tablas 1 a la 4, se notará que el caso 14, solo es resuelto por scout y negascout, requiere un tiempo muy superior al de todos los demás casos. Esto causa que el promedio de los últimos dos algoritmos se dispare. Sería más apropiado hacer la comparación utilizando solo los primeros 20 casos, tal como se hace en la siguiente tabla (Ver siguiente página).

Algoritmo	Casos resueltos	Nodos expandidos	Nodos generados	Tiempo (Segundos)	Generados/segundo
Negamax	17	41327499.00	57882248.41	35.9632	1755385.00
Negamax (con alfa-beta)	20	12127164.25	20105253.00	8.4594	2139162.50
Scout	20	8718393.45	14794438.55	4.6903	2840952.00
Negascout	20	7951249.70	13517006.45	4.2759	3498621.10

Tabla 6: Resultados promedio para cada algoritmo (Hasta 20 casos). En generados/segundos se omiten los valore "inf" para el cálculo del promedio

En este caso, se puede apreciar que Negascout es el algoritmo más efectivo en todos los aspectos medidos (Ya que utiliza tanto elementos de poda alfa-beta así como una versión del procedimiento Test de Scout), seguido de cerca por Scout y con una diferencia notable sobre Negamax alfa-beta y aún más sobre Negamax simple. Se puede apreciar nuevamente entonces, el hecho de que este algoritmo no posea ningún mecanismo de poda, lo pone en una situación clara de desventaja frente a los demás. De acuerdo al paper de Pearl (1980), el factor de ramificación de scout es mucho menor y más manejable ("Tractable") que el de alfa-beta en situaciones donde los valores terminales son discretos y se cumplen ciertas condiciones relacionadas con la distribución de probabilidad de los valores. Aparte de esto, el análisis empírico realizado por el autor en el mismo paper, sobre el juego Kalah, arroja resultados similares a los alcanzados en este paper, en los cuales alfa-beta parece tener mejor desempeño en casos con árboles de menor profundidad, pero a medida que la misma aumenta, el tiempo y los nodos generados/expandidos por Scout (y su variante negascout) crecen con menor rapidez lo que hace de esta familia de algoritmos más eficientes a mayores profundidades. Esto se puede evidenciar en el gráfico 2, donde para todos los casos anteriores al 20, que requieren de árboles de mayor profundidad que los mayores a 20, el crecimiento del tiempo requerido negamax con alfa-beta es muy superior al de los de la familia Scout (Y el de Scout es ligeramente superior al de Negascout).

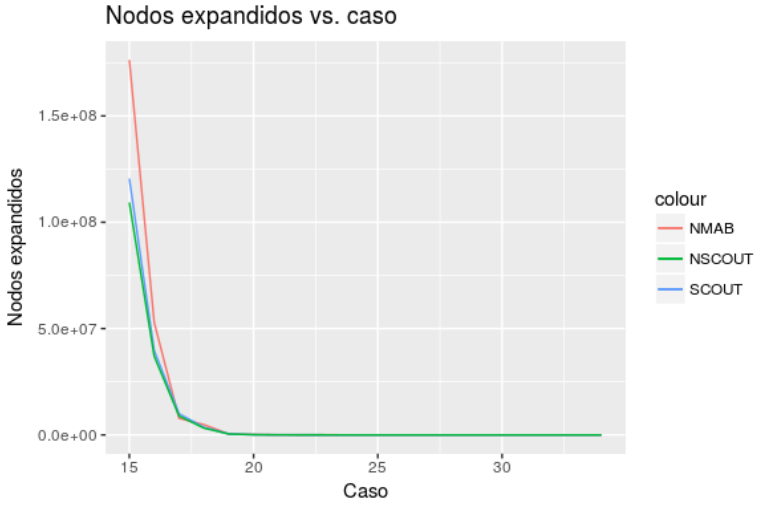


Gráfico 2: Nodos expandidos vs caso para Negamax con alfa-beta, Scout y Negascout, 20 casos (Se omite Negamax para permitir visualizar con mayor claridad el comportamiento)

Es importante resaltar que Pearl, 1980 señala que en muchos casos la superioridad de Scout sobre alfa-beta solo puede ser determinada por análisis empírico

Para terminar de mostrar la ventaja que ofrecen los algoritmos con poda sobre negamax sin ningún tipo de poda, se muestra la tabla que compara los resultados promedio solo con los primero 17 casos, es decir, los que resuelve Negamax

Algoritmo	Casos resueltos	Nodos expandidos	Nodos generados	Tiempo (Segundos)	Generados/Segundo
Negamax	17	41327499.00	57882248.41	35.9632	1755385.00
Negamax (con alpha-beta)	17	323361.06	528759.18	0.2298	2000032.00
Scout	17	245585.88	419514.65	0.1369	2787607.06
Negascout	17	222149.00	380429.82	0.1230	3559570.12

Tabla 7: Resultados promedio para cada algoritmo (Hasta 17 casos). En generados/segundos se omiten los valore “inf” para el cálculo del promedio

Conclusiones

- En algoritmos de búsqueda para árboles de juego, es necesario efectuar una forma de poda adicional al límite de profundidad para búsquedas eficientes
- Para el caso estudiado (Othello 6x6), los algoritmos de la familia Scout parecen arrojar mejores resultados en espacio y tiempo que Negamax, inclusive con poda alfa-beta
- Entre Negamax, Negamax con alfa-beta, Scout y Negascout, el último parece ser la mejor opción para efectuar las búsquedas
- En problemas exponenciales, si se va a comparar el promedio de los resultados obtenidos por distintos algoritmos, deben usarse los mismos casos para todos los algoritmos para evitar que algún caso tenga demasiada influencia en los promedios

Referencias bibliográficas

Pearl, J. (1980, August). SCOUT: A Simple Game-Searching Algorithm with Proven Optimal Properties. In *AAAI* (pp. 143-145).

Sakuta, M., & Iida, H. (2001). The performance of PN*, PDS and PN search on 6× 6 Othello and Tsume-Shogi. *Advances in Computer Games*, *9*, 203-222.

Anexos

En la carpeta “Results” del repositotrio https://github.com/djpg98/proyecto-2-ci5437 se encontrarán las tablas y gráficos (Algunos no incluidos en el informe)