Sistemas de Inteligencia Artificial

Métodos de Búsqueda No Informados e Informados

Trabajo Práctico Especial 1

Alumnos: Serber Valeria (51021), di Tada Teresa (53434) y Mattsson Björn (56557)
Fecha de Entrega: 31 de marzo de 2015
Número de Grupo: 11

1 - Introducción

Se utilizó el motor de inferencia entregado por la cátedra para implementar diferentes algoritmos informados y no informados que resuelvan el problema establecido para nuestro grupo, el juego 0hh1[1].

2 - Objetivos

Crear un Sistema de Producción que será usado para resolver el problema 0hh1, que se describe más adelante, utilizando un motor de inferencia entregado por la cátedra. Implementar estrategias de búsqueda informadas y no informadas para el problema. Presentar al menos dos heurísticas, y presentar las funciones de costo utilizadas. Analizar los resultados obtenidos para cada corrida del programa.

3 - El juego 0hh1

El juego asignado por la cátedra es un juego de satisfacción de restricciones que consiste en un tablero cuadrado con casilleros que pueden tener tres estados: vacío, rojo o azul. Se debe pintar los casilleros vacíos para lograr cumplir las siguientes tres reglas al llenar todo el tablero:

1. No debe haber más de dos celdas del mismo color adyacentes.
2. Las filas y columnas tienen la misma cantidad de celdas de un color y de otro.
3. No puede haber dos filas o dos columnas iguales.

4 - Algoritmos implementados

4.1 - Algoritmos no Informados

4.1.1 - BFS

Breadth First Search es un algoritmo para realizar una búsqueda por el ancho del árbol. Se comienza en el nodo raíz, se expande, luego se expanden todos sus sucesores, después los sucesores de

[1] www.0hh1.com

los sucesores y sigue de esta manera. Es decir, todos los nodos son expandidos en su nivel antes de que los nodos del siguiente nivel sean expandidos.

4.1.2 - DFS

Depth First Search es un algoritmo que realiza búsqueda por profundidad. Permite recorrer todos los nodos del árbol de manera ordenada. Comienza en un nodo raíz, genera un nodo adyacente, a su vez a éste nuevo nodo le genera un nodo hijo y así eligiendo un camino hasta llegar a un nodo hoja. Una vez que llega a éste tipo de nodos, hace backtracking.

4.1.3 - Profundidad Iterativa (IDDFS)

Este algoritmo consiste en hacer DFS con un límite de profundidad, fijarse en esa profundidad si se llegó a la solución, y sino, aumentar gradualmente el límite y volver a empezar desde el comienzo. Combina ventajas de DFS y BFS, entre las cuales se encuentra el uso de memoria de DFS, que es mucho menor al de BFS. Si el límite se aumenta gradualmente de a 1 nivel de profundidad, el árbol se termina recorriendo como un BFS, pero con el requerimiento de memoria de un DFS. Conviene utilizar este algoritmo cuando el espacio de búsqueda es muy grande y cuando no se sabe a qué nivel se encuentra la solución.

4.1.3.1 Implementación del algoritmo

En nuestra implementación de profundización iterativa, elegimos utilizar un aumento gradual de 10 niveles, para llegar más rápido a la solución, porque en nuestro problema podemos calcular en qué nivel se encuentra, y por lo tanto, todos los análisis que se hagan en niveles superiores en realidad no sirven.

4.2 - Algoritmos Informados

4.2.1 - Greedy BFS

Intenta expandir el nodo más cercano al estado final, considerando que sea el que más probablemente nos lleve a la solución rápida. Para esto, evalúa nodos usando solo la heurística.

4.2.2 - A* BFS

Evalúa los nodos con una función f(n) que combina la heurística con el costo para llegar al nodo actual. Intenta expandir el nodo con menor f de la frontera donde están todos los nodos expandidos pero no explotados.

5 - Implementación

5.1 - Representación del juego

El tablero se representó con una matriz de enteros, con un tamaño de 8x8 a pedido de la cátedra. Los colores azul y rojo con los que está diseñada la interfaz del juego se representaron con los números 1 y 2, mientras que el estado vacío se representó con un 0.

Las reglas del juego implican poner una ficha de alguno de los dos colores en algún casillero particular del tablero.

5.2 - Heurísticas Desarrolladas

5.2.1 - Heurísticas

5.2.1.1 - Heuristica: "Columns"

Esta heurística se fija la cantidad de celdas vacías de cada columna, y se queda con la de número máximo. Es decir calcula cuál es la máxima cantidad de celdas vacías que hay entre todas las columnas. Luego, los algoritmos que la utilicen se quedarán con aquellos nodos que tengan menor valor. En este caso, se quedan con los nodos que tienen espacios más reducidos para llenar por columna.

Esta heurística ***es admisible***, ya que el número de celdas vacías que hay en una columna, es siempre menor o igual a la cantidad total de celdas vacías del tablero, que es la cantidad de niveles del árbol que falta recorrer en profundidad para llegar a la solución, por lo que no sobreestima el valor real.

5.2.1.2 - Heurística "MinColor"

Como se sabe que cada tablero final tiene 32 rojos y 32 azules, esta heurística calcula la cantidad de celdas que falta pintar para cada color y se queda con el mayor valor. De modo que los algoritmos que utilizan esta heurística, se quedan con el tablero que tiene menor cantidad de celdas para pintar para un color particular.

Esta heurística ***es admisible***, ya que el valor es siempre menor o igual a la cantidad de rojos/azules que hay en el tablero final, osea menor o igual a 32. Este valor es menor a la cantidad de celdas totales que faltan pintar (es decir menor a la profundidad del tablero).

5.2.1.3 - Heurística "FullColor"

Calcula la cantidad de filas que todavía no tienen ni 4 azules ni 4 rojos. Es decir el rango de *h* va desde 0 hasta 8. Si se llega al goal, hay 0 filas que no tienen ni 4 azules ni 4 rojos, entonces *h* es igual a 0. Los algoritmos que la usen se quedarán con los tableros que tengan más filas con al menos un color lleno.

Esta heurística ***es admisible*** ya que *h* es siempre menor o igual a la cantidad de celdas vacías. Solamente *h* es igual a la cantidad de celdas vacías cuando la misma es 1.

5.2.2 - Diferencias y similitudes entre las heurísticas

1. Las tres heurísticas son admisibles.
2. La heurística "MinColor" es más cercana al costo real (la cantidad de celdas que falta pintar) que las heurísticas "FullColor" y "Columns". A su vez, "Columns" es por lo mismo mejor que "FullColor". Es decir si h1() es Columns, h2() es MinColor, h3() es FullColor() y h*() es el costo real, tenemos en general que h3(n)<=h1(n)<=h2(n)<=h*(n)
3. FullColor va a tener más cantidad de estados que den el mismo valor de heurística a pesar de ser diferentes.

5.3 - Funciones de costo

Al momento de elegir las funciones de costo para las reglas se observó que todas las reglas del juego aplicaban un color en un determinado casillero del tablero y esto no implicaba más costo en función del color o del casillero, por lo que se utilizó un costo unitario de valor uno.

7 - Análisis de los resultados

7.1 - Datos analizados

Los resultados pueden observarse en las tablas adjuntas en el anexo. En ella se detallan los siguientes parámetros para cada algoritmo y cada nivel del juego implementados. También puede verse una representación de los tableros utilizados en cada nivel.
La profundidad de la solución para nuestro problema equivale a la cantidad de celdas del tablero que se encuentran sin pintar, y es un número constante para el mismo tablero.
La cantidad total de estados generados, es la cantidad de estados que quedaron en open sin explotar más los que fueron explotados, que se encuentran en close.
El número de nodos frontera es la cantidad de estados que quedaron en open al finalizar el algoritmo, es decir los que quedaron sin explotar pero ya expandidos.
El número de nodos explotados es la cantidad de estados que quedaron en close al finalizar el algoritmo.

7.2 Análisis

Se analizaron los datos de los diferentes algoritmos y heurísticas con 4 tableros distintos de complejidad ascendente que pueden verse en el anexo del presente informe. El primer tablero tiene profundidad de solución 14, el segundo 20, el tercero 24 y el último 30.

Como se utilizó un orden de reglas no determinístico y esto genera que de manera aleatoria se ordenen las reglas de una manera más conveniente o menos conveniente, se corrió el primer tablero 3 veces por cada conjunto de características y los otros 10 veces, y a esto se le calculó el promedio de los datos que nos interesaban. Si en algún caso a partir de los diez minutos no se llegaba al tablero solución se dejó de correr el programa.

Al comparar BFS y DFS se puede notar que BFS expande todos sus estados generados por lo que al finalizar el algoritmo queda la frontera vacía, mientras que DFS expande los necesarios para llegar a la solución de acuerdo al camino que recorra por lo que su frontera no queda vacía constantemente. DFS en promedio expande y explota menos cantidad de nodos.

También se observa que BFS tarda más tiempo en llegar a la solución que DFS, lo que se debe en parte al ancho del árbol de nodos de nuestro problema (ancho<=64^(nivel de profundidad)).

En cuanto al algoritmo IDDFS, desde un principio sabíamos que no tenía sentido utilizarlo ya que la solución de nuestro problema se encuentra a una profundidad conocida. Si se implementaba el algoritmo de modo que aumente un nivel de profundidad en cada iteración, se hubieran analizado muchas veces niveles anteriores a la solución cuando en realidad no era necesario. En nuestra implementación elegimos utilizar un aumento gradual de 10 niveles para llegar más rápido a la solución.

Se puede observar que como la solución se encuentra en el último nivel del árbol, el algoritmo termina realizando un DFS completo en cada iteración. Esto puede verse en los resultados ya que ambos quedan con una cantidad de nodos frontera similar y además IDDFS explota casi el doble de nodos que explota DFS, valor que crece proporcional a la profundidad de la solución.

Por otro lado, si observamos la cantidad de nodos expandidos con el algoritmo Greedy, la heurística Columns logra resolver en el tiempo estipulado hasta el tablero de mayor dificultad, mientras que

las dos restantes no. Esto se debe a que a pesar de que MinColor es teóricamente mejor, pero genera que el algoritmo ponga el color que le conviene en cualquier lado aunque el casillero pueda no convenir, sólo para obtener un mejor valor heurístico. En cambio, Columns cuenta los casilleros vacíos de la columna menos llena, cantidad que va a mantenerse menos cambiante que los colores y va a obtener colores mejores ubicados.

Por otra parte, FullColor, al estar más lejos del costo real y tener poca variación debido al orden de reglas, logra resolver tablero de mayor dificultad pero sólo si obtiene las reglas en el orden correcto y explota un nodo por nivel.

A pesar de lograr resultados buenos con las heurísticas implementadas y el algoritmo Greedy, no se logra lo mismo con A estrella. Sólo se puede resolver el tablero más sencillo con una cantidad de nodos expandidos similar a los del algoritmo BFS pero con más tiempo de procesamiento que éste y con menos cantidad de nodos generados y explotados, y más nodos en la frontera.

Si comparamos ambos algoritmos informados la funcionalidad del algoritmo A estrella hace que este expanda muchos más nodos que Greedy ya que toma en cuenta todos los nodos ya expandidos y sin explotar al elegir el siguiente nodo a explotar. Además, los costos unitarios no mejoran el algoritmo y generan que se expanda en A estrella casi siempre el nodo que está en un nivel más arriba porque presenta un menor costo, y esto lo vuelve muy similar a un BFS, que claramente no es conveniente en cuestión de tiempos ni procesamiento.

Conclusiones

Greedy funcionó mucho mejor que los algoritmos no informados BFS, DFS y IDDFS, como era de esperarse al ser un algoritmo informado que puede elegir mejor que nodos explotar. Por su parte, A* como aumentó mucho los costos de procesamiento y los tiempos de ejecución. No sirvió su implementación por la estructura del juego, que tiene solución una profundidad conocida y reglas con costo uniforme.

No se pudo analizar el tema de las heurísticas admisibles y no admisibles en A* debido a que el algoritmo no era el indicado para el problema.

IDDFS por su parte tampoco sirvió para nuestro problema ya que se tiene una profundidad conocida, y se terminó procesando demasiado sin necesidad.

DFS logró llegar con menos procesamiento que BFS a la única solución de los tableros de prueba debido al ancho que tienen nuestro árbol de nodos.

Por otro lado, las heurísticas más cercanas al costo real fueron mejores para los algoritmos informados que las más lejanas, pero teniendo en cuenta que fueran óptimas para nuestro juego y su implementación.

Para cerrar, la solución más certera para este juego en particular hubiera sido un ordenamiento de reglas donde se colocara la regla del casillero donde solo puedo poner un color, pero a pedido de la cátedra no se realizó ningún ordenamiento ya que la idea del trabajo era concentrarse en los algoritmos, heurísticas y costos sin simplificar tanto el problema. Dentro de lo realizado, la mejor solución para resolver el problema fue con el algoritmo Greedy y la heurística admisible Columns.

Anexo

Resultados

BFS - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (s)
1	14	16384	0	16383	10.075912441
2	14	16384	0	16383	10.359412636
3	14	16384	0	16384	9.808675322
Promedio	**14**	**16384**	**0**	**16384**	**10.081333466**

DFS - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	9115	91	14	0.032977
2	14	9115	91	14	0.033912
3	14	9115	91	14	0.034856
Promedio	**14**	**9115**	**91**	**14**	**0.033915**

DFS - Board 2

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	20	207429301	207	429300	145.967614352
2	20	206245781	206	245780	75.209176642
3	20	20221	202	20	0.037630554
4	20	2455109	245	5108	1.193807395

5	20	242124465	242	124464	35.671961484
6	20	224513925	224	513924	169.382368172
7	20	206482325	206	482324	156.23477421
8	20	20021	200	20	0.035652344
9	20	22527797	225	27796	6.468815239
10	20	216245797	216	245796	76.037447503
Promedio	**20**	**132806474**	**217**	**207453**	**67**

DFS - Board 3

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	24	31025	310	24	0.065869
2	24	31225	312	24	0.056643
Promedio	**24**	**31125**	**311**	**24**	**0.061256**

IDDFS - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	9115	91	26	0.096976827
2	14	9115	91	26	0.105107071
3	14	9115	91	26	0.029161982
Promedio	**14**	**9115**	**91**	**26**	**0.07708196**

IDDFS - Board 2

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	20	213419253	213	419264	115.999452865

2	20	2095141	209	5152	1.613870236
3	20	204910773	204	910784	439.981107842
4	20	21421	214	32	0.046349441
5	20	20521	205	32	0.044538431
6	20	2091276085	209	1276096	515.976888124
7	20	21221	212	32	0.048140632
8	20	2091186101	209	1186112	466.601273117
9	20	202504821	202	504832	146.646830225
10	20	207180245	207	180256	52.832095559
Promedio	**20**	**501263558**	**208**	**448259**	**174**

Greedy - Heurística “Columns” - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	9115	91	14	0.024776
2	14	9115	91	14	0.024802
3	14	9115	91	14	0.02526
Promedio	**14**	**9115**	**91**	**14**	**0.024946**

Greedy - Heurística “Columns” - Board 2
Se muestra el promedio de haberlo corrido 10 veces

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
Promedio	**20**	**348881715**	**216**	**287304**	**126,3294897**

Greedy - Heurística “Columns” - Board 3
Se muestra el promedio de haberlo corrido 10 veces

	Profundidad de la solución	Estados generados	Número de nodos frontera	Número de nodos	Tiempo de procesamiento

		totales		explotados	(segundos)
Promedio	**24**	**421767949**	**317**	**228408**	**123,9761008**

Greedy - Heurística “Columns” - Board 4

Se muestra el promedio de haberlo corrido 10 veces

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
Promedio	**30**	**44453018**	**443**	**13137**	**7,9617328**

Greedy - Heurística “MinColor” - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	9115	91	14	0.027246
2	14	9115	91	14	0.024837
3	14	9115	91	14	0.024661
Promedio	**14**	**9115**	**91**	**14**	**0.02558133333**

Greedy - Heurística “MinColor” - Board 2

Se muestra el promedio de haberlo corrido 10 veces

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
Promedio	**20**	**65400299**	**219**	**124778**	**59,6049128**

Greedy - Heurística “MinColor” - Board 3

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	24	-	-	-	1800
2	24	30825	308	24	0.076568

3	24	3008217	300	8216	4.026213
4	24	-	-	-	600

Greedy - Heurística “FullColor” - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	9115	91	14	0.028159
2	14	9115	91	14	0.026927
3	14	9115	91	14	0.029411
Promedio	**14**	**9115**	**91**	**14**	**0.02816566667**

Greedy - Heurística “FullColor” - Board 2
Se muestra el promedio de haberlo corrido 10 veces

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
Promedio	**20**	**81043493**	**202**	**231322**	**111,607413**

Greedy - Heurística “FullColor” - Board 3
Se muestra el promedio de haberlo corrido 10 veces

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
Promedio	**24**	**34825367**	**313**	**25496**	**12,3172074**

Greedy - Heurística “FullColor” - Board 4

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	30	43931	439	30	0.079984

2	30	44031	440	30	0.088354
3	30	44031	440	30	0.080632
4	30	43931	439	30	0.076986
5	30	44131	441	30	0.086485
6	30	-	-	-	600

A* - Heurística “Columns” - Board 1

	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	2316361	23	16360	118.501168217
2	14	2316361	23	16360	115.660990734
3	14	2316361	23	16360	120.641056259
Promedio	**14**	**2316361**	**23**	**16360**	**118.2677384**

A* - Heurística “MinColor” - Board 1

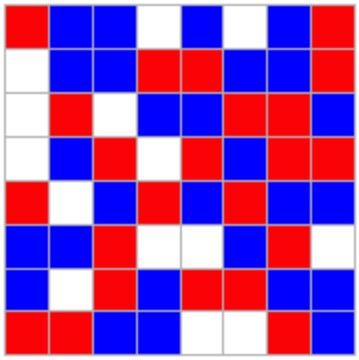
	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	40015984	400	15983	112.110110954
2	14	40015984	400	15983	113.251882848
3	14	40015984	400	15983	113.251882848
Promedio	**14**	**40015984**	**400**	**15983**	**112.87129222**

A* - Heurística “FullColor” - Board 1

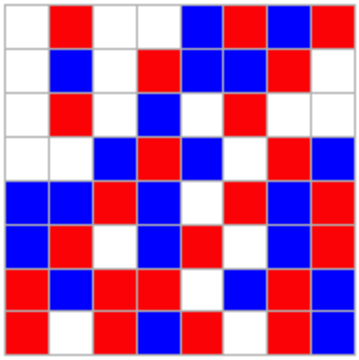
	Profundidad de la solución	Estados generados totales	Número de nodos frontera	Número de nodos explotados	Tiempo de procesamiento (segundos)
1	14	16384	0	16383	131.09096899

2	14	16384	0	16383	128.209036471
3	14	16384	0	16383	109.658382936
Promedio	**14**	**16384**	**0**	**16383**	**122.98612947**

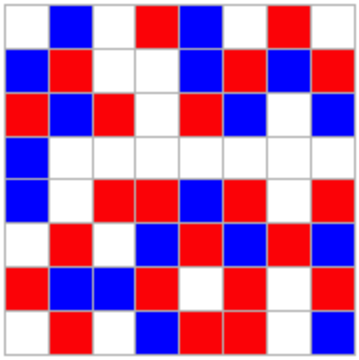
Tableros utilizados



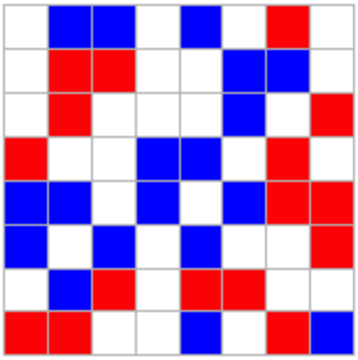
Board 1: Tiene 14 celdas vacías



Board 2: Tiene 20 celdas vacías



Board 3: Tiene 24 celdas vacías



Board 4: Tiene 30 celdas vacías