Trabajo Práctico Especial
Sistemas de Inteligencia Artificial

Segundo Fariña - 56176
Martin Victory - 56086
Sebastian Favaron - 57044
Ximena Zuberbuhler - 57287

Introducción

El objetivo del trabajo práctico es desarrollar un resolvedor general de problemas en Java, el cual toma un problema, una heurística y un método de búsqueda e intenta encontrar la solución con el menor tiempo y uso de memoria posible. Debe ser genérico y por lo tanto capaz de resolver cualquier problema que haya sido implementado siguiendo las interfaces del sistema.

Descripción del trabajo

El GPS (General Problem Solver) pedía que se implementaran una heurística (una estimación del costo restante a la solución), un problema (este debia ser capaz de generar un estado inicial, proveer las reglas del juego y determinar si un estado particular era la solución), reglas (cada una determinaba un movimiento válido en el juego, devolviendo el nuevo estado generado de aquel movimiento), y estados (descripcion completa del estado actual del juego, independiente de los estados anteriores y siguientes).

Estado:

Nuestro estado *SkyscraperState* contiene una clase *Board* que representa un tablero, el cual fue implementado como una matriz de bytes, y a su vez ofrece ciertos métodos para interactuar con este. *SkyscraperState* en sí es responsable de chequear la validez del tablero como solución o, en su defecto, la cantidad de errores en el estado (por ejemplo, observadores[1] que no tengan su restricción satisfecha, columnas con números repetidos)

Heurísticas:

Como primera heurística definimos SkyscraperHeuristicA. Ésta primer heurística no logró completar los tableros 5x5 antes de quedarse sin memoria en nuestras máquinas. Su fórmula es:

```
h = 0
cant = cantidadDeColumnasConValoresRepetidos
si cant es mayor a 1:
    cant = cant - 1
h += cant
si hay algún observador sin su restricción satisfecha:
```

[1] **Observadores:** números alrededor del tablero que indican cuántos edificios deben ser vistos desde aquel borde. Se dice que el número de cada observador es su restricción hacia el tablero.

h += 1

Para la segunda heurística, SkyscraperHeuristicB, la fórmula es:
h = cantDeColumnasConValoresRepetidos + cantDeObservadoresInsatisfechos

Esta heurística es admisible, ya que en el peor de los casos el valor de los casos es igual al costo. Como en el caso de que solo falte un swap para encontrar la solución, el costo sería 8 y como máximo se puede tener dos observadores mal de arriba, dos de abajo, uno de izquierda y uno de derecha (ya que los swaps son horizontales) y dos columnas con repetidos, esto suman 8.

Reglas:

Para reducir el problema decidimos muy temprano realizar swaps únicamente horizontales. De ésta manera nos ahorrabamos chequear si las filas tenían valores repetidos y solo nos restaba chequear las columnas y si los observadores estaban satisfechos. Como el set de reglas era completo (se podría generar cualquier configuración del tablero) decidimos que era una buena reducción.

Aun con ésta reducción, sin embargo, estábamos generando muchas reglas. Por ejemplo, en un tablero de *5 x 5*, se obtenían 50 reglas (10 por fila) y esto generaba a su vez un árbol demasiado ancho para la memoria con la que estábamos trabajando, y así es como toda estrategia terminaba con error de falta de memoria.

La segunda reducción fue sólo realizar swaps horizontales y adyacentes, bajando la cantidad. De esta manera en el tablero *5 x 5* de 50 reglas se redujo a 20. Esto resultó ser el cambio necesario para que el programa pudiera correr en memoria.

Problema:

Para la generación de un estado inicial, simplemente inicializabamos un estado nuevo con el tamaño del tablero *n*, y este se encargaba de generar un tablero con filas con valores del 1 a *n* desordenados.

Las *n x (n - 1)* reglas descritas anteriormente se crean junto con el problema al tener el tamaño del tablero.

La determinación de si un estado es solución es delegada al mismo estado que tiene un método para resolver la consulta.

Análisis de resultados

A continuación un promedio (sobre 10 corridas de cada variante) de los resultados aplicando cada estrategia de búsqueda sobre el juego “Skyscrapers” y ante distintos tamaños de tablero (3, 4 y 5)
Aclaración: se usó la heurística B ya que demostró ser superior a la A.

Estrategia	Tam	Profundidad	Costo total	Nodos visitados	Explosiones	Tiempo(ms)
BFS	3x3	4.2	33.6	133.4	95.8	2
	4x4	11.5	92	181735.5	146108.3	2499
DFS	3x3	30.66	245.28	177.92	106.24	0
	4x4	23774.4	190195.2	299773.1	160299.7	4806
IDDFS	3x3	4.7	37.6	89.2	320.3	4
	4x4	11.5	92	130091.1	893552	20301
GREEDY	3x3	9.1	72.8	40.4	9.1	3
	4x4	135.6	1084.8	1304	135.6	17
	5x5	42353.6	338828.8	704463	42353.6	9744
A*	3x3	4.6	36.8	111.2	72.1	5
	4x4	12	96	145511.8	105976.2	4797

Solo greedy pudo resolver el 5x5

Conclusión

Durante el trabajo tuvimos algunas complicaciones que fuimos solucionando. Uno de nuestros mayores obstáculos fue el buen uso de la memoria. Inicialmente, al tener demasiadas reglas, el programa se quedaba sin memoria muy fácilmente (en especial con las estrategias como BFS que generan siempre todo el árbol en memoria). Al encontrar una forma de reducir esa cantidad a más de la mitad, los nodos del árbol tenían menos nodos hijos y el árbol lograba así expandirse verticalmente.

Otra complicación que tuvimos fue encontrar heurísticas admisibles. Ya que cada swap podía afectar hasta 8 restricciones (los 6 observadores asociados a ambos y la unicidad de las dos columnas de las celdas swapeadas). Para poder facilitar la búsqueda de heurísticas, decidimos cambiar la función de costo de la regla, haciendo que cada swap cueste 8.

Además, los problemas mencionados anteriormente, nos trajeron complicaciones en la resolución de tableros 5x5 ya que la cantidad de tableros

posibles de un juego de esa magnitud es de 22,869 millones, por lo que solo la búsqueda Greedy fue capaz de resolverlo[2].

[2] Greedy resulta ser mucho mejor que A* en nuestro caso particular. Como el sistema de swaps permite que cualquier rama llegue a la solución, si lo que queremos es resolverlo rápido realmente no nos importa que la solución tenga un camino corto, y greedy al no considerar el costo (longitud del camino recorrido) puede concentrar sus esfuerzos en achicar la heurística (aproximación de la longitud al camino a la solución), sin preocuparse cuanto avanza para hacerlo.