INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Sokoban

Sistemas de inteligencia artificial

Grupo 9 - Azarola, Nardini, Sakuda, Vazquez

11 de Abril del 2016

Índice

Introducción

Sokoban es un juego de ingenio creado por Hiroyuki Imabayashi, en el cual, un personaje debe empujar cajas hasta determinadas posiciones de un tablero.

En este trabajo se implementaron algoritmos de búsquedas informadas y no informadas para encontrar la solución a distintos tableros del Sokoban.

Desarrollo

Modelado

Para el modelado del Sokoban, se realizaron implementaciones de las interfaces del GPS brindado por la cátedra.

- GPSEngine-SokobanEngine: esta implementación crea una PriorityQueue con el comparator correspondiente a cada método de búsqueda (DFS, BFS, IDDFS, ASTAR, GREEDY). Además de realizar esta implementación, se tuvo que realizar modificaciones sobre GPSEngine con el fin de poder llevar a cabo el método de búsqueda IDDFS. Esto se debe a que con un comparador no pudimos simular el mismo.
- GPSState-SokobanState: inicialmente se había diseñado la clase para que cada estado estuviese compuesto por el tablero, una matriz de Celdas, y la posición del jugador. Al mismo tiempo el cálculo de la heurística se realizaba cada vez que se accedía al método “getH()”, es decir que se recalculaba la heurística de varios nodos pertenecientes a la cola, cada vez que se intentaba agregar un nodo.

  Debido a la falta de memoria durante las pruebas, se resolvió simplificar la cantidad de información almacenada en cada estado. Por tal motivo se crearon dos clases más que complementan al SokobanState para realizar las operaciones que originalmente ejecutar. Ahora el estado almacena únicamente la lista de cajas del tablero en cuestión y la posición del jugador mientras el resto de la información, que es estática como es el caso de las paredes y los goals, se encuentra disponible en StaticBoard. Además, se modificó para que la heurística sea calculada únicamente en el constructor de la clase y su valor quede almacenado.

  Para poder realizar el procesamiento, se creó una clase nueva: SokobanExpandedState. Esta clase recrea el tablero completo a partir del SokobanState y el StaticBoard. El mismo tiene un método “move” que es llamado desde el SokobanSatate y luego tiene otro método getSokobanState que retorna un nuevo estado con el cálculo de la heurísitca correspondiente.

Acciones posibles

Vienen representado por SokobanRule(GPSRule). Esta clase define las 4 reglas posibles dentro del juego: moverse arriba, moverse a abajo, moverse derecha, moverse izquierda. Además, determina que el costo de aplicar cada una de las reglas es 1

Modelo de transición

Retorna la nueva posición del jugador y el set de las posiciones de las cajas donde alguna de ellas pudo haberse visto modificada.

Condición de terminación

La condición de terminación del Sokoban se obtiene una vez que cada una de las cajas se encuentra sobre un goal.

Heurísticas

Heurística 1:

Se partió de la distancia Manhattan y se le fueron realizando mejoras específicas para el problema del Sokoban.

La distancia Manhattan se calculó entre las cajas y los goals, considerando las paredes intermedias entre los mismos. Este cálculo no considera que las cajas vayan a goals diferentes, sino al más próximo.

O sea, se calcula la distancia Manhattan entre cajas y goals al goal mas cercano, y esos valores se suman,

Al probarla se observó que el jugador en varias oportunidades se movía por el tablero sin entrar en contacto con las cajas. Para solucionar esto, se le adiciona a este valor la mitad de la distancia Manhattan del jugador a la caja.

Se utilizó la mitad de esta distancia porque se descubrió experimentalmente que al utilizar el valor de la distancia Manhattan, para muchos tableros el mover la caja

wall		
wall	box	
wall	wall	wall

más cercana al goal más cercano no siempre era la mejor opción. De este modo disminuyó el peso para elegir el movimiento.
En algunos casos, el mover una caja a una posición, puede generar que el algoritmo de búsqueda llegue a un estado desde el cual ya no es posible llegar a una solución.
A modo de ejemplo, si se posiciona una caja de la siguiente manera,

el jugador puede continuar moviéndose por el tablero pero nunca encontrar una solución, ya que la caja se encuentra bloqueada.
La solución a esto fue asignarle un valor heurístico infinito (MAX_VAL_INT/2). De esta forma, si el tablero tiene solución, se espera que el valor heurístico del estado no alcance el valor infinito.
Todos los tableros a analizar con más posibilidades de solución quedan en el tope de la cola. Con esto, se favorece el análisis de estados con más posibilidades de encontrar una solución y se pospone el análisis estados que es sabido que no llevan a solución alguna.
Los casos analizados para esto fueron aquellos donde se llega a tableros con configuraciones en donde una caja que no está en un goal y se encuentra en un cuadrado de 2x2 completado por paredes o cajas.

Heurística 2:

Se encontró que la heurística anterior, presentaba deficiencias en determinados tableros, y como consecuencia surgió la siguiente heurística.
A modo de ejemplo: Si se tenía un tablero con 3 cajas y 3 goals, y con solución. Si todas las cajas y un goal se encontraban posicionadas de manera cercana no bloqueante y las otras 2 cajas se encontraban en un extremo alejado del resto. La heurística 1, hacia el cálculo de distancia caja-goal utilizando el goal cercano a todas las cajas, y es sabido que todas las cajas no podrán ubicarse en el mismo goal.
Por este motivo, se buscó una solución que llevará a cada caja a un goal distinto, pero a su vez buscando que se tratará de una heurística admisible.
Se decidió lo siguiente, asociar una caja a un goal pero eso no podía ser de una forma azarosa. Asociamos a cada caja un goal, calculamos las distancias de caja Bi a goal Gi, y las sumamos. Si hacemos todas las combinaciones: cajas-goals, y obtenemos los resultados de esas sumas.
El menor de todos estos valores, es una aproximación admisible. El mismo sugiere la suma mínima de distancias caja a goals asociando siempre una caja a un goal distinto.

Encontrar que caja debemos asignar a cada goal tal que la distancia acumulada sea mínima, es un problema que el algoritmo húngaro inventado por Harold W. Kuhn en 1955, puede resolver.(ver https://es.wikipedia.org/wiki/Algoritmo_h%C3%BAngaro)
Este valor por ser mínimo garantiza que es la heurística es admisible, ya que el tablero asumimos que tenía solución.
En el algoritmo, debe pensarse el problema como un grafo bipartito, donde las particiones son por un lado las cajas y por otro los goals. Para esto se confecciona una matriz donde las filas representan las cajas y las columnas a los goals, donde cada elemento (i,j) representa la distancia caja-goal correspondiente. Mediante operaciones sobre la matriz, el algoritmo encuentra la distancia mínima acumulada.
La complejidad del algoritmo húngaro depende de cantidad N (cajas o goals), lo que hace que para grandes cantidades de cajas o goals el procesamiento del valor heurístico para un estado dependa de la cantidad total de cajas o goals.
Esta heurística no tiene en cuenta la posición del jugador, solo el posicionamiento de las cajas y los goals.

Costo

El movimiento del jugador es a una casilla, por lo cual el costo en todos los casos es de 1.

Análisis de resultados

A partir del desarrollo las pruebas realizadas, cuyos resultados se pueden observar en el Anexo, se llegó a las siguientes conclusiones:

1. Dentro de los métodos no informados, ninguno es claramente mejor que otro. La performance de cada método de búsqueda depende de la naturaleza del tablero con la que se está trabajando y se hallaron tableros específicos donde la performance de un algoritmo fue mejor que la del resto.
2. Dentro de los métodos informados, en general, la búsqueda por Greedy resultó ser mucho más eficiente que el A*. Sin embargo, esta eficiencia no garantiza la solución óptima.
3. Dentro de las heurísticas desarrolladas durante el trabajo, se concluyó que en general la heurística 1 es mejor, a pesar que la heurística 2 en varias situaciones necesita expandir menos nodos el tiempo de procesamiento que conlleva no la

hacer más útil que la primera. La heurística 2 es muy pura y está basada en un única variable lo que hace para determinados tableros la heurística 1 tenga más información en lo que representa su valor.

4. Ambas heurísticas dan generalmente un valor muy bajo en relación al costo real, dado que no tienen en cuenta muchas de las variables del tablero, como ser la distancia que tiene que recorrer el jugador para mover una caja o “marcar” tableros que no tienen solución.

Anexo

Tablas de resultados

Board 1			
	DFS	BFS	IDDFS
Analizados	43	48	83
Explotados	19	21	59
Costo	7	5	5
Tiempo	78ms	70ms	45ms
Frontera	3	9	2

Tabla 1: resultados del Board 1 de los métodos no informados

Board 1				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	18	35	16	31
Explotados	8	15	7	13
Costo	5	5	5	5
Tiempo	54ms	73ms	66ms	62ms
Frontera	3	6	3	6

Tabla 2: resultados del Board 1 de los métodos informados

Board 2			
	DFS	BFS	IDDFS
Analizados	nd	34935543	62741202
Explotados	396500000	12145617	26621626
Costo	nd	40	40
Tiempo	4hs 57min	37min 38s 242ms	10min 20s 384ms
Frontera	nd	1056866	34

Tabla 3: resultados del Board 2 de los métodos no informados

Board 2				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	8807	743493	619	1251277
Explotados	2856	249720	197	416593
Costo	49	40	45	40
Tiempo	279ms	16s 234ms	211ms	26s 510ms
Frontera	1931	62442	125	140187

Tabla 4: resultados del Board 2 de los métodos informados

Board 4			
	DFS	BFS	IDDFS
Analizados	6324983	2659411	85117995
Explotados	2528075	1037019	36191904
Costo	184	88	88
Tiempo	3min 52s 826ms	2min 27s 448ms	21min 53s 886ms
Frontera	89	9585	36

Tabla 5: resultados del Board 4 de los métodos no informados

Board 4				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	26505	78231	543523	680280
Explotados	10032	30304	210568	268441
Costo	98	88	108	88
Tiempo	351ms	3s 564ms	14s 114ms	25s 809ms
Frontera	1524	2799	2504	2406

Tabla 6: resultados del Board 4 de los métodos informados

Board 5			
	DFS	BFS	IDDFS
Analizados	1513284	9755	3997
Explotados	527774	3312	2697
Costo	13	11	11
Tiempo	13s 501ms	127ms	109ms
Frontera	14	2626	18

Tabla 7: resultados del Board 5 de los métodos no informados

Board 5				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	108	451	45	834
Explotados	36	151	14	284
Costo	17	11	11	11
Tiempo	67ms	63ms	64ms	74ms
Frontera	43	137	20	203

Tabla 8: resultados del Board 5 de los métodos informados

Board 6			
	DFS	BFS	IDDFS
Analizados	16925	55200	195640
Explotados	5511	18928	82614
Costo	103	27	27
Tiempo	307ms	561ms	1s 229ms
Frontera	97	1833	33

Tabla 9: resultados del Board 6 de los métodos no informados

Board 6				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	151	1243	571	3290
Explotados	46	415	185	1082
Costo	27	27	33	27
Tiempo	90ms	177ms	74ms	110ms
Frontera	78	283	75	550

Tabla 10: resultados del Board 6 de los métodos informados

Board 7			
	DFS	BFS	IDDFS
Analizados	212361	5514708	1120613
Explotados	67875	1918555	426815
Costo	256	40	40
Tiempo	1s 245ms	23s 872ms	4s 699ms
Frontera	200	28820	29

Tabla 11: resultados del Board 7 de los métodos no informados

Board 7				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	177	8088	272	28994
Explotados	53	2598	84	9369
Costo	40	40	42	40
Tiempo	98ms	378ms	62ms	469ms
Frontera	94	1058	105	3343

Tabla 12: resultados del Board 7 de los métodos informados

Board 9			
	DFS	BFS	IDDFS
Analizados	407	107606	8216
Explotados	109	29856	3301
Costo	108	16	16
Tiempo	64ms	963ms	170ms
Frontera	107	34677	24

Tabla 13: resultados del Board 9 de los métodos no informados

Board 9				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	108	862	457	22505
Explotados	27	237	124	6120
Costo	18	16	60	16
Tiempo	86ms	164ms	84ms	419ms
Frontera	54	50	111	4204

Tabla 14: resultados del Board 9 de los métodos informados

Board 10			
	DFS	BFS	IDDFS
Analizados	6393987	258964	298216
Explotados	2082996	84285	142081
Costo	205	21	21
Tiempo	1min 24s 681ms	4s 717ms	4s 93ms
Frontera	128	18194	20

Tabla 15: resultados del Board 10 de los métodos no informados

Board 10				
	Greedy	A*	Greedy	A*
	Heurística 1		Heurística 2	
Analizados	9058	18914	221	22269
Explotados	2889	6073	68	7320
Costo	29	21	23	21
Tiempo	369ms	638ms	69ms	317ms
Frontera	1590	2502	40	3321

Tabla 16: resultados del Board 10 de los métodos informados