Sistemas de Inteligencia Artificial

Trabajo Práctico Especial N°1

Métodos de búsqueda no informados e informados

Integrantes:

- Alejandro Bezdjian, 52108
- Horacio Miguel Gómez, 50825
- Juan Pablo Orsay, 49373
- Sebastián Maio, 50386

Tabla de contenidos

Introducción

El objetivo de este trabajo práctico es implementar los algoritmos de búsqueda no informados *Breadth first, Depth first e Iterative deepening depth first search,* así como también los algoritmos de búsqueda informados *Greedy search* y *A*.*

Dichos algoritmos deben ser utilizados para resolver el juego *Gridlock,* donde el objetivo es minimizar la cantidad de movimientos.

Implementación

Para la implementación, tal como lo pidió la cátedra, se separó la lógica del problema del motor de inferencia.

Motor de inferencia

Al motor de inferencia dado por la cátedra se le realizaron unas pequeñas modificaciones siempre manteniendo las interfaces provistas.

La clase GPSEngine se convirtió en una clase abstracta, y se crearon 5 clases concretas (una para cada estrategia). Cada una de estas clases hace un override del método explode() y cambia el tipo de estructura de datos utilizada para los nodos frontera, según le sea conveniente. Por ejemplo, las clases AStarEngine y GreedyEngine instancian una PriorityQueue, cada una con diferentes comparators mientras que BFS y DFS utilizan una Queue cambiando el orden de inserción de los nodos.

Modelado del problema

Para modelar el tablero se utilizó una matriz de enteros donde cada elemento en el mismo es representado con distintos números. Los tableros con los que se ejecutó el problema (y los que se encuentran en la gran mayoría de implementaciones online) es de 6x6, sin embargo nuestro modelado admite tableros de cualquier tamaño y forma. Se decidió agregarle un borde al tablero que representaban las paredes, esto se debe a que permite realizar las iteraciones sobre el tablero sin necesidad de realizar chequeos de condición de borde, lo que hace que el código sea más legible.

Se creó la clase Chip que representa las fichas en el tablero y brinda información útil como por ejemplo si la ficha es de tipo vertical u horizontal.

Finalmente se creó la clase Board que contiene la matriz y una lista de las fichas.

Lógica del problema

Estado

El estado del juego es representado por la clase Board que contiene la matriz del juego y la lista de fichas en el tablero.

Reglas

Las reglas son representadas por la clase GridlockRule que implementa la interfaz GPSRule. Las reglas utilizadas son:

- Mover cada ficha a la derecha, a la izquierda, hacia arriba y hacia abajo.

La clase GridlockProblem (que extiende de GPSProblem) tiene un método llamado calculateRules() que devuelve una lista de tipo GPSRule que contiene las reglas. Para un tablero con N cantidad de fichas se devuelven N*4 reglas. Los movimientos inválidos (fichas horizontales se mueven arriba o abajo, fichas verticales se mueven a la izquierda o derecha o fichas que se encuentran contra una pared se mueve hacia la pared) son descartados por el motor de inferencia ya que no son aplicables.

Condición de “goal”

Para este trabajo, decid
mos que un estado se toma como goal si la ficha principal del juego no tiene ninguna ficha que le bloquee la salida. La otra alternativa sería esperar a que la ficha principal esté en el borde del tablero junto a la salida.

Esta elección tiene algunas consecuencias. Una de ellas es que que el resultado final que arroja el algoritmo dice que lo resolvió en N pasos, pero en realidad la solución del tablero es N + M, donde M es la distancia de la ficha principal a la salida. La otra consecuencia es que el algoritmo termina más rápido, debido a que tiene que expandir M niveles menos el grafo de búsqueda.

Costo

Para este problema, se tomó el costo como una constante 1. Ésto se debe a que lo que nos interesa es resolver el problema en la menor cantidad de pasos. Debido al costo elegido, sabemos que el algoritmo BFS debería arrojar una solución óptima. Lo que buscaremos es que el algoritmo A* obtenga un resultado óptimo con al menos una heurística admisible.

Heurísticas

Admisibles:

- La primer heurística que se probó es la más “naive”. Esta es casi trivial y corresponde a contar cuántas fichas tiene la ficha principal entre ella y la salida.
- La segunda heurística es una extensión de la anterior y lo que hace es, además de contar las fichas que bloquean a la principal, cuenta cuántos movimientos tendrían que realizar las fichas obstructoras para despejar la salida.
- La tercer heurística es una extensión de la anterior, que además de contar cuantos movimientos las fichas que bloquean deberán realizar para despejar el camino, se fija si en la posición final en la que debería estar la ficha obstructora, para no obstruir la salida, cuantas otras fichas se solaparon con ella y las suma.

No admisibles:

- La primer heurística no admisible cuenta cuantos espacios ocupados hay en la parte superior del tablero, donde la parte superior se considera desde la línea de la ficha principal hasta la pared más cercana. Ésta heurística nos parece que tiene sentido debido a que un algoritmo GREEDY buscará elegir tableros que tengan menos ocupadas las posiciones superiores, y la ficha principal necesita espacios libres para moverse.
- La segunda heurística es el complemento de la anterior, se cuentan la cantidad de espacios vacíos en la parte superior del tablero. Esta heurística se eligió simplemente porque es la complementaria a la anterior, y porque quizás aunque parezca mala pueda lograr buenos resultados.

Algoritmos

- BFS: como se mencionó anteriormente, debido al costo tomado, se espera que encuentre al menos una solución óptima ya que lo que se intenta optimizar es la cantidad de movimientos, que coincidirá con la profundidad del árbol de búsqueda.
- DFS: este algoritmo es fuertemente dependiente del tablero y del orden de aplicación de reglas. Como se puede ver en el anexo de resultados, se probó la versión clásica de DFS (tomando un orden de reglas arbitrario y constante) y una versión donde el orden de la aplicación de las reglas es azaroso.
- IDDFS: para este algoritmo se probaron 3 variantes distintas. La primera, la versión clásica y menos eficiente, que por cada iteración que incrementa el nivel descarta toda la información

obtenida en la iteración (nodos visitados y mejores pesos) y vuelve a comenzar con un nivel de profundidad mayor. La segunda versión es una optimizada, en la que la siguiente iteración comienza con los nodos frontera de la iteración anterior (aquellos nodos cuyo costo iguala al costo máximo definido para la iteración actual). Por último, se probó aumentar el nivel en cada iteración por 5 y por 10 en lugar de 1, de este se espera que sea más eficiente en memoria y CPU pero abre la posibilidad de encontrar una solución no óptima.

- GREEDY: este algoritmo tomará el menor valor de H en la frontera para expandir.
- A*: se espera que con las heurísticas admisibles descriptas, el algoritmo sea capaz de encontrar una solución óptima. Además se espera que expanda una menor cantidad de nodos respecto a BFS y IDDFS ya que realiza una poda. Se intentará develar si el costo computacional de las heurísticas encontradas es menor que explorar una mayor cantidad de nodos (ya que lo ideal sería encontrar un punto óptimo en el cual la heurística sea rápida de computar, realice una poda y en definitiva sea más eficiente que BFS).

Conclusiones

La ejecución de los algoritmos con nuestro modelado y lógica (que se pueden ver en el anexo de este mismo informe) arrojó lo que se esperaba y concuerda con la teoría:

- BFS encuentra una solución óptima pero notamos que tiene un uso de memoria considerable.
- Las heurísticas mencionadas logran realizar una poda de una gran cantidad de nodos. Aunque utilizan muchos recursos para ser calculadas que repercute en el tiempo total de ejecución.
- Un resultado no esperado fue que la primer heurística no admisible resultó ser mejor de lo que se esperaba. En eficiencia de memoria y CPU es mejor que las admisibles y además logra resultados muy cercanos a los óptimos tanto para A* y GREEDY.
- Utilizar el GPS engine dio muy buenos resultados ya que permitió abstraer el problema de una forma muy eficiente.
- El algoritmo IDDFS sin optimizaciones resulta muy poco performante. En cambio la versión optimizada es una muy buena alternativa a BFS y A*. A su vez, existe la posibilidad de llevar la optimización un paso más adelante haciendo que la variación de niveles entre iteraciones sea mayor a 1 logrando mejorar el tiempo de ejecución y en algunos casos utilización de memoria pero se corre el riesgo de encontrar una solución no óptima.

Anexo

Resultados

Para las pruebas, se corrieron en una PC con CPU Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz con 6GB RAM y sistema operativo Ubuntu 16.04 LTS. Dichas pruebas se corrieron dentro del IDE Intellij Idea siendo esta la única aplicación abierta al momento de las pruebas (no tomamos en cuenta los procesos en ejecución del sistema operativo ya que son los mismos para todas las corridas).

El código utilizado para correr las pruebas se encuentra en el archivo Benchmark.java. Se ejecutaron 10 corridas para cada estrategia y tablero y luego se promedió el tiempo total por la cantidad de corridas. En esta sección se dejó que los métodos de búsqueda informados elijan la mejor heurística para cada caso, en la siguiente sección se compararán las heurísticas. Previo a la prueba, se ejecuta una ronda de *warm up* donde se soluciona un tablero trivial una gran cantidad de veces con la estrategia elegida para la prueba, esto permite que la JVM realice las optimizaciones que crea necesarias y que los tiempos obtenidos luego tengan una mejor confiabilidad.

Los tableros utilizados para las pruebas fueron tomados del juego *Unblock Me*, que es una implementación del juego para Android. Los nombres de los tableros corresponden al nivel de dificultad en dicho juego y son:

Nombre	Gráfico
371	

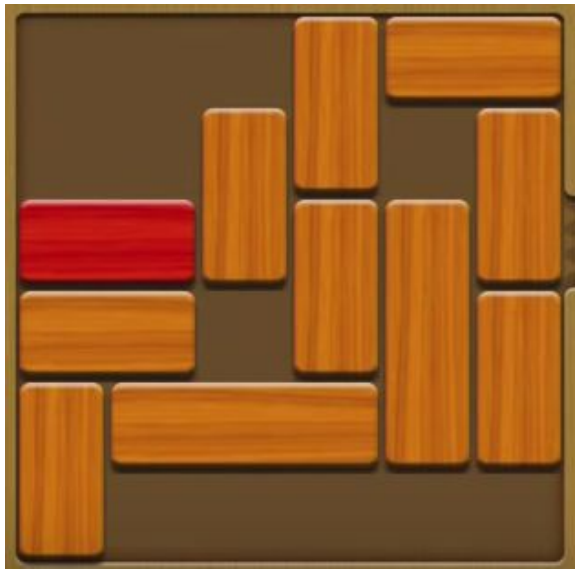
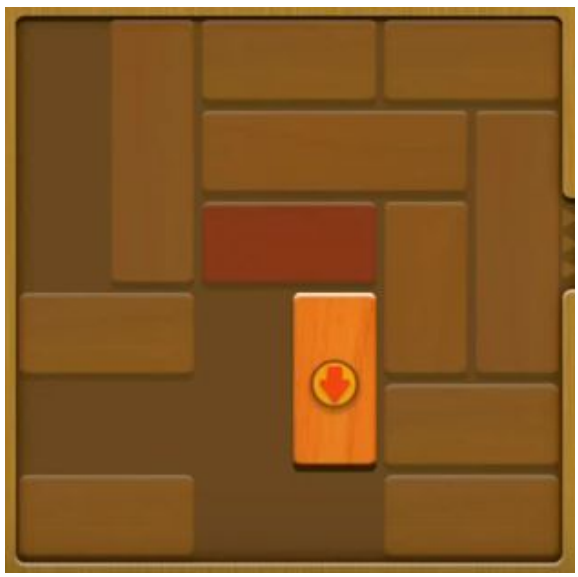
700	
800	
1200	

Tabla de resultados obtenidos

Tablero	Estrategia	Tiempo promedio [s]	Pasos hasta la solución	Nodos explotados	((node/BFSn) - 1) * 100	Tiempo respecto a BFS (t/BFSt)
371	BFS	0,4	52	9854	-	-
371	DFS	0,5	6807	11478	16,48%	1,25
371	IDDFS	37,29	52	1212823	12207,93%	93,23
371	Opt IDDFS	1,64	52	9375	-4,86%	4,10
371	GREEDY	9,22	74	33346	238,40%	23,05
371	ASTAR	4,01	52	7209	-26,84%	10,03
700	BFS	0,32	53	8971	-	-
700	DFS	55,05	5794	1567987	17378,40%	172,03
700	IDDFS	70,53	53	2009197	22296,58%	220,41
700	Opt IDDFS	1,43	53	8574	-4,43%	4,47
700	GREEDY	16,34	71	39158	336,50%	51,06
700	ASTAR	5,82	53	6985	-22,14%	18,19
800	BFS	0,1	51	3427	-	-
800	DFS	1,34	2322	46807	1265,83%	13,40
800	IDDFS	27,4	51	899333	26142,57%	274,00
800	Opt IDDFS	0,41	51	3339	-2,57%	4,10
800	GREEDY	4,62	53	12313	259,29%	46,20
800	ASTAR	1,83	51	2953	-13,83%	18,30
1200	BFS	0,03	35	1018	-	-
1200	DFS	0,12	486	4295	321,91%	4,00
1200	IDDFS	1,44	35	51246	4933,99%	48,00
1200	Opt IDDFS	0,09	35	989	-2,85%	3,00
1200	GREEDY	0,58	80	3935	286,54%	19,33
1200	ASTAR	0,33	35	875	-14,05%	11,00

Una de las primeras observaciones que podemos notar es que la “dificultad” de los niveles según el creador del juego del cual obtuvimos los tableros no es representativo a la cantidad de operaciones necesarias para resolverlo, ya que para el tablero de dificultad 1200 se requieren 35 pasos, mientras que para el del 700 se requieren 53 pasos.

Debido a que para este problema en particular la solución óptima es minimizar la cantidad de pasos requeridos para resolver el tablero, sabemos que BFS encontrará al menos una de las soluciones óptimas. Esto nos sirvió, en parte, para verificar que ASTAR estaba encontrando una

solución óptima y por lo tanto verificar que las heurísticas que elegimos para dicho algoritmo son admisibles.

De los resultados obtenidos, también podemos observar que el algoritmo GREEDY siempre fue más performante que DFS, tanto en tiempo, memoria y solución encontrada.

Por último, lo que podemos notar es que para el problema en cuestión y las heurísticas encontradas, ASTAR incluso podando una buena cantidad de nodos es varias veces más lento que BFS.

Comparación de heurísticas para A*

A continuación ampliaremos sobre los resultados obtenidos por cada heurística. Para este análisis solamente utilizaremos una heurística en cada corrida. Las heurísticas son las descritas previamente en el informe con más detalle. En el análisis anterior el algoritmo ASTAR tuvo la posibilidad de elegir el mejor valor de H() para cada estado. Como se vió en clase, lo que se espera es que en dicho análisis el algoritmo realice una mejor poda que en los casos individuales que se mostrarán a continuación.

Tablero	Heurística	Pasos hasta la solución	Nodos explotados	Nodos explotados en análisis previo	Diferencia de nodos explotados
371	1	52	8324	7209	15,47%
700	1	53	7900	6985	13,10%
800	1	51	3201	2953	8,40%
1200	1	35	951	875	8,69%
371	2	52	7209	7209	0,00%
700	2	53	6985	6985	0,00%
800	2	51	2953	2953	0,00%
1200	2	35	875	875	0,00%
371	3	52	7233	7209	0,33%
700	3	53	6985	6985	0,00%
800	3	51	2953	2953	0,00%
1200	3	35	875	875	0,00%

Con estos resultados podemos ver que la segunda heurística, para estos valores, es la más cercana a H*, esto lo deducimos del hecho que es la heurística que menos nodos explotó para llegar a la solución.

Selección de reglas al azar en DFS

Una de las cosas que notamos es que el algoritmo DFS tomaba caminos malos debido al orden de las reglas. Para este juego no hay, en principio, una ventaja por el orden de aplicación de las reglas, por eso decidimos probar el algoritmo con una selección al azar y observar los resultados. Los datos promedio fueron tomados sobre 100 corridas en la misma PC mencionada anteriormente.

Tablero	Pasos a la solución promedio	Pasos a la solución en DFS	Nodos explotados promedio	Nodos explotados por DFS
371	3837	6807	932016	11478
700	4092	5794	803710	1567987
800	1790	2322	137384	46807
1200	471	486	1420	4295

Como se puede ver, utilizando una elección al azar de las reglas, en promedio se logra bajar los pasos a la solución aunque en algunos casos se ve como la cantidad de nodos explotados tiene un aumento muy significativo. Otra de las desventajas es que hay una varianza muy grande entre el tiempo de ejecución, donde algunas corridas tardaban minutos mientras que otras apenas unos milisegundos. Esos resultados son los esperados, ya que al ser movimientos al azar, en algún momento, el algoritmo elige movimientos que se encuentran en el camino óptimo, mientras que en otros momentos elige movimientos que se encuentran en el peor camino.

Variaciones de IDDFS

En esta sección hablaremos de las variantes de IDDFS que fueron probadas. Como vimos en el primer análisis, la versión optimizada era ordenes de magnitud más eficiente que la versión sin optimización. Ahora veremos el resultado de aumentar el nivel de profundidad por iteración, en el primer análisis el aumento de nivel era de 1.

Tablero	Algoritmo	Step	Pasos a la solución	Tiempo promedio [s]	Nodos explotados

371	IDDFS	5	55	10.11	324158
700	IDDFS	5	55	18.13	484978
800	IDDFS	5	55	7.22	238604
1200	IDDFS	5	35	0.32	12415
371	IDDFS	10	60	6.44	219747
700	IDDFS	10	60	10.42	326480
800	IDDFS	10	60	4.17	158099
1200	IDDFS	10	40	0.27	9969
371	Opt IDDFS	5	55	0.92	15771
700	Opt IDDFS	5	55	0.83	14739
800	Opt IDDFS	5	55	0.21	5875
1200	Opt IDDFS	5	35	0.05	1630
371	Opt IDDFS	10	60	0.92	23296
700	Opt IDDFS	10	60	0.85	23680
800	Opt IDDFS	10	60	0.3	10156
1200	Opt IDDFS	10	40	0.05	1910

Efectivamente podemos ver que aumentar el nivel de profundidad en cada iteración aumenta la performance, pero las soluciones que encuentra no son óptimas, aunque son definitivamente mejores que las encontradas por DFS sin ninguna optimización.

Heurísticas no admisibles

Finalmente, como un punto extra, mostramos el resultado que arrojan los algoritmos ASTAR y GREEDY utilizando las heurísticas no admisibles:

Tablero	Algoritmo	Heuristica	Pasos hasta la solución	solución óptima	Nodos explotados
371	ASTAR	NonEmptySpacesTop	52	52	7590
700	ASTAR	NonEmptySpacesTop	54	53	7939
800	ASTAR	NonEmptySpacesTop	51	51	3217
1200	ASTAR	NonEmptySpacesTo	35	35	884

		p			
371	ASTAR	EmptySpacesTop	54	52	10192
700	ASTAR	EmptySpacesTop	53	53	8822
800	ASTAR	EmptySpacesTop	52	51	3483
1200	ASTAR	EmptySpacesTop	35	35	1002
371	GREEDY	NonEmptySpacesTop	66	52	2822
700	GREEDY	NonEmptySpacesTop	107	53	4235
800	GREEDY	NonEmptySpacesTop	70	51	2732
1200	GREEDY	NonEmptySpacesTop	82	35	461
371	GREEDY	EmptySpacesTop	163	52	8544
700	GREEDY	EmptySpacesTop	140	53	7919
800	GREEDY	EmptySpacesTop	71	51	2229
1200	GREEDY	EmptySpacesTop	69	35	901

Analizando los resultados podemos observar que la heurística 2 es peor que la 1, ya que la mayoría de resultados muestra que explotó más nodos y que tardó más pasos en encontrar la solución. Un resultado interesante que surge es que la heurística 1 devuelve los mismos resultados que en el análisis completo para el algoritmo GREEDY, algo mas interesante aun es que esa misma heurística devuelve resultados en cantidad de pasos muy cercanos a los óptimos obtenidos en el análisis completos. Esto último resulta interesante debido a que si bien en todos los casos explota más nodos, el costo computacional del cálculo de dicha heurística es mucho menor que el de calcular las otras 3 heurísticas admisibles, por lo que, si el tiempo fuese un limitante podría utilizarse esta heurística para obtener un buen resultado aproximado.