NOMBRE: Joaquín Eichholz

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 — Inteligencia artificial — 2' 2019

Tarea 02 – Informe

1 Heurística

La Heurística escogida fue la suma de las distancia de Manhatan real que deben moverse las piezas para llegar a su objetivo dividida por las dimensiones de la tabla para que se admisible. Notemos que se realizo una relajación del problema para realizar la heuristica, ya que se considero que el movimiento de cada pieza puede hacerse independiente de las otras, de esta forma el movimiento de una pieza no afecta (no mueve) a las otras piezas que estan en la fila o columna. Dado esto y que se considero que la heuristica es menor igual a la cantidad de piezas que puede arreglar un movimiento, la heuristica provista es admisble y se explica en mayo detalle a continuación

Esta heuristica se dividio en 2 sub heuristicas, la distancia en x y en y, para cada pieza se calculó la distancia de su posición x hasta la columna donde realmente debería estar. Es importante hacer notar que la pieza puede atravezar paredes y por lo tanto, a lo más deberá recorrer la mitad de las columnas para llegar a su posición objetivo. Finalmente, la suma de estas heuristicas en x (h_x) se deben dividir por él ancho del problema para que sea admisible, ya que en 1 movimiento hacia la izquierda o derecha, se pueden arreglar a lo más m espacios, siendo m el ancho del tablero. Por ejemplo si tenemos un tablero de $1x3$ y la fila tiene todos sus números corridos un espacio a la derecha, el movimiento para arreglarlo es 1 hacia la izquierda y la heurística debe ser menor o igual a 1 es por eso que se divide en el ancho, para que en estos casos, $h_x = 1$ y no $h_x = 3$.

Por otro lado, se calcula la heuristica en y de la misma forma, sumando las distancias de cada ficha hasta su fila correspondiente, notemos que como se pueden atravezar las paredes, la distancia será a lo más la mitad del alto. Finalmetne, al igual que en h_x, la heuristica en y (h_y) bajo el mismo argumento de antes de que 1 movimiento puede acercar n numeros a su posición de llegada, se dividio por el alto del tablero y así la h_y es admisible.

Para terminar, la heuristica retornada es la suma de h_x con h_y.

$$h = h_x + h_y$$

$$h = \sum_{i=1}^{n*m} \frac{min(|p_{xi} - width|, |width - p_{xi}|)}{width} + \frac{min(|p_{yi} - height|, |height - p_{yi}|)}{heigth}$$

Donde p_{xi} representa la columna (posicion x) de la $i - esima$ celda del tablero y p_{yi} representa la fila (posicion y) de la $i - esima$ celda del tablero

2 Iteraciones

2.1 AWA*

Este algoritmo se implemento con las clases suministradas, se realizo un ciclo dentro del tiempo *timebound* en el cúal se realizaba WA*. Al encontrar una solución se seguía explorando la Open para encontrar nuevas soluciones que mejoraran el costo actual (mayor extricto).

2.2 RWA*

Este algoritmo se implemento con las clases suministradas, consiste en realizar RA* varias veces hasta que (1) el tiempo de ejecución acabe (2) no queden estados en la open. En cada iteración se realiza la busqueda bajando el valor de w en un factor de $phi = 0.8$ (notemos que $w >= 1$ para toda iteación) y solo se exploraban nodos con menor costo a la solución ya encontrada

3 Comparación de resultados y analisis

Luego de implementar los 2 algoritmos, se corrieron los 3 algoritmos (WA*, AWA*, RWA*) y se anotaron los costos de las soluciones que encontraron y los tiempos de ejecución que tardaron en encontrar esas soluciones.

Los experimentos fueron realizados con $w = 10$, $timebound = 20$ y $phi = 0.8$

	WA*		AWA*		RWA*		optimo	
	costo solución	tiempo ejecución	costo solución	tiempo ejecución	costo solución	tiempo ejecución	Costo	Tiempo con WA*
Tablero 0	2	0.001645	2	0.001645	2	0.001645	2	0.005194
Tablero 1	4	0.003278	4	0.003767	4	0.003278	4	0.011801
Tablero 2	1	0.002454	1	0.001476	1	0.002454	1	0.001419
Tablero 3	3	0.005559	3	0.003875	3	0.005559	3	0.012556
Tablero 4	1	0.001688	1	0.001382	1	0.001688	1	0.001386
Tablero 5	1	0.001352	1	0.001432	1	0.001352	1	0.001409
Tablero 6	5	0.007759	5	0.008005	5	0.007759	5	0.024577
Tablero 7	8	2.408471	8	2.43424	8	2.408471	7	5.423795
Tablero 7			7	17.077575	7	7.855202		
Tablero 8	8	2.361841	8	2.423452	8	2.361841	7	4.091293
Tablero 8			7	17.419363	7	7.871696		
Tablero 9	8	2.432926	8	2.383545	8	2.432926	7	4.203089
Tablero 9			7	17.155046	7	7.918511		
Tablero 10	22	2.840591	22	2.813167	22	2.850264	--	>2100 [seg]
Tablero 10			20	5.549493	--	--		

Tabla 1: Comparación entre métodos

Como podemos observar de la Tabla 1, hasta el tablero 6 no notamos diferencias con los algoritmos, los 3 algoritmos encuentran la solución optima de manera muy rápida (más rapido que con A*), es por esto que tanto AWA* y RWA* NO encuentran mejores soluciones (porque no existen). Esto se produce debido a que las soluciones de esos tableros son muy triviales y faciles, si bien el algoritmo asegura encontrar soluciones w-optimas, vemos que en estos casos se encontro la solucion optima en menos tiempo

Por otro lado, vemos como en el caso del tablero 7, 8 y 9, el algoritmo WA* no llega a la solución optima y en cambio AWA* y RWA* si encuentran el óptimo, ambos metodos se demoran más tiempo que la ejecuciçon con A*, pero, aún así, podemos ver que RWA* se demora menos de la mitad del tiempo que tarda AWA* en encontrar la solución optima. Sin embargo para tableros más complejos como lo es en caso del ultimo tablero en el cual no se logró obtener una solución con A* luego de 2100 segundos de ejecución, el algoritmo AWA* superó con creces al algoritmo RWA*, dado que logro encontrar una solución optima en 5.5 segundos versus RWA* que no alcanzo a encontrar una solución mejor en el *timebound* de 20 segundos.

No se puede asegurar la optimalidad de la solución encontrada con AWA* pero se sospecha que es bastante buena, ya que se corrio el algorimo y AWA* durante 1000 seg y no se encontro una mejor solución.

En conclusión, para problemas pequeños y faciles es mejor usar directamente A* ya que en estas soluciones no es caro encontrar el óptimo y uno puede asegurar optimalidad. En casos de problemas un poco más compejos pero no tan grandes, el algoritmo RWA* obtuvo mejores resultados que AWA*. Sin embargo, para casos de mayor complejidad se recomienda utilizar el algoritmo AWA* ya que dado los experimentos realizados, se obtuvieron mejores resultados en terminos de tiempo. De igual forma esto es una opinión, ya que dependerá de cada problema, por un lado RWA* reinicia la busqueda disminuyendo el w en cada iteración y por otro lado, AWA* gasta menos tiempo en recacular valores ya que sigue recorriendo la open actual pero no diminuye el valor de w, por lo que la expansión de estados es peor en terminos de la estimación para encontrar el óptimo respecto a RWA*