UNIVERSITAT DE LLEIDA
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Inteligència Artificial

Primera pràctica d'Intel·ligència Artificial

Joaquim Picó Mora
PraLab1

Professorat : Carlos Ansotegui
Data : Divendres 15 de Novembre

Contents

1 Taula Evaluació Experimental

		ucs	bds	bfsh			astar		
				mandH	eucdH	custH	mandH	eucdH	custH
Tiny	Cost	8	8	8	8	10	8	8	8
	Time	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Nodes	15	11	8	8	10	14	13	15
Medium	Cost	68	68	74	152	152	68	68	68
	Time	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Nodes	269	169	78	159	168	221	226	228
Big	Cost	210	210	210	210	210	210	210	210
	Time	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Nodes	620	591	466	471	461	549	557	606
Custom 100x10	Cost	194	194	204	194	220	194	194	194
	Time	0.1	0.1	0.0	0.0	0.0	0.1	0.1	0.1
	Nodes	8671	8299	206	195	292	7194	8532	8603
Custom 500x50	Cost	582	582	850	708	922	582	582	680
	Time	2.2	0.9	0.0	0.0	0.0	0.5	1.0	2.0
	Nodes	166292	83090	1734	1214	3034	22765	62878	109955

Figura 1: Taula Evaluació Experimental.

2 Algoritmes

2.1 Uniform Cost Search

L'ucs és caracteritza per expandir sempre el node al qual s'ha arribat amb el menor cost. Per a realitzar això he declarat la frontera com una cua per prioritat, fent així que cada cop que es tregui un element d'ella per a expandir-lo sigui el que té el cost més baix, ja que la llista és trobarà ordenada per prioritat.

La principal característica d'aquest algoritme i que el fa diferir del Breath First Search és que si trobem un node n1 a la frontera que té un cost determinat, i un altre node n2 el qual ens porta al mateix estat que n1 i amb menor cost. Aleshores afegirem aquest segon node a la frontera, i la mateixa cua per prioritat farà que n2 s'expandeixi abans que n1, ja que tindrà un cost menor i per tant una major prioritat.

Aquest algoritme ens trobarà sempre el camí òptim a la solució. En aquest cas però, al ser el cost lineal, l'espai d'estats finit, el factor de ramificació també finit i havent-hi solució, ens trobarà el camí òptim, però executant-se de forma idèntica a l'algoritme Breath First Search

Podem veure en la Figura1 que l'ucs de tots els algoritmes proposats és el menys eficient quant a espai. En canvis, el temps d'execució és molt proper al de tots els altres algorismes.

2.2 Bidirectional Search

S'ha decidit implementar mitjançant l'algoritme bfs des de dos punts diferents. Aquests punts són el punt d'inici i el punt d'arribada i cada un d'ells tindrà fronteres i llistes de nodes expandits diferents.

D'aquesta forma, cada cop que afegim un element a una de les dues fronteres, comprovarem que no és trobi a la llista de generats de l'altre.

Quan es generen els nodes successors que deriven del punt d'arribada, l'acció d'aquest s'haurà de canviar per la seva antagònica, fent d'aquesta forma que al concatenar els recorreguts el que en resulta sigui vàlid. Per tal de reaprofitar la mateixa funció s'ha decidit implementar-ho amb lambda funcions.

2.3 Greedy Best First Search

Quant al bfsh, és una aplicació de l'ucs, però en canvis d'expandir el node amb menor cost, expandeixen aquells amb l'heurística que més prometi. Per la resta, se segueix exactament el mateix patró de disseny que l'ucs.

2.4 A*

De la mateixa manera que el bfsh l'algoritme A* és una derivació del ucs, però ara en canvi d'expandir el node amb la millor heurística, expandira el que el seu cost sumat a la seva heurística sigui el mínim.

Per assegurar la consistència d'una heurística donada l'admissibilitat, realitzem la funció getPriority la qual ens retornarà el màxim entre l'heurística més el cost entre pare i fill.

3 Heurístiques

3.1 Distancia de Manhattan

Aquesta heurística consisteix a agafar la distància total entre el node i la posició en la qual volem arribar. Per a realitzar-ho sumem amb valor absolut la distancia entre node i goal en l'eix x més la distancia en l'eix y.

3.2 Distància d'Euclides

La distància d'Euclides consisteix a aplicar la regla de Pitàgores amb les distàncies x i y calculades a l'apartat anterior per obtenir la distancia en línia recta en què es troba el node de la posició goal.

3.3 Distancia xAxe

Aquesta és l'heurística que he proposat com a alternativa. Consisteix en la distancia en l'eix de les x entre el node i la posició d'arribada.

4 Anàlisis Evaluació Experimental

Com podem veure, el que expandeix la menor quantitat de nodes i que per tant és el més eficient en termes d'espai és l'algoritme bfsh, aquest si existeix una solució ens la trobarà, però no garanteix que aquesta sigui òptima.

Tots els altres algoritmes garanteixen que la solució que trobaran serà l'optima, però es pot veure que el més eficient serà l'A* degut a que expandeix el menor nombre de nodes d'entre els algoritmes que garanteixen una solució òptima.
Seguidament a l'A* s'hi troba el bds, aquest expandeix menys nodes que l'ucs, això és degut a que expandeix des de dos punts diferents del mapa, i per tant l'amplada de l'expansió prop del punt de trobada serà menor que si ho fes només des d'un únic punt.

Finalment trobem l'ucs com l'algorisme menys eficient dels optimals, ja que expandeix més nodes que els altres dos algoritmes.

Amb el que respecte a temps l'algoritme en trobar una solució més ràpid és el bfsh, però aquesta no és òptima. Per altra banda l'A* amb l'heurística de la distància de Manhattan és el més ràpid d'entre els òptimes.