Universitat de Lleida
Escola Politècnica Superior

07/12/2020

Pràctica 1
Pac-Man Search

Grau en Enginyeria Informàtica

Joel Farré Cortés (78103400T)
Joel Aumedes Serrano (48051307Y)
ESCOLA POLITÈCNICA SUPERIOR – UNIVERSITAT DE LLEIDA

Introducció.

Per a la realització d'aquesta pràctica es proposava, mitjançant els diversos algorismes de cerca informada i no informada vists durant les hores de classe, resoldre diverses qüestions en el clàssic i popular joc del Pac-Man.

Descripció de les decisions preses en la implementació de l'algoritme A*.

Degut a la semblança entre l'A* i l'algoritme UCS implementat a classe, veiem que l'A* és el mateix codi amb l'adició de l'heurística. Per a garantir que l'algoritme sigui òptim, i assumint que la cerca es en graf tal com diu l'enunciat, necessitem una heurística consistent. La tècnica emprada per a transforma una heurística admissible a consistent es la tècnica del *path-max*. Hem optat per aplicar el *path-max* ja que en garanteix una heurística consistent a partir d'una heurística admissible ja que si es admissible i consistent al aplicar el *path-max* seguirà essent consistent.

```
path_max_fn = n.cost
path_max_gnhn = ns.cost+heuristic(ns.state, problem)
path_max = max(path_max_fn, path_max_gnhn)
fringe.push(ns, path_max)
```

Descripció de les heurístiques proposades i de les decisions preses.

- *CornersProblem*

Per a definir la heurística *CornersProblem* el que fem es la distància de Manhattan des del Pac-Man a cada *corner* no visitat, i escollim la mínima. Aquesta heurística és admissible ja que si solament queda una cantonada per visitar, la heurística com a molt serà la distància del Pac-Man a aquesta cantonada i mai es sobreestimarà el cost. Si tenim més d'una cantonada el Pac-Man tampoc ho sobreestimarà perquè la distància mínima no sobreestimarà el cost del camí d'anar a la cantonada més propera i a les restants.

```
for corner in corners:  # Corners no visitats
xy1 = position
xy2 = corner
dist = util.manhattanDistance(xy1, xy2)
if dist < minim:
     minim = dist
return minim
```

- *FoodSearchProblem*

Dins del *FoodSearchProblem* disposem de dues heurístiques: una de consistent i una de no consistent i no admissible però molt més òptima i ràpida que la consistent.

La consistent, semblant al C*ornersProblem*, retorna la mínima de les distàncies als punts per on hem de passar encara, es a dir, les boles de menjar que el Pac-Man encara no s'ha menjat. Al igual que en el problema anterior, la demostració de consistència d'aquesta heurística es fa a

partir de si queda una bola o més d'una. Si queda una bola és consistent ja que es el camí més proper i si en queden més d'una serà el camí més proper més el camí cap a les altres.

Aquesta heurística, en el mapa petit *trickySearch*, donava un camí bastant òptim i no tardava massa. Al mapa gran b*igSearch*, en canvi, la solució tardava molt, fins al punt de no completar-se.

Vam intentar optimitzar l'heurística canviant la distància mínima per la suma de totes les distàncies, pensant que d'aquesta manera el Pac-Man sempre buscaria el lloc amb més boles. Aquest canvi sol ja fa que l'heurística no sigui consistent, però esperàvem un augment de rendiment que no es va donar. La suma també tardava molt, expandia més de 50,000 nodes i la solució mostrada era òbviament lluny de ser òptima. Al veure això vam pensar que podia causar aquests resultats tan dolents. Observant el camí que feia el Pac-Man i tenint en compte el que calculava l'heurística vam descobrir l'error.

Aquest error es donava al *bigSearch* ja que al *bigSearch* el Pac-Man comença al mig del mapa. Això feia que el Pac-Man realitzés contínuament: «menjo una bola i em desplaço cap a l'Est», «ara com que he menjat una bola a l'Est, l'heurística dóna millor resultat anar cap a l'Oest ja que estaré mes prop de la resta de boles», aleshores menjarà dues boles de l'Oest i es repetirà el mateix però ara amb l'Est.

Observant aquest comportament vam crear la nova heurística per a arreglar aquest comportament. El que causava aquest comportament era que el Pac-Man s'anés movent d'Est a Oest augmentant la distància entre les boles de menjar. Per tant vam afegir la distància de Manhattan entre les boles de menjar restants, cada bola amb cada bola. Això feia que el Pac-Man també intentés disminuir aquesta dada i ho feia menjant-se primer totes les boles de l'Oest i després totes les de l'Est.

Aquest canvi, tot i que causa que l'heurística no sigui ni admissible ni consistent genera un increment massiu de l'eficiència, trobant la solució al *bigSearch* en menys de 15 segons, expandint solament 300 nodes i amb una solució que requereix 700 passos menys que l'heurística original.

La última heurística que hem fet i que considerem millor està implementada a foodHeuristic, mentre que la consistent està implementada a foodHeuristicConsistent.

Opcional: Búsqueda subòptima

Hem pogut realitzar aquesta búsqueda dividint-la en búsquedes més petites. L'agent *ClosestDotSearchAgent* consisteix en anar sempre al punt de menjar més proper, i per a fer això utilitzem el problema *AnyFoodSearchProblem*, un problema que consisteix en arribar a qualsevol punt de menjar. Aleshores, el que farem serà que cada cop que el *ClosestDotSearchAgent* soliciti el camí a la propera bola, es crearà i es solucionarà un problema *AnyFoodSearchProblem* amb les boles de menjar i la posició del Pac-Man com són al problema del *ClosestDotSearchAgent*.

Per a definir el problema *AnyFoodSearchProblem* sol ens fa falta definir la funció *isGoalState*. En aquesta funció sol hem de comprovar si a la nostra posició actual hi ha una bola de menjar o no.

A continuació, a la funció *findPathToClosestDot* sol ens farà falta crear un problema *AnyFoodSearchProblem* a partir de les dades de l'estat guardades a *GameState* i solucionar-lo per a que ens retorni solament el camí a la bola més propera. Això ho podem fer amb una crida al BFS implementat a classe.