Pràctica 1: Navegació

Part 2: Cerca Informada

Intel·ligència Artificial

2023-2024

Universitat Autònoma de Barcelona

1 Introducció

Durant la pràctica anterior vam veure com treballar amb la classe `Path`, implementant dos mètodes de Cerca no informada (cerca en amplada i cerca en profunditat) per trobar una ruta entre dos punts. Aquests dos mètodes no són òptims, ja que no fan servir costos, simplement cerquen el primer camí que porta a la solució sense considerar el cost, ni considerar cap mena d'heurística que pugui ajudar a reduir el cost de la cerca.

En aquesta segona part del projecte introduirem el concepte de cost i d'heurística als algorismes de cerca, fet que ens permetrà trobar solucions òptimes i reduir el cost de la cerca.

2 Fitxers necessaris

Seguirem treballant amb els mateixos fitxers que a la Part 1 de la pràctica, i un altre cop, tot el que programareu es farà en el fitxer `SearchAlgorithm.py` on ja es van programar totes les funcions de la Part 1 de la pràctica.

3 Preparació

Abans de començar a programar és molt recomanable entendre els conceptes que treballarem en aquesta part de la pràctica: **Cost i Heurística**. En cas de dubtes, podeu trobar una explicació sobre com es poden calcular als apunts de la teoria. Per tant, abans de seguir, hauríeu de ser capaços de:

1. Entendre el concepte de **Cost**, el qual representarem amb la funció g, i com poder-lo calcular amb la informació que teniu a la classe `Map`. També heu de veure com s'interpreta aquest concepte en cadascun dels diferents criteris plantejats: parades,

temps, distància, transbord. Per implementar els diferents criteris és important que tingueu en compte la següent informació:

- El temps entre totes les parades de totes les línies ens ve donat.
- Cada línia té una velocitat constant donada.
- Cada estació té unes coordenades donades que són compartides per les diferents línies d'una mateixa estació (suposarem doncs que un transbord no comporta un cost en distància)
- Les vies entre estacions no van sempre en línia recta.

2. Entendre el concepte d' **Heurística** com estimador del cost que pot tenir desplaçar-se de l'estació actual fins a la solució, el qual és desconegut. Tal com hem vist a teoria l'heurística la representarem amb la funció h. Aquesta funció heurística dependrà del cost que estimi, haurem de definir una heurística per a cada criteri que vulguem optimitzar, sigui temps, transbords o distància. Per tant, cada g té la seva h.

4 Què s'ha de programar?

En la segona part d'aquesta pràctica programareu dos algorismes de Cerca: la Cerca de Cost Uniforme i la Cerca A*. Ambdós algorismes fan cerca òptima fent servir el cost acumulat dels camins, i en particular, l'A* també fa servir una heurística per millorar el temps que triga a trobar la solució òptima.

En aquesta Part 2 del projecte farem servir les mateixes funcions `Expand` i `Remove_cycles` que vàrem implementar a la Part 1 del projecte. Els càlculs dels costos i les heurístiques dels camins s'aplicaran sobre les llistes de camins resultants d'aquestes dues funcions.

Primer programarem les funcions de la Cerca de Cost Uniforme i seguidament programarem l'A*.

4.1 Cerca de Cost Uniforme

Haurem de programar les següents tres funcions:

`Calculate_cost:` Funció que pren com a entrada la llista de `Paths` fills, el mapa (`Map`), i un número (`int`) que fa referència al criteri que calcularem (parades, temps, distància, transbords). Calcula el cost des de la penúltima estació que estàvem explorant fins a l'última (fill actual) i el suma al que ja es tenia. Això ho fa per a cada un dels `Paths` fills de la llista, i actualitza el valor total de cost de cada camí.

EXEMPLE:
Crida: `expand([14,13,8,12,g=15]),MAP)`
Retorna: `[[14,13,8,12,8,g=15],[14,13,8,12,11,g=15],[14,13,8,12,13,g=15]]`
Crida: `calculate_cost([[14,13,8,12,8,g=15],[14,13,8,12,11,g=15],[14,13,8,12,13,g=15]],1)`
Retorna: `[[14,13,8,12,8,g=17],[14,13,8,12,11,g=21],[14,13,8,12,13,g=20]]`
**** Pista: Utilitzeu la funció de la classe* `Path "update_g"`

`Insert_cost:` Funció que pren com a entrada la llista de `Paths` fills que hem expandit i la llista global de `Paths` explorats de l'arbre. La funció retorna la unió d'aquestes dues llistes ordenades segons el cost, de manera que el camí de millor cost queda a davant de tot, ja que aquest serà el següent camí que expandirem.

`Uniform_cost_search:` Funció que donada una estació d'origen, una estació destí, el mapa de la ciutat i el tipus de cost que volem avaluar representat per un número (`int`), retorna una ruta òptima entre les dues estacions implementant l'algorisme de cerca de cost uniforme que es dona a la figura 1.

```
Funció CERCA_cost_uniforme (NodeArrel, NodeObjectiu)

1. Llista=`[ [ NodeArrel ] ];
2. Fins que (Cap(Cap(Llista))=NodeObjectiu O bé (Llista=NIL) fer
   a) C=Cap(Llista);
   b) E=Expandir( C );
   c) E=EliminarCicles(E);
   d) Llista=Inserció_ordenada_g(E,Cua(Llista));
3. Ffinsque;
4. Si (Llista<>NIL) Retornar(Cap(Llista));
5. Sinó Retornar("No existeix Solucio");
Ffuncio
```

Figure 1: Pseudo-codi de la Cerca de Cost Uniforme vist a teoria.

4.2 Cerca A*

Per aquest algorisme haurem de programar primer dues funcions que implementaran l'ús de l'heurística, una funció que permetrà eliminar camins redundants (això és, camins parcials no òptims), i finalment dues funcions que implementaran la Cerca A* en si. Ho detallem tot seguit.

Funcions que gestionen l'heurística:

`Calculate_heuristics:` Funció que pren com a entrada la llista de `Paths` fills, el mapa (`Map`), la ID de l'estació destí i un número (`int`) que fa referència al criteri que intenta estimar l'heurística (parades, temps, distància, transbords). Calcula l'heurística entre l'última parada que estem explorant de cada `Path` fill i l'estació final. Una vegada calculada, actualitza el valor d'heurística de cada `Path` fill.

**** Pista: Utilitzeu la funció de la classe* `Path "update_h"`

`Update_f:` Funció que pren com a entrada la llista de `Path` fills, als quals hem actualitzat prèviament el cost i l'heurística i retorna la mateixa llista amb el cost total actualitzat per tots els camins.

**** Pista: Utilitzeu la funció de la classe* `Path "update_f"`

Funció que elimina camins redundants.

Durant la navegació podem trobar que arribem a una mateixa estació utilitzant camins diferents. Per a optimitzar la nostra cerca, cal deixar d'explorar qualsevol camí que arriba a una estació que ja hem explorat abans amb un cost parcial millor. Un camí parcial no òptim serà un camí redundant que mai formarà part d'una solució òptima. Per tant, caldrà definir la funció `Remove_redundant_paths` que s'encarregarà d'eliminar aquests camins redundants.

Per a poder implementar aquesta funció necessitem guardar en tot moment quin és el cost òptim per a arribar a cada estació. Haurem de crear un diccionari, `visited_stations_cost`, que guardarà la informació de les estacions visitades i el cost mínim fins a elles en cada moment, serà la taula de costos parcials.

`Remove_redundant_paths:` La funció pren com a entrada la llista de `Paths` fills que acabem d'expandir, la llista global de `Paths` de l'arbre explorat i el diccionari de costos parcials. La funció s'encarrega de comprovar si un dels nous `Paths` fills, ha arribat a un node que ja havíem explorat abans i del que guardem el seu cost al diccionari, en cas afirmatiu comprovarà si el nou cost és millor o pitjor que l'anterior:

- Si el cost anterior és millor o igual, eliminarem el nou camí de la llista de `Paths` fills.
- Si el cost anterior és pitjor, actualitzarem el nou cost al diccionari, i eliminarem aquells camins de la llista global de `Paths` que contenen aquest node, ja que tots ells arribaven al node d'una manera subòptima.

La funció ha de retornar dues llistes i un diccionari. La primera llista és la dels `Paths` fills sense camins redundants, la segona llista és la llista global de `Paths` explorats, també sense camins redundants, i finalment el diccionari de tots els nodes visitats amb els costos parcials òptims actualitzats.

Funcions que implementen la Cerca A*:

`Insert_cost_f:` Funció que pren com a entrada la llista de `Paths` fills que hem expandit, i la llista global de `Paths` explorats de l'arbre. La funció retorna la unió d'aquestes dues llistes ordenades segons el cost total estimat ($f = g + h$), de manera que el camí amb menor cost total estimat quedi a davant de tot, ja que aquest serà el següent que expandirem.

`A_star:` Funció que donada una estació d'origen, i una estació de destí, el mapa de metro d'una ciutat i el tipus de criteri que volem optimitzar, que ve representat per un número (`int`), cerca una ruta òptima entre les dues estacions utilitzant l'algorisme A* que s'especifica a la figura 2.

```
Funció CERCA_A* (NodeArrel, NodeObjectiu)
1.    Llista= [ [NodeArrel] ];
2.    Fins que (Cap(Cap(Llista))=NodeObjectiu O bé (Llista=NIL) fer
      a)    C=Cap(Llista);
      b)    E=Expandir( C );
      c)    E=EliminarCicles(E);
      d)    EliminarCaminsRedundants(E,Llista,TCP);
      e)    Llista=Inserció_ordenada_f(E,Llista);
3.    Ffinsque;
4.    Si (Llista<>NIL) Retornar(Cap(Llista));
5.    Sinó Retornar("No existeix Solucio");
Ffuncio
```

Figure 2: Pseudo-codi de la Cerca A* vist a teoria.

5 Una funció addicional: coordenades enlloc d'estacions

En aquesta secció us demanem que implementeu una versió addicional d'A*, pel cas en que la posició de l'usuari vingui donada en coordenades i no estacions. Fins ara, hem assumit que l'usuari del nostre navegador, coneix les millors estacions d'orígen i de destí. Ara bé, normalment, això no és així. L'usuari coneix on és i on vol anar, però no les estacions més convenients per fer el trajecte més òptim.

En aquesta part del projecte es planteja implementar el mètode de la cerca A* donant coordenades d'orígen i de destí. La cerca A* haurà de trobar la millor ruta, tenint en compte que l'usuari pot desplaçar-se a l'estació més convenient, la qual no té per què ser la més propera. Per simplificar el problema es considerarà que l'usuari s'apropa a aquesta estació caminant en línia recta a una velocitat constant de 5. Aquesta millora només té sentit per al criteri de temps, ja que per a qualsevol dels altres criteris el camí que minimitzaria el cost seria anar caminant des del punt d'orígen al punt de destí.

A la ruta que retornarà aquest algorisme, en la primera posició sempre hi haurà un 0, el qual representa les coordenades origen. L'última posició de la ruta serà sempre un -1, el qual representa les coordenades de destí. És a dir, la ruta `[0,-1]` indica que l'usuari es desplaça caminant des de l'origen al destí, mentre que una ruta `[0,8,9,10,-1]` indica que l'usuari es desplaça caminant des de l'origen a l'estació 8, de l'estació 8 fins la 10 va en metro, i de la 10 es desplaça caminant al destí.

Per afegir aquesta funcionalitat final haureu d'afegir una nova funció, la qual anomenareu Astar_improved.

`Astar_improved`: Funció que donades unes coordenades d'origen, unes coordenades de destí i el mapa de metro d'una ciutat, cerca una ruta òptima en temps entre les dues coordenades, suposant que:

1. l'usuari va en línia recta a velocitat constant de 5 quan va caminant,
2. l'usuari pot anar caminant a qualsevol de les estacions del mapa i també a les coordenades de destí.
3. l'usuari pot anar caminant des de les coordenades d'origen i des de qualsevol de les estacions a les coordenades de destí.

4. l'usuari no es desplaça d'una estació de metro a una altra caminant.

6 Entrega de la Part 2

Per a l'avaluació d'aquesta segona part de la pràctica haureu de pujar al Campus Virtual el vostre fitxer `SearchAlgorithm.py` que ha de contenir el vostre NIU a la variable author que hi ha a l'inici de l'arxiu. L'entrega s'ha de fer abans del dia **17/03/2024** a les **23:55**.

ATENCIÓ! és important que tingueu en compte els següents punts:

1. La correcció del codi es fa de manera automàtica, per tant, assegureu-vos de penjar els arxius amb la nomenclatura i format correctes.
2. El codi està sotmès a detecció automàtica de plagis durant la correcció.
3. Qualsevol part del codi que no estigui dins de les funcions de l'arxiu `SearchAlgorithm.py` no podrà ser avaluada, per tant, no modifiqueu res fora d'aquest arxiu.
4. Tant la correctesa com l'eficiència del codi es valoraran, per tant, si les vostres funcions triguen massa, el codi les comptarà com a errònies.