Pràctica 1: Navegació

Part 2: Cerca Informada

Intel·ligència Artificial

2019-2020

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. Aquest dos mètodes no són òptims, ja que no fan servir costos i simplement cerquen el primer camí que porta a la solució sense considerar cap cost, i sense considerar cap tipus 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 que ens permetrà trobar solucions òptimes i a més a més intentant arribar-hi el més ràpidament possible.

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 sobre el fitxer `SearchAlgorithm.py` on ja es van programar totes les funcions de la part 1 del projecte.

3 Preparació

Abans de començar a programar es molt recomanable entendre els conceptes que treballarem en aquesta part de la practica: **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** i com poder calcular-lo amb la informació que teniu a la classe `Map`, que el representarem amb la funció g. I veure com es pot estendre a

diferents criteris (parades, temps, distància, transbord). Per usar 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 falta des d'una estació fins a la solució, que 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 dels camins, i a més a més l'A* fa servir l'heurística per millorar el temps que triga en trobar la solució.

En aquesta Part 2 del projecte farem servir les mateixes funcions `Expand` i `Remove_cycles` que varem implementar a la Part 1 del projecte. Els càlculs dels costos i les heurístiques del 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 la última (fill actual) i el suma al que ja 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 dóna 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 alquest algorisme haurem de programar primer dues funcions que implementaran l'ús de l'heurística, una funció que permetrà eliminiar camins redundants (això és, camins parcials no òptims), i finalment dues funcions que implementaran la Cerca A* en sí. 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, distancia, 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 cami 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 es 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.

`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 es pitjor, posarem el nou cost al diccionari, i eliminarem aquells camins que contenien aquest node, ja que tots els camins arribaven a aquell 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 lista és la llista global de `Paths` explorats, també sense camins redundants, i finalment el diccionari de tots els nodes visitats amb el costos 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 de millor cost total estimat quedi a davant de tot, ja que aquest serà el següent que expandirem.

`A_star:` Funció que donada les COORDENADES d'una posició d'orígen, d'una posició 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 els dos punts 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 Funcions addicionals

Tot seguit us donem les especificacions per poder fer les funcions addicionals:

Heurística de la distància

`calculate_heuristics:` modificar aquesta funció pel `type_preference=2`

`calculate_cost:` modificar el `type_preference=2`

Heurística de les parades

`calculate_heuristics:` modificar aquesta funció pel `type_preference=0`

`calculate_cost:` modificar el `type_preference=0`

Heurística dels transbords

`calculate_heuristics:` modificar aquesta funció pel `type_preference=3`

`calculate_cost:` modificar el `type_preference=3`

5.1 Introducció de coordenades a l'A*

La funció que hem fet servir per aplicar l'A* a partir d'un orígen donat en coordenades es basa en anar a l'estació més propera. Ara bé, poodria ser que l'estació més propera no fos l'estació més optima per arribar al nostre destí. Veure l'exemple de la Figura 3.

En aquesta part del projecte es planteja implementar el mètode de la cerca A*, millorant el problema vist a l'exemple vis anteriorment. Es planteja que donades unes coordenades inicials i finals, la cerca A* trobi la millor ruta per a qualsevol dels criteris, tenint en compte que l'usuari pot desplaçar-se a l'estació més convenient. Es considerarà que l'usuari s'apropa a aquesta estació caminant en línia recta a una velocitat de 5.

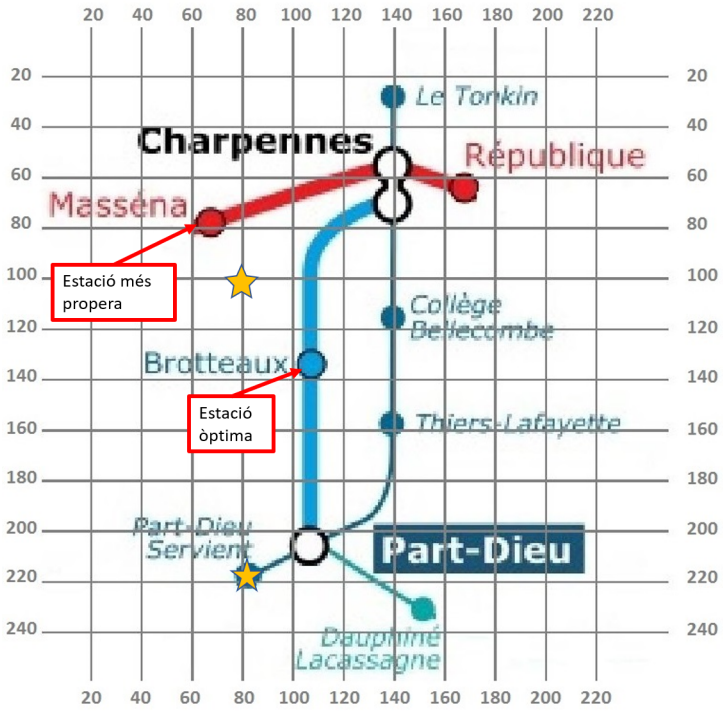

Figure 3: Exemple de dues possibles rutes entre el punt orígen $(80, 100)$ i el punt destí $(80, 220)$. Ruta 1: Anar a l'estació més propera (Masséna), i fer 6 parades comptant els transbords. Ruta 2: Anar a l'estació Brotteaux, i fer només 3 parades comptant els transbords.

Per a programar-ho haureu de crear una Nova funció anomenada Astar_improved per a programar-ho. En el cas que implementeu aquesta funcionalitat de l'A*, ho heu d'indicar al professor de pràctiques per a la seva correcció. Ja que aquesta part no es corregeix automàticament.

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 NIA a la variable authors i el vostre grup a la variable group (a l'inici de l'arxiu).

ATENCIÓ!

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 esta sotmès a detecció de plagis automàtica 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. Per evitar que el codi entri en bucles hi ha un límit de temps per a cada exercici, per tant si les vostres funcions triguen massa les comptarà com a error.