Proiect Inteligenta Artificiala

Popa Bogdan-Gabriel
Grupa 364

Facultatea de Matematica
si Informatica

Rezumat enunt

Enuntul problemei presupunea aranjarea unor blocuri de greutati si rezistente pe un numar dat de coloane astfel incat coloanele sa aiba fiecare ***n*** sau ***n+1*** blocuri, unde n este (***numarul total de blocuri / numarul de coloane***). Insa, deasupra unui bloc nu pot fi puse blocuri cu suma greutatilor mai mare decat rezistenta blocului pe care sunt, aceasta fiind o stare invalida.

Functii si Clase

Ca algoritmi folositi pentru cautarea solutiilor au fost folositi toti algoritmii ceruti: DF, DFI, BF, UCS, A*, A* Optimizat si A*I. Acesti algoritmi au fost implementati ca cei de la laborator si adaptati pentru a se folosi de clasele implementate de mine pentru a se adapta la problema data.

Pe langa functiile de cautare mai exista si o functie de citire din fisier, care primeste ca parametrii numele fisierului din care urmeaza a se citi si euristica care urmeaza a fi folosita pentru algoritmii care o si folosesc. In cadrul acestei functii se determina si starea initiala, daca aceasta este solutie, sau daca aceasta este invalida, oprind programul in oricare dintre aceste cazuri.

Clasele care au fost create pentru rezolvarea problemei sunt urmatoarele:

- **Bloc**: aceasta este o clasa care memoreaza datele despre un bloc dat de problema (numele, masa si rezistenta);

```
class Bloc:
    def __init__(self, nume, g, r):
        self.nume = nume  # memoram numele blocului
        self.g = g  # greutatea blocului
        self.r = r  # rezistenta blocului
```

- **Stare**: aceasta clasa contine informatii despre coloanele actuale, parintele care a dus la aceasta stare, valoarea starii, care reprezinta cat de aproape este de solutie (cu cat acesta este mai mic cu atat este mai aproape de solutie, cand este 0, aceasta este o solutie), iar ultima variabila memorata este costul intregului drum pana la starea curenta. Pe langa variabilele memorate, exista si o serie de functii ajutatoare: *valoare_stare()* se foloseste de una din euristicile introduse ca parametru a determina valoarea starii actuale, *obtineDrum()* returneaza o lista cu toate starile coloanelor care au dus la starea curenta, *afisDrum()* se ocupa de afisarea solutiei dupa formatul cerut in enunt, *contineInDrum()* determina daca o stare se regaseste in drumul starii din care a fost apelata functia si *stare_valida()* determina daca coloanele starii respective constituie o stare valida;

```python
class Stare:
    def __init__(self, coloane, parinte=None, cost=0, euristica="banala"):
        self.coloane = coloane  # list de coloane
        self.parinte = parinte  # parintele din arborele de parcurgere
        self.valoare = self.valoare_stare(euristica)  # aproximare a costului pentru a ajunge la o solutie
        self.cost = cost  # costul total al blocurilor mutate pana la starea actuala
```

- **Graph:** clasa respectiva memoreaza starea de start si timpul de incepere al programului, iar ca functii se regasesc: *genereazaSuccesori()* care, dupa cum implica si numele, genereaza toate ramificarile care pot aparea dintr-o stare primita ca parametru. In cadrul acestei functii se verifica si daca o stare succesor este solutie si se afiseaza in cazul acesta direct. Cea din urma functie a clasei Graph este *testeaza_scop()* care determina daca starea pasata ca parametru este sau nu solutie;

```python
class Graph:  # graful problemei

    def __init__(self, start):
        self.start = start  # informatia nodului de start
        self.start_time = time.time()  # timpul de la inceperea programului
```

Euristici

Ca euristici am acoperit toate tipurile de euristici cerute de enunt, iar toate s-au folosit de variabila *valoare* din clasa Stare pentru a determina cat de favorabila este o stare:

- **Euristica banala**: pentru aceasta, *valoare* este fie 1 daca starea nu este solutie, fie 0 daca starea este solutie;
- **Euristica admisibila 1**: pentru aceasta, *valoare* este dat de numarul de blocuri care ar trebui mutate pentru a se ajunge la o stare solutie, deci daca nu trebuie mutat niciun bloc el va fi 0, deci este solutie;
- **Euristica admisibila 2**: pentru aceasta, *valoare* este dat de suma costurilor (greutatilor) blocurilor care ar trebui mutate pentru a se ajunge la o stare finala, iar daca niciun bloc nu trebuie mutat suma va fi 0, deci este solutie;
- **Euristica neadmisibila**: pentru aceasta am luat formula de obtinere a valorii de la euristica admisibila 1 si am inmultit valoarea obtinuta cu 1000;

Exemple de date de intrare

In cadrul proiectului se pot regasi fisierele de intrare cerute cu criteriile din enunt: *blocheaza.txt* (la df)*, nu_blocheaza.txt, solutie_initiala.txt si fara_solutii.txt*.

In cadrul functiei main se pot modifica variabile care sa modifice decursul programului *nrSolutiiCautate* pentru obtinerea unui numar fix de solutii, *euristica_aleasa* pentru alegerea unei euristici diferite pentru algoritmii care o folosesc si *fisier_ales* care determina care vor fi datele de intrare.

Pe langa acestea, tot in main se regasesc si apelurile functiilor de cautare, fiecare avand posibilitatea de a fi decomentata si rulata.

Ex 1: solutie_initiala.txt

Date de intrare:

```
c,3,10|a,5,14|g,2,8
e,10,17|b,8,10
d,10,18|f,4,9
```

Consola:

```
C:\Users\user\.virtualenvs\ProiectKR\Scripts\python.exe C:/Users/user/PycharmProjects/ProiectKR/main.py
Starea initiala este solutie!

[g/2/8]
[a/5/14]   [b/8/10]   [f/4/9]
[c/3/10]   [e/10/17]   [d/10/18]
==================================

Process finished with exit code 0
```

Ex 2: nu_blocheaza.txt, nrSolutiiCautate=2, BF

Date de intrare:

```
c,3,10|a,5,14|g,2,8
e,10,17|b,8,10
d,10,18
```

Consola:

```
Coada actuala:
[[(c, 3, 10), (a, 5, 14), (g, 2, 8)]
[(e, 10, 17), (b, 8, 10)]
[(d, 10, 18)]
Valoare: 1
--------------------------------
]
Solutie:
[g/2/8]
[a/5/14]   [b/8/10]
[c/3/10]   [e/10/17]   [d/10/18]
==================================
[a/5/14]   [b/8/10]   [g/2/8]
[c/3/10]   [e/10/17]   [d/10/18]
==================================
Cost: 2
Timp cautare: 0.0 s
##################################

Coada actuala:
[[(c, 3, 10), (a, 5, 14)]
[(e, 10, 17), (b, 8, 10), (g, 2, 8)]
[(d, 10, 18)]
Valoare: 1
--------------------------------
, [(c, 3, 10), (a, 5, 14), (g, 2, 8)]
[(e, 10, 17)]
[(d, 10, 18), (b, 8, 10)]
Valoare: 1
--------------------------------
]
Solutie:
[g/2/8]
[a/5/14]   [b/8/10]
[c/3/10]   [e/10/17]   [d/10/18]
==================================
          [g/2/8]
[a/5/14]   [b/8/10]
[c/3/10]   [e/10/17]   [d/10/18]
==================================
[a/5/14]   [b/8/10]   [g/2/8]
[c/3/10]   [e/10/17]   [d/10/18]
==================================
Cost: 4
Timp cautare: 0.0009949207305908203 s
##################################


Process finished with exit code 0
```