Documentatie proiect PTP

1. Introducere

Proiectul se ocupa de Pacient Transportation Problem, in care un numar de pacienti trebuie transportati la spital si inapoi in functie de un timp dat si un numar limitat de vehicule.

In abordarea problemei am folosit algoritmul Branch and Bound, pentru gasirea unei solutii cat mai optime in timp eficient.

2. Etape

2.1. Modelare

Modelarea a constat in reprezentarea problemei in obiecte:

- Place.py, care contine id-ul locatiei, coordonatele carteziene, si categoria (0 – spital, 1 – depou, 2 – locatia pacientului)
- Vehicle.py, care contine id-ul vehiculului, categoriile de pacienti pe care ii poate transporta, depoul de unde pleaca si unde trebuie sa se intoarca, numarul maxim de locuri din vehicul, programul de functionare si istoricul locatiilor vizitate.
- Patient.py, care contine id-ul pacientului, categoria in care se incadreaza pacientul, numarul de locuri ocupat in vehicul, locatia initiala a pacientului, spitalul la care trebuie sa ajunga, locatia la care trebuie sa se intoarca dupa vizita medicala, timpul la care incepe vizita medicala si durata acesteia, impreuna cu durata de timp in care pacientul se urca/coboara din vehicul.
- Request.py, modelat dupa Tabelul 1[1]
- Activity.py, definita ca o instanta din orarul unui vehicul, prin timp, loc, requestul de care a fost creata si incarcatura vehiculului dinainte si dupa efectuarea activitatii.
- Instance.py, rezultatul returnat de cautare, definita de un set de Request-uri si un set de Vehicles.
- Problem.py: Clasa parinte in care este prezent algoritmul de cautare, importarea datelor din fiserele json[2].
- Interface.py: Clasa care se ocupa de reprezentarea grafica.

Motivatia pentru alegerea unei astfel de modelare:

Clasa Request a fost creata pentru a incapsula toate detaliile unei cereri facuta de un pacient.

Clasa Activity a fost creata pentru a defini locurile prin care trebuie sa treaca vehiclul in programul sau pentru a satisface toate requesturile care i-au fost asignate. Asignarea se realizeaza astfel: Un request poate fi scris ca un set de 6 activitati, 3 forward si 3 backward. Cele 3 forward fiind in ordine: activitate1 pentru timpul si locul in care vehiculul pleaca spre locatia pacientului; activitate2 pentru timpul la care ajunge la casa pacientului; activitate3 pentru timpul la care ajunge la spital si incepe vizita medicala. Cele 3 backward fiind in ordine: activitate1 pentru timpul si locul in care vehiculul pleaca spre spitalul in care este pacientul; activitate2 pentru timpul la care ajunge la spital; activitate3 pentru timpul in care ajunge la locația de întoarcere a pacientului.

Clasa Instance a fost făcută ca o subclasa a problemei, în care putem incapsula requesturile care trebuie satisfacute și asignarea lor la vehicule + vehiculele disponibile în care putem găsi parcursul lor de-a lungul programului.

2.2. Implementarea cautarii

Am definit un set de functii ajutatoare:

- getRequests() care creeaza si preia toate requesturile din clasa Patients
- getBestRequest(), euristica folosita pentru a evalua un anumit request din subtree-ul cautarii, folosindu-se de SlackTime (timpul petrecut de request, cu cat e mai mic cu atat mai bine) si numarul de requesturi satisfacute pana in momentul apelarii, care trebuie maximizata prin definitia problemei.
- getBestVehicle(), euristica folosita pentru a returna cele mai bune vehicule potrivite pentru drumurile forward si backward ale unui request. Aceasta cauta masinile cele mai pline, pentru a elibera restul de vehicule disponibile.
- reqTime(), returneaza timpul total folosit de activitatile unui request.
- checkVehicle(), verifica daca vehiculul poate satisface toate cerintele curente.
- insertForward() si insertBackward(), care insereaza activitatile definite de request in istoricul de activitati ale vehiculului
- search() si subsearch(), algoritmii care se ocupa de cautare

Functia de cautare foloseste metoda Branch and Bound, deoarece din natura problemei, in cautarea cu backtracking vom ajunge mai mereu in situatii in care vehiculele nu vor putea satisface unul din drumurile requestului (forward sau backward), in acest caz requestul devine imposibil de satisfacut. Pentru a usura cautarea, Branch and Bound elimina instant cautarea in astfel de requesturi.

Conform desenului din figura 3[3], cautarea sorteaza requesturile dupa timpul de start, si verifica in ordine subtree-urile generate de selectia (respectiv non-selectia) requestului curent, alegand-o pe cea care returneaza euristica maxima dintre cele 2.

2.3. Rezultate

Comparate cu rezultatele din documentatia citata in bibliografie, rezultate algoritmului nostru se comporta astfel: |R| = numarul de requests, |V| = nr de vehicule, |H| = nr de spitale

Dificultatea	Valorile parametrilor	Cel mai bun rezultat	Rezultatul nostru
Easy	|H|=4; |V|=2; |R|=16	15	10
	|H|=8; |V|=4; |R|=32	32	29
	|H|=16; |V|=6; |R|=64	62	43
	|H|=20; |V|=8; |R|=80	74	55
	|H|=24; |V|=9; |R|=96	95	69
Medium	|H|=8; |V|=2; |R|=16	12	10
	|H|=16; |V|=3; |R|=32	19	16
	|H|=24; |V|=4; |R|=48	32	23
	|H|=32; |V|=4; |R|=64	37	28
	|H|=40; |V|=5; |R|=80	55	37
Hard	|H|=48; |V|=4; |R|=48	32	16

3. Echipa

Echipa acestui proiect este compusa din :

- Agarafine Beniamin Andrei
- Ciocirtau Dragos
- Paunescu Andrei Alexandru
- Putanu Alexandru
- Tugui Stefan Cristian

Responsabilitatile fiecaruia au fost:

Intreaga echipa s-a ocupat de modelarea problemei si alegerea claselor Activity si Request, principalele clase folosite in algoritm;

Agarafine Beniamin si Paunescu Andrei s-au ocupat de functiile setForward() si setBackward(), de testare+patching pentru functia subsearch() si de cateva constrangeri din checkVehicle().

Ciocirtau Dragos s-a ocupat de functia checkVehicle(), reqTime(), si de transformarea tuturor timpilor din clase din “00h00” in timedelta.deltatime()

Tugui Cristian s-a ocupat de interfata grafica, functiile euristice (getBestVehicle() si getBestRequest()), preluarea datelor din fisierul .json

Putanu Alexandru s-a ocupat de finisarea functiilor setForward() si setBackward(), a ajutat la definirea requirement-urilor pentru vehicule (checkVehicle() si reqTime()), de functia search() impreuna cu subsearch() si de testare+patching pentru varianta finala