Nume: Sprincenatu Bogdan-Andrei

Grupa: 332CC

1. Reprezentarea unei stari a problemei

- Pentru rezolvarea temei am considerat in prima faza o modalitate de reprezentare a unei stari a problemei prin intermediul clasei **State**, inspirata din laboratorul 2 de IA(**Hill Climbing**).
- O stare a problemei contine urmatoarele elemente:
    i. ***Orarul***, reprezentat sub forma unui dictionar de zile avand ca valori dictionare de intervale, ale caror chei sunt numele salilor avute la dispozitie(cele citite din fisierul **yaml** dat ca parametru in linia de comanda) si valorile reprezinta tupluri de forma **(nume_profesor, disciplina_predata)** (ideea de reprezentare a fost preluata din scheletul de cod pus la dispozitie inca de la inceput, pentru a putea utiliza functia **pretty_print_timetable).**
    ii. ***Numarul de conflicte,*** ce reprezinta in acest context numarul total de constrangeri soft incalcate pentru configuratia curenta a orarului.
    iii. ***Numarul de ore predate de fiecare profesor,*** ce reprezinta un dictionar continand asocierile {**Profesor**: **Numar ore saptamanale**}, folosit pentru verificarea faptului ca niciun profesor nu are mai mult de 7 intervale alocate.
    iv. ***Ore tinute,*** ce reprezinta un dictionar in care calculez pentru fiecare materie cati studenti au mai ramas sa fie distribuiti pentru a acoperii materia respective.
    v. ***Intervalele profesorilor,*** ce reprezinta un dictionar in care salvez intervalele din fiecare zi a fiecarui profesor in care acesta preda.
- Pentru a putea realiza actiuni asupra orarului si genera urmatoarele stari posibile ce pot conduce la o solutie pentru restrictiile impuse, am considerat 2 functii importante si anume:

- ***apply_action(day, interval, discipline),*** in care caut toate salile si toti profesorii posibili ce pot preda materia respectiva in ziua si intervalul dat. In interiorul acestei functii generez doar acele stari posibile care pot conduce la o configuratie valida in care restrictiile hard sunt respectate, pe cat si cele soft ce tin cont de preferintele legate de o anumita zi(de exemplu nu caut o sala care nu poate sustine sau un profesor ce nu preda materia respective, nu las un profesor sa predea in acelasi interval mai mult de o data, nu las un profesor sa predea de mai mult de 7 ori, nu caut adaugarea unui profesor pentru o materie intr-o sala deja ocupata, etc…). Pentru fiecare profesor ce nu vrea sa predea in intervalul respective cresc numarul de conflicte ale orarului cu 1. Tot aici tratez si bonusul pentru care verific ca diferenta dintre intervalul curent pe care-l voi adauga si ultimul interval adaugat in lista de intervale ale profesorului gasit pentru ziua respective este mai mare decat pauza pe care acesta vrea sa o aibe maxim, caz in care adaug numarul de ore la conflicte. La final functia intoarce toate starile posibile prin care putea fi acoperita materia la intervalul si ziua precizata.
- ***get_next_states(discipline),*** care intoarce toate starile urmatoare posibile ce pot rezulta din cea curenta, pana in momentul in care materia data ca parametru este acoperita(moment marcat de faptul ca numarul de studenti pentru materia respectiva e mai mic sau egal cu 0).
- ***f(discipline),*** ce reprezinta functia de cost a starii curente pentru o anumita materie(f = g + h). Am considerat ca functie ***g*** numarul de studenti ce raman pentru acoperirea materiei respective si ca euristica initial am considerat numarul de conflicte al starii curente, dar nu aveam rezultate corecte din cauza ordinelor diferite de marime dintre conflicte si numarul de studenti necesari pentru acoperire. Ca solutie m-am gandit sa folosesc niste factori de balansare prin care sa echilibrez ordinele de marime ale

celor 2 componente(adica sa inmultesc numarul de conflicte cu 10**(ordin_de_marime_g – ordin_de_marime_h) deoarece astfel le aducem la ordinul zecilor, sutelor, miilor, etc… pe ambele) dar nici asa nu generam solutie buna pentru constrans de fiecare data si nici pentru mic exact si am vazut ca daca inmultesc cu 14(dupa ce am incercat cu 15 si am vazut ca nu merge) va fi generata solutie buna de fiecare data(fara conflicte hard).

2. Reprezentarea constrangerilor problemei

1. Am reprezentat constrangerile problemei sub forma unui dictionar in care am pus ca si chei profesorii si ca valori un dictionar ce contine constrangerile profesorilor referitoare la materii, zile, intervale favorabile si nefavorabile pentru acestia si pauza pe care fiecare o are.

3. Hill Climbing

1. Pentru implementarea algoritmului de **Hill Climbing** am folosit un **heap** in care am adaugat materiile in functie de numarul de sali in care acestea se pot preda, deci am facut acoperirea materiilor incepand de la materia cu cele mai putine sali disponibile(am observat ca astfel obtin solutie buna de fiecare data si de asemenea imi sunt acoperite toate materiile pe orarul constrans, obtinand doar constrangeri soft incalcate).
2. Am inceput acoperirea fiecarei materii utilizand algoritmul clasic de **Hill Climbing** in care am folosit ca functie de cost functia ***f*** din clasa **State** pentru determinarea starii ce duce la o solutie fara constrangeri. Optimizarea adusa algoritmului o reprezinta faptul ca nu generez toate starile urmatoare posibile pentru problema, ci incerc sa generez doar starile care indeplinesc conditiile hard si pe cele soft legate de zile.

4. *Monte Carlo Tree Search*

1. Pentru **Monte Carlo** am utilizat varianta implementata la laborator unde etapa de **selectie** se face pana in momentul in care ajung la o stare finala care reprezinta faptul ca o materie a fost acoperita(adica nod frunza), etapa de **exapandare** se face doar in momentul in care am ajuns la frunza si aleg dupa o distributie **softmax** urmatoarea stare in care ne ducem si pentru care generam o actiune noua, etapa de **simulare** se face tot dupa o distributie **softmax** pornind din nodul nou adaugat in arborele de stari, iar etapa de **backpropagation** se face pe toate nodurile in sus in arbore pana la radacina, adaugand de fiecare data un reward reprezentat de functia -***f*** in momentul in care gasim o stare finala(conflicte + toate materiile acoperite), -***f / 10*** (in momentul in care am acoperit o materie total) si **f** cand nu se indeplineste niciuna din conditiile anterioare.
2. Ca optimizare fata de laborator am utilizat distributia **softmax** in loc de **choice** si am taiat din actiuni pentru noduri pentru care **node[N] < 10**, deoarece astfel sunt acoperite mai putine stari si facut mai putine simulari pentru problema.

5. *Diferente intre cei doi algoritmi*

1. *Timpi de rulare*

i. In urma rularii celor doi algoritmi am obtinut timpi mai buni pentru **Hill Climbing** decat pentru **Monte Carlo.**

ii. Acest lucru se datoreaza cel mai probabil faptului ca primul algoritm cauta mereu cea mai buna prima acoperire a materiei, dupa care continua pe ramurile cu cele mai putine conflicte pana gaseste o solutie finala valida, adica exploreaza oarecum spatiul starilor urmatoare de fiecare data, aspect ce ar garanta faptul ca de fiecare data obtinem aceleasi solutii, pe cand la

celalalt algoritm obtinem solutii diferite in timpi diferiti datorita faptului ca acesta isi alege random din copii, lucru ce poate genera stari ce nu merita neaparat explorate dar pe care deja le-am explorat.

iii. Timpii pentru Hill Climbing:
    1. Dummy: 0.034 seconds.
    2. Orar mic exact: 1.859 seconds.
    3. Orar mediu relaxat: 28.951 seconds.
    4. Orar mare relaxat: 67.644 seconds.
    5. Orar constrans incalcat: 5.468 seconds.
    6. Orar bonus exact: 71.650 seconds.

iv. Timpii pentru Monte Carlo:
    1. Dummy: 0.060 seconds.
    2. Orar mic exact: 11.444 seconds.
    3. Orar mediu relaxat: 202.336 seconds.
    4. Orar mare relaxat: 233.507 seconds.
    5. Orar constrans incalcat: 41.927 seconds.
    6. Orar bonus exact: 943.289 seconds.

2. Numar de stari construite

i. In cadrul primului algoritm numarul de stari construite este mai mic decat in cadrul celui de-al doilea algoritm din cauza numarului de stari construite in urma simularii cel mai probabil.

ii. Numar stari Hill Climbing:
    1. Dummy: 128
    2. Orar mic exact: 2272
    3. Orar mediu relaxat: 19123
    4. Orar mare relaxat: 18015
    5. Orar constrans incalcat: 6931
    6. Orar bonus exact: 26777

iii. Numar stari Monte Carlo:
    1. Dummy: 288
    2. Orar mic exact: 14877
    3. Orar mediu relaxat: 146226

4. Orar mare relaxat: 67339
5. Orar constrans incalcat: 41781
6. Orar bonus exact: 323311

3. Calitatea solutiei

i. Primul algoritm produce solutii cu mai putine conflicte decat Monte Carlo(cel putin pe orarele constrans si bonus). Datorita randomness-ului pe care Monte Carlo il are, nu este garantata generarea unei solutii cu 0 constrangeri soft incalcate la final.

ii. Numar conflicte Hill Climbing:

1. Dummy: 0
2. Orar mic exact: 0
3. Orar mediu relaxat: 0
4. Orar mare relaxat: 0
5. Orar constrans incalcat: 8
6. Orar bonus exact: 2

iii. Numar conflicte Monte Carlo:

1. Dummy: 0
2. Orar mic exact: 0
3. Orar mediu relaxat: 0
4. Orar mare relaxat: 0
5. Orar constrans incalcat: 7 soft, 1 hard(se obtin si solutii cu 0 hard)
6. Orar bonus exact: 3 soft, 1 hard (se obtin si solutii cu 0 hard si chiar si cu 0 soft).

6. Rulare

1. Tema se poate rula utilizand comanda:
   i. python3 orar.py algoritm input
   ii. unde **algoritm** poate fi **hc** sau **mcts,** iar input este de forma **inputs/file(**se ruleaza programul din directorul sursa al temei).