TEMA 1 IA

CAIM ALEXANDRU GABRIEL
334CA

Programul este constituit din implementarea celor doi algoritmi, fiecare având o clasă proprie.

➔ A*

Pentru algoritmul A* am construit un data class care prezintă câmpurile necesare reprezentării unei stări. Acestea se împart în 2 categorii:

- câmpuri care identifică starea (nume materie, nume profesor, sala, intervalul orar, ziua)
- câmpuri care ajută la calcularea euristicii și la procesul de generare de noi stări din cea curentă

Euristica se bazează pe un calcul polinomial, în care coeficienții polinomului sunt rezultați din câmpurile stării iar ‘x’ este 10. Avem deci puterile lui 10 de la 0 până la 4. Deoarece alegerea viitoarei stări are la bază găsirea unei valori minime pentru această euristică am considerat coeficientul lui BIG_CONS ca fiind cel mai important factor în această alegere și anume materiile rămase. Trebuie să acoperim toate materiile, aceasta fiind o cerință hard. Coeficientul lui SMALL_CONS este dat de numărul de restricții soft încălcate, iar restul polinomului ține cont de restul aspectelor (numărul de intervale disponibile, în câte intervale a predat fiecare
profesor). Dacă se ajunge la 8 intervale la un profesor, încălcăm o constrângere hard, deci INVALID_VAL va trebui sa fie o valoare cât mai mare. Am ales să păstrez fiecare coeficient în intervalul (0, 9) pentru a separa clar prioritățile și a urmării mai ușor evoluția algoritmului. Din acest motiv, niciodată numărul de studenți rămași la o materie nu va valora mai mult decât materiile rămase ca exemplu.

Algoritmul pornește dintr-o stare inițială invalidă ca și atribute, nu definește o sală, un profesor sau un interval. Aceasta alege doar prima materie din listă și ini-
țializează restul câmpurilor.

În etapa de generare de stări noi căutăm toate stările posibile care să nu fie

în conflict cu cele deja existente, adică se caută evitarea încălcarii constrângerilor hard. Pentru a putea lucra mai usor, în next_states generăm doar primul tip de câmpuri ale unei stări, sub forma unui tuplu. Această listă este mai apoi completată cu restul câmpurilor pe baza stării curente și a tuplului în update.

Funcția Astar are o implementare similară cu cea din laborator, având o frontieră implementată ca heap și o listă de noduri descoperite. Diferența față de versiunea din laborator este absența lui 'f' deoarece în abordarea mea, distanța de la starea inițială la cea curentă este mereu aceeași (numărul de slot-uri ocupate), deci folosirea ei nu este tocmai utilă. După ce ajungem într-o stare finală reconstruim traseul până la starea inițială cu ajutorul câmpului parent.

➔ Hill-Climbing

Pentru algoritmul Hill-Climbing am schimbat puțin abordarea. Fiind un algoritm local, am încercat să plec cu o stare random ca stare inițială și am încercat să reduc cât mai mult posibil conflictele hard și soft.

Stările sunt reprezentate de această dată sub forma unui timetable(un dicționar ca cel necesar pentru afișare).

În create_random_tt generăm un timetable random în care căutăm să acoperim toate materiile. Pentru fiecare materie, alegem o sală unde se poate preda materia, un profesor care o poate preda, un interval și o zi. Dacă în acel interval orar în sala respectivă se mai desfășoară altă materie, atunci alegem altă combinație. Este important de precizat că datorită alegerii random a unor atribute numărul de stări, timpul și alte măsurători diferă de la orar la orar.

Primul pas este găsirea slot-ului care încalcă cele mai multe constrângeri. Avem o euristică în acest sens pentru fiecare slot, care este utilizată și în calcularea euristicii pentru întreaga stare/timetable. Calcularea ei are similarități cu obținerea restricțiilor soft încălcate de la A*. După ce am găsit slot-ul generăm stările următoare, stări care se obțin prin următoarele 3 'mișcari':

- schimbare profesor cu altul care predă aceeași materie
- mutarea în alt interval orar, zi și sală libere, astfel încât să nu încălcăm restricțiile hard
- schimbul de intervale cu alt profesor care predă în aceeași sală

Fiecare astfel de mișcare generează o stare, care este adăugată într-o listă. În funcția hill-climbing setăm un maxim de iterații și alegem la fiecare pas starea cu cele mai puține restricții încălcate. Este un algoritm Hill-Climbing simplu, deoarece am considerat că este cea mai bună variantă dintre cele prezentate la laborator.

NUME TEST	A* TIMP	A* STĂRI	HC TIMP	HC STĂRI	HC RES
dummy	0.009	11	0.004	5	0
mic_exact	3.97	34	0.19	28	0
mediu_relax	107	50	1.07	56	0
mare_relax	177	51	3.12	100	6
cons_incalcat	>21600	>5000	0.96	100	9
bonus	612	151	4.25	100	20

Timpul este măsurat în secunde.

Stările sunt cele parcurse, nu doar generate.

Am rulat mai multe teste și în tabel am trecut cele mai bune performanțe obținute de fiecare algoritm. Am considerat maximul de iterații 100 în cazul HC.

Pentru testul constrans_incalcat la astar am renunțat la execuție după 6 ore. Pentru restul testelor s-a obținut constant o soluție de cost 0, însă ca și timp HC este clar superior, iar la unele teste a atins soluția de cost 0.

În cazul lui constrans_incalcat la Astar execuția durează atât de mult deoarece se încearcă găsirea soluției de cost 0, dacă costrângerile soft aveau o prioritate mai mică în acel polinom, am fi obținut o soluție rapidă de cost foarte mare.

Deși A* nu se bazează în mare pe alegeri random, în alegerea stărilor mai apar stări care au aceeași euristică, mai ales la început, care sunt diferențiate de un parametru random. Astfel, la teste precum mic_exact am obținut soluția de cost 0 și în 4 secunde (cazul ce mai des întâlnit), dar și în aprox 10 - 12 minute.