Stefan Alexia
332CB

Ma folosesc de clasele: Room, Teacher, Class, SchedulerTimeTable, ClassSchedule.
In ClassSchedule am implementata logica de ‘State’ (constrangerile in self.all_Constraints, frecventa profesorilor in self.teachers_freq, in self.classes am o lista de obiecte de tip Class, obiecte ce se caracterizeaza prin liste de profesori care predau materia, de sali in care poate fi tinuta materia, nume si numar de student).

Functiile hard_schedule_hc si hard_schedule (sunt la fel, doar ca una a fost conceputa initial pentru algoritmul csp, iar pentru cel de hc unde creez multe stari successor ale unei stari curente apareau conflicte cu self.available_time_slots si a trebuit sa ma folosesc de copii pentru hard_schedule_hc) intorc un orar care respecta doar constrangerile hard.

- Pentru a construe starea initiala, sortez materiile dupa numarul de sali in care se pot tine
- Iau fiecare materie in parte, timp de un anumit interval (interval care corespunde la cate slot-uri ar fi daca sala cu cea mai mica capacitate ar fi disponibila)
- Imi generez un ‘entry’ in orar random, cu o zi si o ora random si verific ca in sala din entry sa se poate tina cursul
- Aleg un profesor random dintre cei care tin cursul, tinand cont de frecventa lui

1. Partea de CSP

- In functia my_min_conflicts(), pornesc cu o stare initiala (un orar care rescpecta doar constrangerile hard) si apelez next_state_mc(), unde calculez numarul de constrangeri hard, soft si in variabila crossed_neg_constraints imi intorc constrangerile soft inculcate (Acestea sunt intr-un dictionar de forma’ ('Alexandru Dumitrescu', ('Marti', '(8, 10)')): {'zi': ['Marti'], 'ora': []}’, unde key-ul ne spune numele profesorului si slotul orar in care are or ce nu respecta constrangerea, iar value-ul este un dictionar cu doua intrari zi si ora; in acest exemplu deoarece value-ul zilei este ‘Marti’ iar in cel al orei nu este nimic inseamna ca ziua nu respecta constrangerile soft).
- Din lista de constrangeri soft inculcate aleg random una.
- Apelez solve_constraint() ce primeste o un orar si o constrangere pe care incearca sa o resolve: ori incerc sa mut profesorul care are constrangerea intr-un slot de al lui preferat (search_available_slots), ori caut un alt profesor (care preda materia si care are frecventa < 7) care sa aiba printre preferintele lui slotul curent din constrangere. Daca nu se gaseste nici asa, atunci caut un slot liber care sa fie in preferintele profesorului nou gasit.
- Astfel, revin in next_state_mc cu noul orar care, teoretic are cu o constrangere soft in minus, verific daca chiar are nr de constrangeri soft mai bun si daca da, devine noua stare curenta
- Fac asta tip de un numar iteratii, dupa care reincep cu o noua stare initiala (un fel de random restart)
- In lista best_options() retin cele mai bune stari (care sa aiba sub 5 constrangeri soft), pe care incerc intr-un loop mai mic sa le rezolv din nou
- Daca nu reusesc, intorc orarul cu cele mai putine constrangeri din lista

- Calitatea solutiei:
    - Dummy: 5/5 teste cu cost 0
    - Orar mic: 4/5 teste cu cost 0
    - Orar mediu: 4-5/5 teste cu cost 0 (am observant ca merge putin mai bine decat la orar mic, surpinzator)
    - Orar mare: 2/5 teste cu cost 0, restul cu cost sub 5
    - Orar constrans: se blocheaza (la generarea de orare initiale, sunt anumite cazuri in care se intampla ca pe toate sloturile posibile sa fie o materie Y care s a assignat deja intr-o sala, si atunci cand vine o materie care se poate tine doar in acea sala de curs, nu mai are efectiv unde sa se puna => cicleaza; aceasta problema se intampla doar la acest test, am incercat sa iau materiile dupa nr lor de Sali, dar tot nu s a rezolvat aceasta problema)
- Consider aceasta abordare una de CSP, deoarce spre deosebire de HC, incerc sa rezolv constrangerile soft ale starii curente si sa analizez variante mai bune de mutari ale unui slot in orar si nu doar generez succesori fara a analiza calitatea lor, algoritmul fiind "conscious" de schimbarea pe care o face.
- Cele doua loop-uri din functia my_min_conflicts() executa 50*25= 1250 de pasi, in cel mai rau caz (+ inca 25 pentru best_option), cu toate acestea este mult mai rapid decat HC

2. Partea de HC

- HC cu random restarts (in my_hill_climbing())
- Generez o stare initiala, apelez apoi get_next_states() pentru succesori
- In get_next_states() se ia fiecare entry din orarul curent, si se apeleaza generate_succesors_hc()
- In generate_succesors_hc() se primeste orarul curent si un entry (de forma {'day': 'Luni', 'time': (8, 10), 'teacher': 'Alexia Stefan', 'room': 'PR03', 'class': 'MS'})
- Incerc sa caut toate sloturile libere din orar unde as putea muta (pe acest exemplu) ora de MS, tinuta de 'Alexia Stefan' in sala 'PR03' (sau oricare alta sala unde se poate tine 'MS')
- Astfel, in get_next_states() pentru fiecare entry al starii curente se intorc alte variante de orar, fara a se analiza calitatea succesorilor (functia este 'unconscious' din acest pct de vedere)
- Aleg sa ordonez succesorii dupa nr de constrangeri soft incalate
- Apoi se face comparatia clasica intre nr de conflicte, inapoi in functia my_hill_climbing().
- Retin cea mai buna optiune dintre toate cele analizate in best_choice
- Calitatea solutiei: mult mai scazuta comparative cu CSP-ul, in cazul testului de dummy intorcandu-mi in cel mai bun caz o stare de cost intre 1 si 5, iar in cazul celorlalte teste intoarce o stare cu 20 constrangeri mai putine fata de starea initiala, dar dureaza extrem de mult.
- Cred ca motivul pentru care starea finala nu are un cost foarte mic (aproape 0) este din cauza starii initiale, care a fost conceputa sa nu ia in considerare absolut nicio constrangere soft si cu multe elemente de randomness, astfel incat starea initiala are un nr de constrangeri soft foarte mare. Totodata, mereu functia nu ajunge sa isi faca

numarul total de max_iters, deoarece ajunge de foarte multe ori la un maxim local, motiv pentru care trebuie sa se ia de la inceput totul.

- Se fac 10*10 = 100 de iteratii in cel mai rau caz in functia de HC +++ generarile de succesori – care depind de nr de intrari din orarul initial si care maresc numarul de pasi extrem de mult
- Desi un numar mai mic in cele doua loop-uri din functia de HC, timpul de executie se observa ca este cu extrem de mult mai mare decat la CSP, din cauza generarii de vecini.