Anton Stefan, 331CC

TEMA1 – Inteligenta Artificiala - Orar

Tema isi propune reprezentarea unui orar folosind 2 algortimiti de cautare Hill Climbing si

Monte Carlo Tree Search.

Doar partea de hill climbing este implementata.

Instructiuni de rulare:

time python3 orar.py hc inputs/orar_bonus_exact.yaml

si trebuie schimbat la output_path linia 299 ‘outputs/orar_bonus_exact.txt’

conform tesutului in cauza pentru a putea verifica cu

python3 check_constraints.py orar_bonus_exact, checkerul local

Durata implementare:

18~ ore

Detalii implementare:

Am un state care contine fisierul yaml parsat, si pe langa am mai adaugat pentru ‘professors’ ‘scheduled hours’ care tine cont de cate ore are un professor assignate, ‘schedule’ care va tine orarul propriu zis afisat cu pretty print, si ‘student_needs’ care tine legatura dintre cursuri si numarul de studenti care au ramas de assignat pentru acel curs. De asemenea ‘constraints’ din professors a fost preprocesat pentru a transforma din intervale de genul 8-12 in intervale separate de cate exact 2 ore, 8-10 si 10 -12 pentru usurinta in parcurgere. Mai am functia check_soft_ constraints_cost si check_hard_constraints_cost care returneaza costul unui orar dat.

Am initializat costul hard foarte mare, 5* numarul de studenti neasignati pentru a se putea modifica dinamic, in timp ce aloc studenti o sa am un cost mai mic si o sa fie favorabil de ales in acea directie.

Pentru fiecare cost soft am pus 1 si pentru celelalte hard am pus 10 * numarul de constrangeri, pentru a putea evita incalcarea celor hard.

Flowul programului:

Se porneste din hill climbing singura optiune de algoritm implementata. In hill climbing avem current_state care se initializeaza cu functia make_initial_assignment, aceasta functie intoarce un orar cu o singura materie assignata cu un professor random din posibilitatile date, care sa respecte soft si hard. Din cauza ca avem cost mare pentru studentii neasingati plecam cu un initial nefavorabil. Dupa luam costul cu combined cost al starii respective si intram in while si verificam cat timp mai putem face iteratii ne uitam daca toti stundetii au fost assignati, daca da putem spune ca orarul este complet, daca nu, generam toate posibilitatile de orare cu inca o stare din starea curenta adica daca avem 1 stare vom genera toate posibilitatile de orare cu 2 stari in lista new_states din get_all_possible_states si dupa le vom returna. In functia get_all_possible_states de asemenea am sortat camerele descendend dupa numarul de studenti, si am mai sortat cursurile dupa numarul de sali disponibile pentru acesta. Alegem ca stare urmatoare starea cu cost minim, urmatorul cost devine costul urmatorii stari, si dupa verificam daca starea urmatoare in care am ajuns are costul mai mic decat best_cost de la care am plecat, daca are, best_state devine urmatoarea stare, best_cost devine urmatorul cost, si continuam cautarea din starea next. La final returnam starea gasita dupa ce s-a indeplinit conditia sa nu mai avem studenti neasignati sau sa nu mai avem un next states valid.

Testele trec toate constrangerile hard, incalcand un numar de cateva constrangeri soft la unele teste. Implementerea e eficienta si are un trade-off mic cost vs timp, puteam sa dau random restart si sa aleg urmatoarele posibilitati random pana gasea o solutie valida da asta ar fi dus la un timp foarte mare. Tinand cont de problema propusa pot sa spun ca am obtinut rezultate favorivabile, deoarece si in viata de zi cu zi se mai intampla sa fie niste studenti intr-un orar neplacut, sau unii dintre profesori sa predea la niste ore inconveniente, si cateva constrangeri soft e real pentru un numar mare de variabile.

Pot sa spun ca optimizari fata de varianta din laborator pe care le-am adus sunt sortarile specifice pentru problema propusa, are un numar de stari construite favorabil, nu fac random restart si random de fiecare data, ci aleg la inceput o stare random de professor care sa respecte atat hard si soft din toate posibilitatile pe care le am si dupa aleg sa continui iterativ, alegand cea mai buna solutie de fiecare data cand gasesc un cost mai mic.

Trec toate testele inclusiv cel de bonus, care e rezolvat automat prin euristica descrisa fara a fi necesar de conditii implicite, datorita manierei alese a problemei.

La rularea manuala se poate vedea la final printul orarului cat si numarul de iteratii sau costul.

Timpi / cost hard,soft/ iteratii

dummy 0m0.321s / 0,0 / 11

orar_bonus_exact 1m24.269s / 0,2 / 103 , in unele cazuri mai da 0,3 variatii foarte mici din cauza alegerii random a profesorului initial

orar_constrans_incalcat 0m35.293s / 0,6 / 54

orar_mare_relaxat 3m18.174s / 0,1 / 51

orar_mediu_relaxat 0m41.073s / 0,2 / 47

orar_mic_exact 0m10.868s / 0,3 / 33

Durata orarului mare fiind de doar 3minute pot spune ca este un trade-off bun costul cu durata.