Inteligenta Artificiala: Tema 1 – Orar

ANA Elena-Diana, 331CB

1. Reprezentarea starilor
   1. Algoritmul Hill Climbing

Pentru a reprezenta starile am imprementat o clasa ‘TimetableState’. Aceasta contine:

* informatii extrase din fisierul de intrare (intervale, materii, profesori, zile, sali de curs) sub forma de dictionare stucturate exact ca in fisier;
* un dictionar de forma: ‘nume profesor : zi a saptamanii : lista de intervale’ pentru a retine programul profesorilor;
* un dictionar de forma ‘materie : valoare’ unde valoarea reprezinta numarul de studenti nerepartizati pentru materia respectiva;
* o varianta curenta a orarului, reprezentat printr-un dictionar cu structura din enunt.

1. Reprezentarea restrictiilor hard
   1. Algoritmul Hill Climbing

Pentru rezolvarea acestei teme, am ales sa generez o stare initiala in care asigur acoperirea tuturor materiilor, precum si satisfacerea tututror celorlalte restrictii hard. Apoi aplic algoritmul HC pentru a genera o solutie de cost minim dpdv restrictii soft, totodata asigurandu-ma ca nu compromit satisfacerea restrictiilor hard.

Asadar o parte din restrictiile hard sunt verificate explicit cu functia ‘check\_hard\_constraints’, unde ma folosesc de dictionarul ‘professors\_programs’ (verific 2 constrangeri). Celelalte sunt verificate implicit in functiile ‘generate\_timetable’ si ‘get\_next\_states’ pentru a optimiza procesul de generare a vecinilor (de exemplu, cand incerc sa pun/mut o materie intr-un interval si parcurg toti profesorii, voi merge mai departe doar cu cei care predau materia respectiva)

1. Reprezentarea restrictiilor soft
   1. Algoritmul Hill Climbing

Restrictiile soft sunt verificate cu ajutorul dictionarului ‘professors’ de unde preiau preferintele profesorilor si dictionarului ‘professors\_programs’ unde retin programul curent pentru fiecare dintre acestia.

In functia ‘count\_soft\_constraints’ calculez numarul de constrangeri soft incalcate de starea curenta, comparand informatiile din cele 2 dictionare mentionate.

In functia ‘get\_next\_states’ verifc direct daca o mutare incalca sau nu o constrangere soft, de asemenea, prin compararea celor 2 dictionare.

1. Optimizari pentru HC

* Generez doar vecini valizi. Functia ‘apply\_move’ nu va face mutarea orei daca acest lucru incalca constrangeri hard
* Generez doar vecinii despre care stiu ca vor reduce costul. Concret, in functia ‘get\_next\_states’ caut acele ore din solutia curenta care incalca minim o constrangere si incerc sa le mut intr-un interval care nu va genera nicio noua constrangere.
* In crearea starii initiale, prioritizez salile mari si profesorii care au putine materii de predat. De asemenea, tin cont de unele constrangeri soft pentru a incepe HC cu un cost initial cat mai mic. Ulterior ma asigur ca repartizez eventualii studenti ramasi, indiferent de constrangerile soft incalcate.

1. Concluzii HC

* Varianta de HC pe care am folosit-o este Stochastic Hill Climbing. Am ales-o deoarece, implementand varianta standard a algoritmului in tema mea, am realizat ca riscam sa ma plimb intre 2 stari.
* Un trade-off al folosirii acestei variante este variatia output-ului la rulari diferite; prin urmare, sunt necesare mai multe rulari pana gaseste o solutie de cost 0. De asemenea, faptul ca nu am folosit o metoda de random restart face mai greu de gasit solutii de cost 0 pe unele inputuri – de exemplu pe ‘orar\_constrans\_incalcat’ unde imi gaseste solutii cu cost mediu = 15.
* Generarea vecinilor este o operatie destul costisitoare dpdv timp. Am incercat sa optimizez acest proces prin mutarea exclusiv a orelor care incalca constrangeri. In logica mea, o stare cu acelasi numar de conflicte ca starea parinte poate conduce la o solutie mai buna (este cazul in care un profesor NU prefera o zi dar are mai multe intervale in acea zi, iar eu mut unul din intervale => costul nu se schimba dar poate duce la o solutie mai buna). Asadar, daca las comparatia din HC intre starea curenta si vecini sa accepte si vecinii de acelasi cost, cresc posibilitatea gasirii unei solutii de cost 0, dar cresc de asemenea si timpul de executie.
* Timpul total de rulare a testelor nu depaseste un minut.
* Algoritmul gaseste solutii de cost 0 pe toate testele, mai putin ‘orar\_constrans\_incalcat’.