Tema 1 IA
Boiangiu Victor-Miron
332 CD

A*

- Reprezentare stari:

Pentru A*, am reprezentat starile ca in fisierul utils.py, ca dictionare de zile, intervale si sali, cu valorile finale tupluri cu profesori si materii.

- Restrictii:

O parte din restrictiile hard sunt respectate din structura starii: nu se poate ca o sala sa fie ocupata de 2 profesori in acelasi interval sau un profesor sa predea 2 materii in aceeasi sala, etc.

Restrictia ca un profesor nu preda in 2 sali diferite in acelasi interval este respectata selectand doar profesorii liberi intr-un anumit interval atunci cand sunt create starile vecine. Asemenea sunt filtrate si salile la crearea vecinilor pentru a selecta doar salile unde materia dorita se poate preda.

Acoperirea materiilor si evitarea suprasolicitarii profesorilor se fac memorand dictionare cu aceste informatii in fiecare stare si verificarea acestora la crearea vecinilor.

Restrictiile soft incalcate sunt calculate la crearea unei noi stari vecine adaugand la numarul de restrictii soft incalcate de parinte restrictiile soft incalcate de noua intrare adaugata in orar, evitand astfel calculul tuturor la crearea starilor vecine.

- Euristica:

Euristica aproximeaza numarul de intrari pe care il va avea orarul atunci cand va acoperi toate materiile.

In mod intentionat, euristica nu este admisibila, caci nu vrem sa gasim orarul cu numarul minim de intrari ce acopera materiile, functia h() supraestimand numarul de intrari ramase necesar acoperirii tuturor materiilor. Astfel, euristica favorizeaza selectarea starilor cu mai multe intrari, aka. explorarea deep a starilor.

g(stare) = numarul de intrari din stare
h(stare) = nr. de studenti ramasi nealocati / (capacitatea celei mai mici sali / 5)
+ 2 * numarul de constrangeri weak incalcate.

- Optimizari:

Atunci cand sunt create starile vecine, verificam doar pentru noua intrare in orar daca incalca vreo constrangere hard si cate constrangeri soft incalcate adauga, evitand parcurgerea tuturor acestora.

La crearea vecinilor, sunt selectati direct doar profesorii liberi din acel interval si doar salile in care se poate preda materia aleasa.

Sunt generati doar vecinii care adauga o intrare pentru o anumita materie selectata, materiile fiind ordonate dupa urmatoarea formula in ordine crescatoare si trecandu-se la urmatoarea materie atunci cand anterioara este acoperita.

Pentru o materie M:

$$\frac{\sum_{Sala\ S\ ce\ accepta\ materia\ M} \frac{1}{Nr.\ de\ materii\ ale\ lui\ S}}{Numarul\ salilor} + 0.25 * \frac{\sum_{Prof\ P\ ce\ preda\ materia\ M} \frac{1}{Nr.\ de\ materii\ ale\ lui\ P}}{Numarul\ profesorilor}$$

PCSP

- Reprezentare stari:

Am ales ca o solutie sa fie de forma unui dictionar ce mapeaza tupluri cu structura (zi, interval, sala) la o intrare de forma (profesor, materie).

- Restrictii:

Ca sa fie *frumos* si ca sa se potriveasca cu ce structura avea laboratorul (caci am plecat de la el), am modelat restrictiile soft sub forma tuplurilor cu ce variabile contin si functia ce verifica respectarea acestora.

Restrictia hard ca intr-o sala sa nu fie mai multe materii sau mai multi profi in acelasi timp este respectata de structura solutiei.

Restrictia hard ca materiile sa fie acoperite este verificata printr-un dictionar ce mapeaza materiile la cate locuri le mai sunt necesare, dictionar dat ca parametru apelului recursiv. Asemenea se respecta si constrangerea referitoare la suprasolicitarea profesorilor.

Restrictia ca intr-o sala sa se predea doar materiile compatibile cu aceasta si restrictia ca un profesor sa predea doar materiile sale sunt ambele respectate, domeniile variabilelor fiind construite doar cu valori ce respecta aceste doua constrangeri.

- Optimizari:

Domeniile sunt generate de la inceput pentru a respecta automat o parte din restrictiile hard, asa cum este mentionat mai sus.

Domeniilor variabilelor sunt ordonate astfel incat intrarile sa respecte ordinea de prioritate a materiilor, ordine calculata dupa formula mentionata la A*. In cazul in care doua intrari se refera la aceeasi materie, se face compararea satisfactiei profesorului cu acel interval, intrarile unde profesorii sunt satisfacuti avand prioritate.

Variabilele sunt si ele luate in algoritmul de backtracking intr-o anumita ordine, facand intai umplerea salilor cu capacitatile cele mai mari.

Constrangerile soft sunt calculate luand costul anterior si adaugandu-i acestuia doar constrangerile soft incalcate de noua intrare.

Odata ce o materie este acoperita, sunt scoase din toate domeniile valorile ce contin acea materie.

Odata ce un profesor ocupa un interval, este scos din toate domeniile care fac referire la acelasi interval, dar alta sala.

Odata ce un profesor isi ocupa toate slot-urile, este scos din toate domeniile.

Pentru ca altfel ar dura excesiv de mult executia, exista un parametru *CSP_DELTA_ITERATIONS_UNTIL_TIMEOUT* ce actioneaza ca un prag: daca de la gasirea ultimei solutii considerata cea mai buna de pana atunci au trecut atatea iteratii fara gasirea altei solutii mai bune, algoritmul se opreste.

Rezultatele rularii algoritmilor

	A*			PCSP		
Nume test	Timp	Constrangeri soft incalcate	Stari generate	Timp	Constrangeri soft incalcate	Stari generate
dummy	0m0,050s	0	188	0m0,055s	0	67
mic exact	0m0,310s	0	5690	0m0,364s	0	432
mediu relaxat	0m1,778s	0	29846	0m0,404s	0	88
mare relaxat	0m2,589s	0	39579	0m5,904s	0	1553
constrans incalcat	0m0,747s	15	13926	6m46,553s	8	517692

Desi din aceste teste nu este vizibil acest aspect decat din ultimul test, PCSP tinde sa gaseasca solutii mai bune decat A*, intrucat acesta nu se opreste la gasirea primei solutii, ci continua sa caute pana gaseste una de cost 0, dar backtracking-ul poate fi costisitor, asa cum se vede din timpul de executie al acelui test. Algoritmul s-a oprit intrucat am introdus un prag de iteratii de la gasirea ultimei solutii considerata cea mai buna pana atunci, altfel ar fi continuat sa caute pana la gasirea unei solutii de cost 0 sau epuizarea variantelor.

Algoritmii fiind bine optimizati si testele mici, ambii algoritmi au convers repede pe primele teste la solutii de cost 0, dar totusi se observa cum A* gaseste mai repede o solutie decat PCSP, dar este mai costisitor din punct de vedere al memoriei.