IA - Tema1

DUDU Matei-Ioan: 333CB

Cuprins

1. Reprezentarea starilor si a restrictiilor

Pentru aceasta parte a implementarii temei am creat un fisier denumit *structures.py* in care am realizat reprezentarea:

- intervalelor
- materiilor
- profesorilor
- salilor
- starii curente a orarului pentru implementarea Hill-Climbing

Dupa cum se poate observa, primele patru clase sunt folosite de ambii algoritmi, iar a cincea (clasa State) este folosita doar de algoritmul Hill-Climbing (exceptie face doar situatia in care CSP instantiaza o stare doar pentru a calcula numarul de restrictii hard si soft incalcate pentru rezultatul final)

Pentru algoritmul Hill-Climbing, plec de la un orar gol si il completez pe masura ce aleg vecinul urmator. Prin urmare, un State care nu primeste un orar la initializare va crea un orar gol.

Pentru PCSP aleg ca multime de valorile pentru variabile doar perechile de (profesor, materie) care satisfac conditia ca sala sa accepte materia si profesorul sa poata preda materia respectiva. Astfel ajung sa reduc considerabil numarul de valori.

2. Workflow-ul temei

Fisierul Python care este apelat pentru aceasta tema este cel specificat si in cerinta temei: orar.py.

Se poate observa felul in care functioneaza orar.py in fragmentul de cod de mai jos:

```
n = len(sys.argv)

# verific ca numarul de argumente sa fie cel corect
if n != 3:
    print('Usage: python3 orar.py <method> <input_file>')
    exit(1)

# extrag algoritmul ce se vrea a fi folosit si fisierul de intrare
method = sys.argv[1]
input_file = sys.argv[2]

# citesc specificatiile orarului din fisierul de intrare si apelez exit in
cazul in care acesta nu este unul valid
try:
    timetable_specs = utils.read_yaml_file(input_file)
except Exception as e:
    print(e)
    exit(1)

# verific ca metoda specificata sa fie una existenta
if method == 'hc':
    hca.hca_main(timetable_specs, input_file)
elif method == 'csp':
    csp.csp_main(timetable_specs, input_file)
else:
    print('Invalid method')
    print('The available methods are: hc and csp')
    exit(1)
```

Pentru rezolvarea problemei Hill-Climbing am creat fisierul hca.py, iar pentru CSP am creat csp.py (fiecare problema cu main-ul sau). In orar.py doar este stabilit main-ul care trebuie apelat, restul problemei fiind rezolvata in unul dintre cele doua fisiere.

3. Optimizari fata de laborator

3.1 Optimizari aduse algoritmului Hill-Climbing

Pentru acest algoritm m-am folosit de varianta standard de HC, in care aleg drept stare urmatoare starea vecina cu cel mai bun cost.

O stare **a** este mai buna decat o stare **b** daca **a** are mai putine constrangeri hard incalcate decat **b**. In caz de egalitate, se compara constrangerile soft incalcate:

```
if neigh_hard_conflicts < hard_conflicts or (neigh_hard_conflicts ==
hard_conflicts and neigh_soft_conflicts <= soft_conflicts):
```

Pentru a imbunatatii algoritmul, am venit cu urmatoarele euristici:

- sortez salile crescator dupa numarul de materii care pot fi tinute in acestea si descrescator dupa capacitatea lor
- sortez profesorii crescator dupa numarul de materii pe care le pot tine
- sortez materiile crescator dupa numarul de profesori care pot preda materia respectiva si descrescator dupa numarul de studenti atribuiti acestora
- calcularea constrangerilor se face cu o singura parcurgere a orarului, folosindu-ma de diferite structuri (liste, dictionare, set-uri) pentru a contoriza anumite restrictii
- pentru fiecare profesor mai intai verific zilele si intervalele preferate, dupa care pe cele care ar incalca restrictiile soft

3.2 Optimizari aduse algoritmului CSP

Pentru acest algoritm am preluat functia PCSP din laboratorul aferent si am modificat-o astfel:

- Pentru a evita copierile domeniului cu deepcopy la fiecare apel recursiv pentru urmatoarea variabila am ales sa iterez prin fiecare valoare a variabilei cu ajutorul unui for
- De asemenea, acest for optimizeaza si adancimea la care se ajunge in stiva
- Ca sa evit slice-urile am decis sa mai adaug un parametru functiei recursive care sa reprezinte variabila curenta
- Modelarea constrangerilor hard presupune verificarea la fiecare pas in care adaug o valoare noua daca respecta constrangerile, iar daca nu, trec la urmatoarea valoare. Astfel tai mai multe ramuri din parcurgere fara a le verifica mai departe
- Pentru constrangerile soft verific pentru profesorul curent daca are in setul sau de constrangeri ziua si intervalul curent. Rezultatul acestei verificari il adaug la costul de pana in acel moment

4. Comparatia celor doi algoritmi

4.1 Dummy

Pe dummy consider ca nu sunt prea multe idei de preluat pentru ca este un test cu putine date.

Algoritm	Constrangeri hard	Constrangeri soft	Stari generate/iteratii	Timp de executie
HC	0	0	367	0.048 s
PCSP	0	0	68	0.031 s

4.2 Orar mic exact

Pentru orar mic exact deja incep sa iasa in evidenta anumite aspecte.

Din punct de vedere al timpului de executie cele doua metode sunt mai mult sau mai putin egale, dar cand comparam numarul de stari generate la numarul de iteratii se poate observa faptul ca generarea starilor este mult mai costisitoare decat iterarea de la o configuratie la alta in backtracking.

Acest lucru se poate datora faptului ca la fiecare generarea a unei stari se calculeaza si numarul de conflicte hard si soft.

Algoritm	Constrangeri hard	Constrangeri soft	Stari generate/iteratii	Timp de executie
HC	0	0	14,359	2.210 s
PCSP	0	0	3,198,719	2.409 s

4.3 Orar mediu relaxat

Aici, fata de orarul mic, numarul de stari generate este mai ridicat, iar cel de iteratii (pentru CSP) este mai scazut.

Presupunerea mea este ca ceea ce face ca algoritmul PCSP sa fie mai rapid decat HC in acest test este fix faptul ca orarul este "relaxat". Acest fapt permite existenta mai multor solutii finale raportate la numarul total de solutii la care ar putea ajunge PCSP.

Ceea ce ingreuneaza HC este ca la fiecare pas trebuie sa verifice care sunt vecinii sai si sa ii calculeze fiecaruia numarul de constrangeri hard si soft.

Algoritm	Constrangeri hard	Constrangeri soft	Stari generate/iteratii	Timp de executie
HC	0	0	91,465	21.762 s
PCSP	0	0	1,165,134	0.267 s

4.4 Orar mare relaxat

In acest test se poate observa ceea ce am observat la precedentul, doar ca dus la extrem.

Se poate observa ca PCSP are norocul de a ajunge repede la o solutie finala (probabil si prin modalitatea in care parcurge variabilele si valorile).

Si HC ajunge la solutie in putine iteratii, doar ca numarul de vecini creste si mai mult fata de testul precedent si ajunge sa se impotmoleasca in calcule.

Algoritm	Constrangeri hard	Constrangeri soft	Stari generate/iteratii	Timp de executie
HC	0	0	151,663	56.651 s
PCSP	0	0	6,324	0.070 s

4.5 Orar constrans incalcat

Pentru varianta de orar constrans am apelat la tehnica de random restart. Rulez algoritmul PCSP de un numar de maxim 500 de iteratii (de fiecare data cu ordinea variabilelor si a valorilor randomizata) pana cand gasesc o configuratie care satisface costul acceptabil. Pornesc cu costul acceptabil de la 8 si tot cresc pana

cand gasesc un cost acceptabil pentru care am solutie. In plus, am un numar maxim de iteratii in parcurgerea PCSP de 70000 cand folosesc aceasta metoda.

Din observatiile mele, pentru costul acceptabil 7 este mai greu sa gasesti solutie (nu gasesc intotdeauna dupa cele 500 de iteratii), dar am decis sa reprezint in tabel un astfel de caz.

In general gasesc solutii pentru costul 8 (din acest motiv l-am desemnat drept average) si sigur gasesc pentru 9.

Fara varianta de Random Restart algoritmul meu ajunge la 0 constrangeri hard si 61 de constrangeri soft (nu am reusit sa ajung la un rezultat pozitiv pentru un cost mai mic de 61).

Algoritm	**Constrangeri hard**	**Constrangeri soft**	**Stari generate/iteratii**	**Timp de executie**
HC	0	11	34,749	6.271 s
PCSP simplu	0	61	939	0,047 s
PCSP RR* best (cu acceptable_cost 7)	0	7	5,300,237	4.299 s
PCSP RR avg (cu acceptable_cost 8)	0	8	11,954,507	9,597 s

*RR = Random Restart