Inteligență Artificială - Tema 1 - Orar

Cismaru Diana-Iuliana

331CA

Cuprins

1 Descrierea problemei **1**

2 Abordarea problemei **1**
2.1 Generarea și reprezentarea stării inițiale . . . 2
2.2 Generarea stărilor succesoare . . . 3
2.3 Reprezentarea restricțiilor . . . 3
2.4 Bonus - restricții legate de intervalul de pauză pentru profesori . . . 3

3 Analiza rezultatelor obținute **4**
3.1 dummy . . . 4
3.2 orar_mic_exact . . . 5
3.3 orar_mediu_relaxat . . . 6
3.4 orar_mare_relaxat . . . 7
3.5 orar_constrans_incalcat . . . 8
3.6 orar_bonus_exact . . . 9
3.7 Rezultatele testelor . . . 10

4 Concluzii **10**

5 Utilizare ChatGPT **10**

1 Descrierea problemei

Problema pe care am rezolvat-o constă în implementarea unei soluții în Python pentru generarea automată a unui orar, ținând cont atât de constrângeri "neîncălcabile", cat și de constrângeri "încălcabile", încercând minimizarea acestora.

2 Abordarea problemei

Metoda mea de rezolvare presupune utilizarea algoritmului de căutare **Hill Climbing**, utilizând varianta **stochastică**, alături de **Random Restart**.

Un restart al algoritmului Hill Climbing Stochastic presupune generarea unei noi stări inițiale și generarea a mai multe stări succesoare, până maximul de iterații este atins. Dacă la acel moment soluția nu este de cost 0, se reia mecanismul de restart, până la maximul de încercări stabilite. În abordarea mea, am limitat iterațiile la **40** și restart-urile la **70**, astfel încât să ajung la rezultatul dorit.

Înainte să încep aplicarea algoritmului, am parsat datele din fișierul de intrare corespunzător testului și le-am reorganizat într-un mod cât mai eficient. Astfel, mă folosesc de 3 dicționare constante pe parcursul rulării:

1. **timetable_specs** - rezultatul funcției din scheletul temei, *read_yaml_file()*
2. **teacher_constraints** - constrângerile pe zile, intervale și pauze, pentru fiecare profesor
3. **subject_info** - informații referitoare la fiecare materie, precum: câți studenți au fost asignati la aceasta, ce clase pot găzdui ore pentru materia respectivă și ce profesori o pot susține. În plus, materiile sunt sortate în dicționar după numărul de profesori ce o pot preda

2.1 Generarea și reprezentarea stării inițiale

O stare este reprezentată de o instanță a clasei **State**, asemănătoare cu cea din laborator. Membrii acestei clase sunt cele 3 dicționare menționate anterior, dar și următoarele:

- **timetable** - dicționarul de la momentul curent ce reprezintă orarul efectiv
- **teacher_schedule** - orarul fiecărui profesor, în care, pentru orice interval din oricare zi, este contorizat numărul de ore pe care profesul respectiv le susține
- **breaks_hard_conflicts** - se actualizează la fiecare nouă instanțiere si reține True dacă exista constrângeri hard încălcate în orarul curent, sau False altfel
- **soft_conflicts** - reține numărul efectiv de constrângeri soft încălcate de orarul curent

Atunci când se generează o stare inițială, e nevoie de generarea unui orar de la care să pornesc. Pentru a eficientiza timpul de căutare, am decis să pornesc cu starea inițială ca fiind un orar deja completat, care nu încalcă nicio constrângere hard, dar poate avea oricâte constrângeri soft încălcate.

Modul în care am generat acest orar este următorul:
- am luat pe rând **fiecare materie** disponibilă în dicționarul subject_info, pentru a încerca să acopăr întâi acele materii care nu au multe opțiuni în ceea ce privește profesorii ce pot să o predea;
- în cazul în care soluția generată până la acest moment se estimează ca fiind una cu șansă mică de succes, se oprește căutarea;
- **cât timp există studenți ce nu au fost alocați** încă la niciun slot din orar pentru materia curentă, se alege random **o sală**, dintre cele în care se poate preda materia curentă;
- se alege random **o zi** și **un interval**, dintre cele în care sala aleasă este disponibilă;
- se face o listă de profesori **„candidați"**, în care sunt selectați doar cei care pot preda materia și care nu sunt ocupați în intervalul selectat;
- din această listă de candidați, ordonez profesorii în felul următor: cei care nu au restricții împotriva slot-ului selectat vor fi la începutul listei, iar cei care impun restricții vor fi plasați la final;
- se parcurge lista de candidați, până se găsește unul care să aiba **mai puțin de 7 ore predate** pana la momentul curent, pentru că altfel s-ar încălca o constrângere obligatorie; dacă nu există niciun candidat, atunci soluția sigur este una proastă;
- dacă se ajunge în punctul în care am selectat o materie, o zi, un interval, o clasă și un profesor, trebuie ocupat slot-ul din orarul mare, dar și din orarul intern al profesorului.

În această abordare, dacă soluția a fost marcată ca fiind una fără șansă de succes, se regenerează acest orar, până se găsește unul ce nu încalcă deloc constrângeri hard.

2.2 Generarea stărilor succesoare

Funcția *get_best_neighbors()* generează o listă cu toate stările succesoare de la momentul curent, care au un cost **cel putin egal** cu cel găsit anterior. Printr-o stare succesoare înțeleg o stare ce diferă de starea curentă printr-o permutare între conținuturile a două sloturi din orar. Generarea și verificarea stărilor succesoare pornește de la un produs cartezian între toate combinațiile de zile, intervale si săli de clasă existente. Totuși, pentru a reduce spațiul de căutare, am evitat verificarea duplicatelor, astfel că, dacă, de exemplu, am generat starea corespunzătoare permutării *(Luni, 8-10, EG324) - (Miercuri, 10-12, EG390)*, atunci nu o voi mai genera și pe cea corespunzătoare permutării *(Miercuri, 10-12, EG390) - (Luni, 8-10, EG324)*.

Înainte de a fi generată, o stare succesoare trece printr-un proces de verficare, astfel încât să nu încalce absolut nicio constrângere hard, dar și pentru ca aceasta să nu fie redundantă, ca de exemplu, generarea unei permutari între două slot-uri goale. Este important de menționat că nu am restricționat generarea stărilor care nu influențează deloc costul, deoarece, există stări din care se pot aduce îmbunătățiri în mai mulți pași, nu neapărat din primul pas, adică sunt stări cu șansă mare de reușită, chiar dacă la prima vedere nu par așa.

Pentru o stare, **lista stărilor succesoare se comprimă** doar la acei succesori care au costul minim posibil la acel moment, iar din cei rezultați se va alege unul singur cu care căutarea continuă.

În cazul în care, la două iterații consecutive nu găsesc un cost strict mai mic decât cel anterior, și costul curent este 1 sau 2, **ofer șansa a 30 de iterații în plus**, ce maximizează șansa de a ajunge într-o stare de cost 0.

2.3 Reprezentarea restricțiilor

Pentru verificarea restricțiilor pe parcursul executării programului, m-am folosit în principal de funcțiile din fișierul helper.py:

- **breaks_hard_constraints()**: este aproape la fel ca cea prezentă în schelet, cu excepția că nu se mai numără conflictele, deoarece scopul abordării mele este de a ajunge la o soluție ce nu încalcă niciodată constrângerile hard, astfel că, atunci când este detectată o constrângere încălcată, funcția întoarce direct True, pentru a economisi timp;

- **check_soft_constraints()**: funcția a fost rescrisă astfel încât să minimizeze overhead-ul temporal. Pentru a verifica dacă se respectă constrângerile atât pentru zile, cât și pentru intervale, se parcurge orarul tuturor profesorilor o singură dată și se verifică existența restricțiilor atunci când se observă că profesorul respectiv predă la acel moment. Pentru restricțiile legate de pauza dintre orele profesorilor, voi detalia în subcapitolul următor.

2.4 Bonus - restricții legate de intervalul de pauză pentru profesori

Verificarea restricțiilor în acest caz se face tot în funcția *check_soft_constraints()*. Pentru a analiza pauzele din orarul profesorilor, se extrage lista itemilor din acest dicționar, conținând doar valori de 0 si 1, unde 1 reprezintă un slot ocupat de profesor, iar 0 un slot gol. În momentul în care, în lista respectivă, există un zero sau mai multe zero-uri între cel puțin două valori de 1, înseamnă ca s-a detectat o pauză în orar. Răspunsul de pe forum a făcut ca interpretarea costului în aceste situații să fie la latitudinea mea. Așadar, am considerat următoarele:

- dacă profesorul nu acceptă deloc pauze în orar, costul să fie numărul pauzelor, fără a se ține cont de durata lor. De exemplu, [1, 0, 0, 1, 0, 1] va avea cost 2;

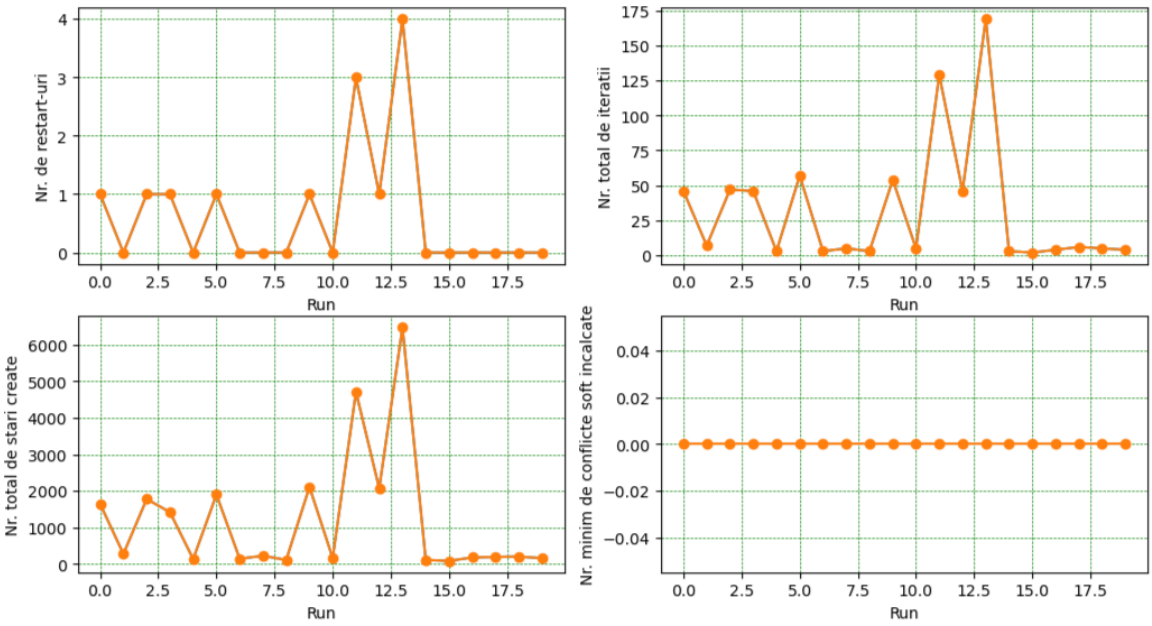
- dacă nu se acceptă pauze mai mari de două ore, se vor număra situațiile în care apar cel puțin două zero-uri între cel puțin două valori de 1. De exemplu, [1, 0, 0, 1, 0, 1] va avea cost 1. Analog pentru situațiile în care nu se acceptă pauze mai mari de patru sau de șase ore.

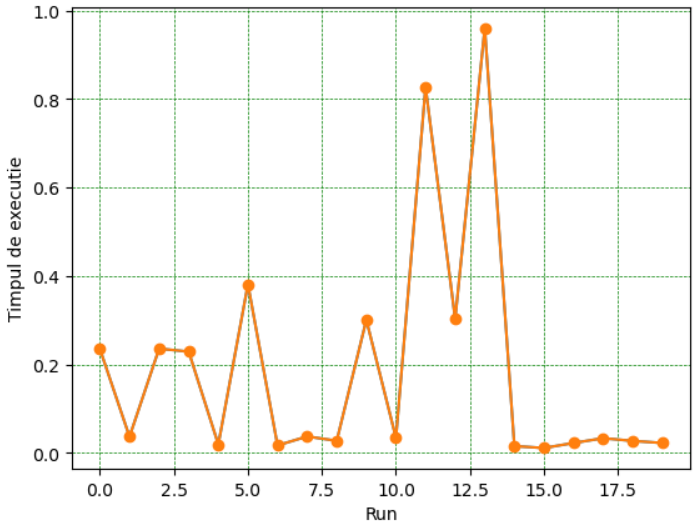
3 Analiza rezultatelor obținute

Pentru analiza rezultatelor, am plotat numărul de restart-uri, numărul total de iterații, numărul total de stări create, numărul minim de conflicte soft încălcate și timpul de execuție pentru mai multe rulări ale fiecărui test. Testarea s-a realizat pe un sistem de calcul cu Intel i3-7100, 3.90GHz si 16GB RAM.

3.1 dummy

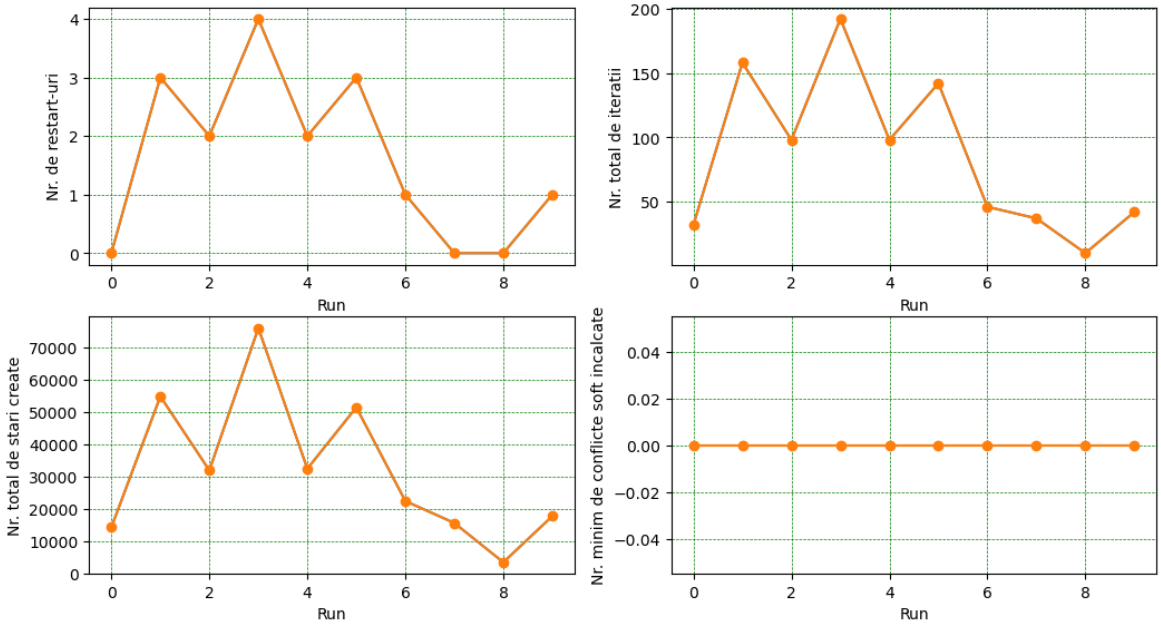
Pe testul *dummy*, analiza s-a realizat pe 20 de rulări. Timpul de execuție a fost aproape instant. Pentru a ajunge la o soluție de cost 0, se observă că numărul de restart-uri este foarte mic, uneori chiar 0. Numărul de iterații este proporțional cu numărul de restart-uri, ajungând la un minim de 2 iterații și un maxim de 169 de iterații. Numărul de stări create poate ajunge să fie de ordinul miilor.

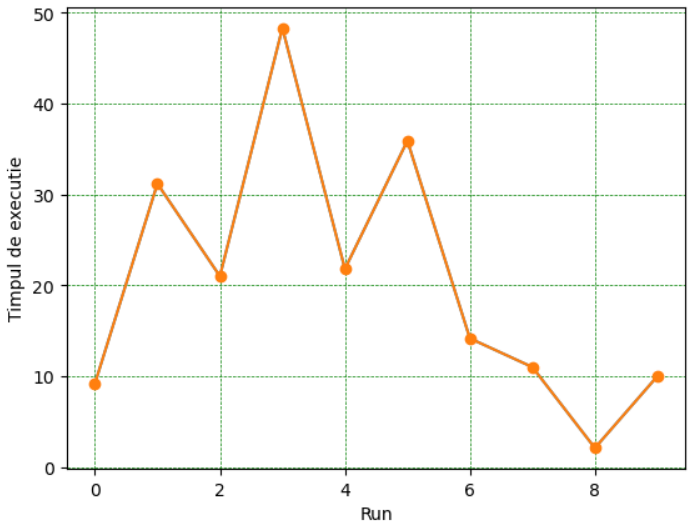




3.2 orar_mic_exact

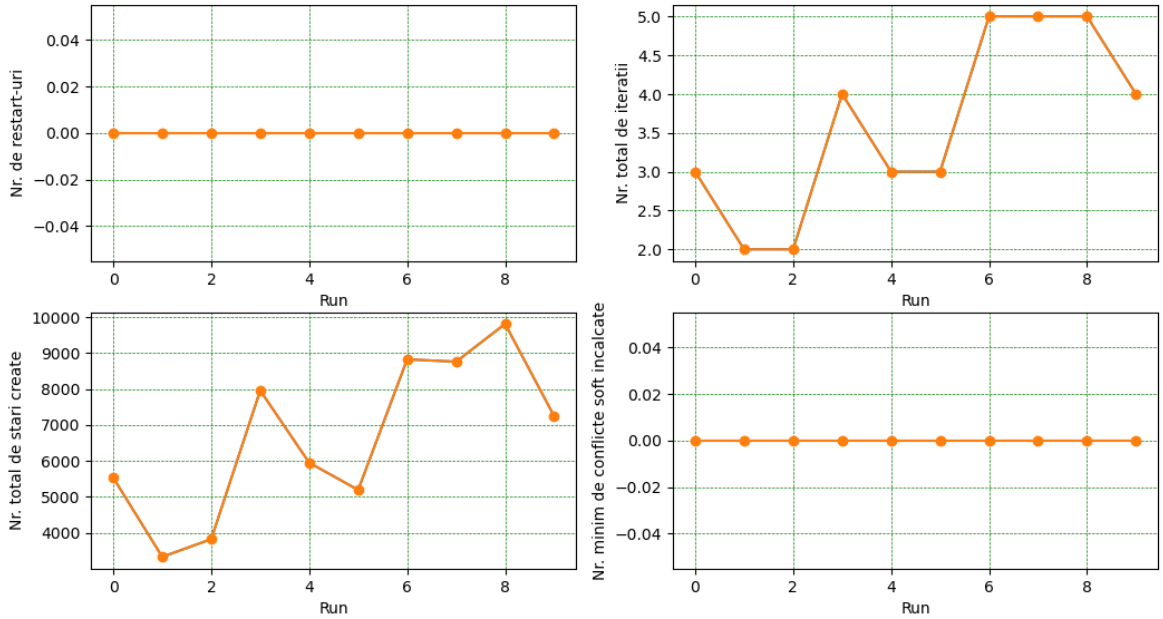
Analiza s-a realizat pe 10 rulări. Timpul de execuție a fost unul destul de mic, între 3 si 49 de secunde. Pentru a ajunge la o soluție de cost 0, se observă că numărul de restart-uri este foarte mic, uneori nefiind nevoie de restart. Numărul de iterații este proporțional cu numărul de restart-uri, ajungând la un minim de 10 iterații și până la maxim 192 de iterații. Numărul de stări create poate ajunge să fie de ordinul zecilor de mii.

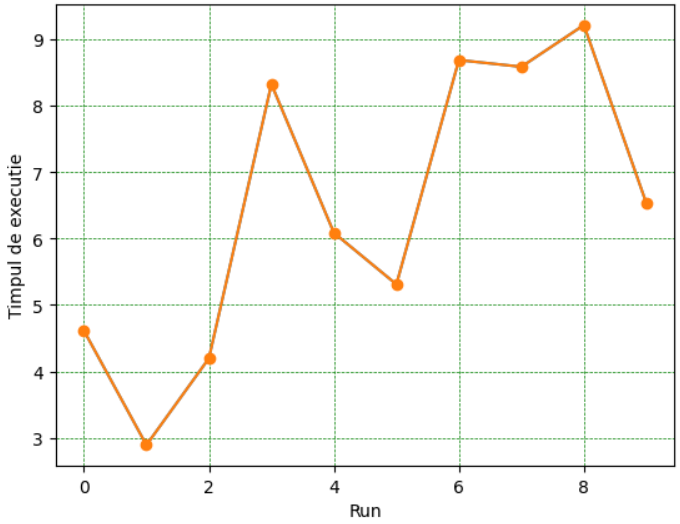




3.3 orar_mediu_relaxat

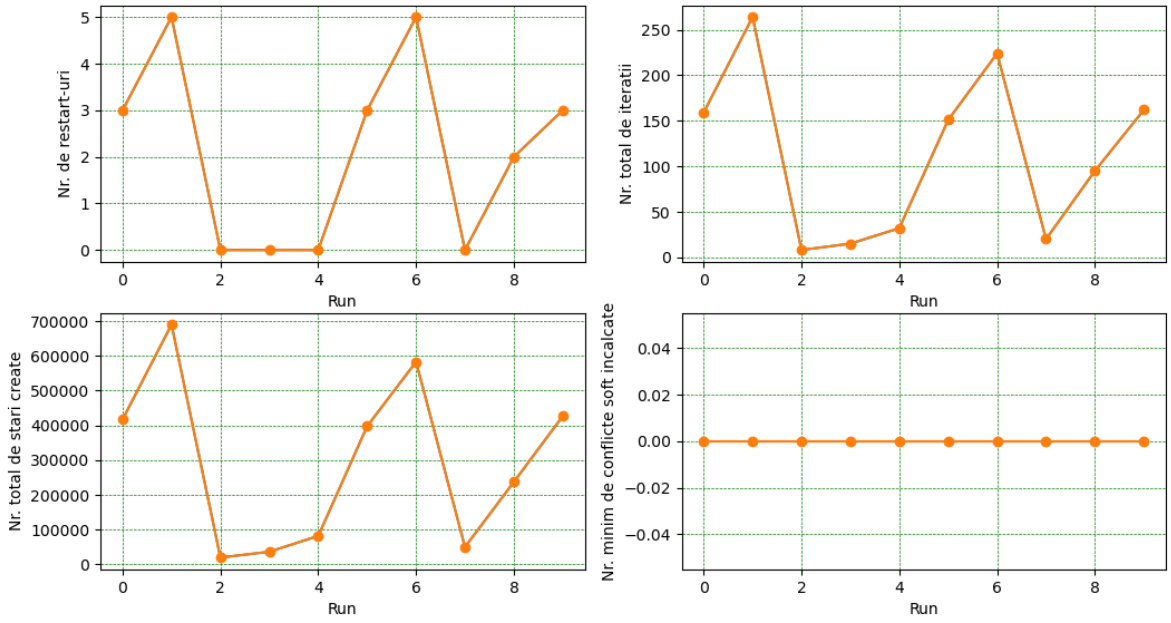
Analiza s-a realizat pe 10 rulări. Timpul de execuție a fost unul foarte mic, între 3 și 10 secunde. Pentru a ajunge la o soluție de cost 0, se observă că nu a fost nevoie de restart, reușind să se ajungă la starea finală în mai puțin de 5 iterații de fiecare dată. Numărul de stări create este de ordinul miilor.

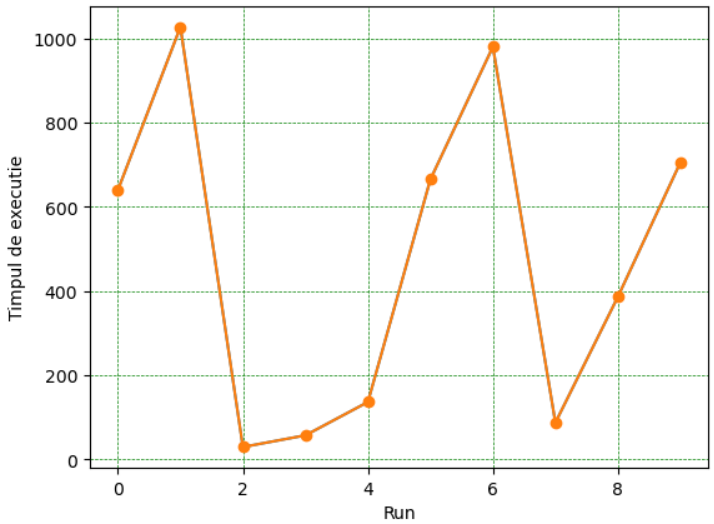




3.4 orar_mare_relaxat

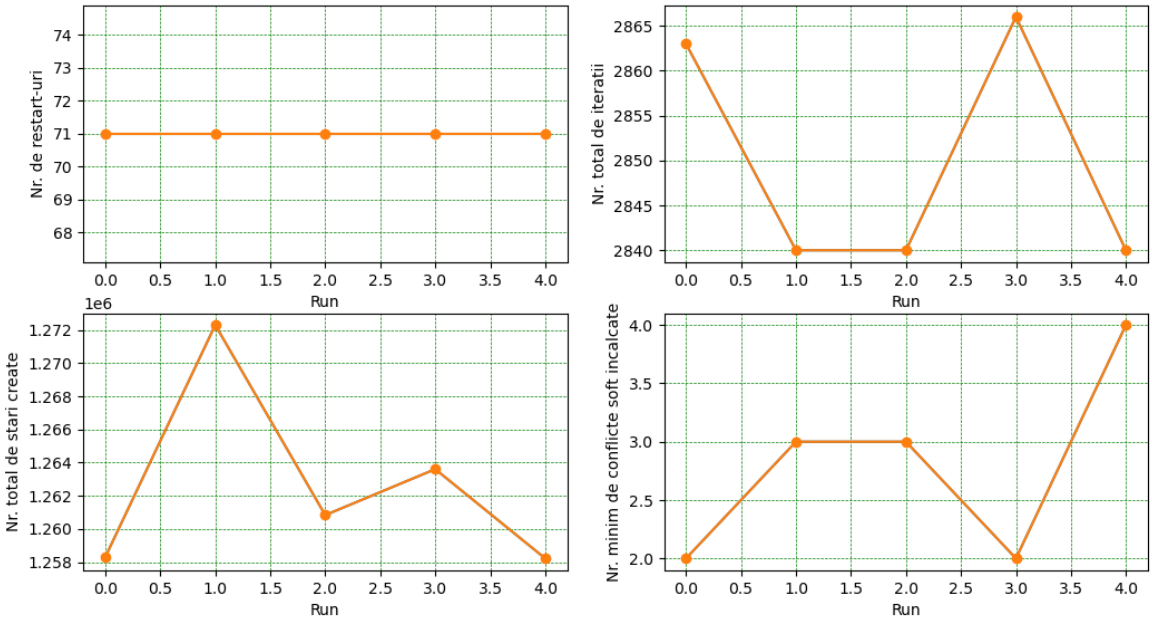
Pe testul *orar_mare_relaxat*, analiza s-a realizat pe 10 rulări, timpul de execuție variind foarte mult în functie de numărul de restart-uri. Pentru a ajunge la o soluție de cost 0, se observă că a fost nevoie de maxim 5 restart-uri, iar numărul de iterații este proporțional cu acestea, ajungând la un minim de 2 iterații și un maxim de 260 de iterații. Numărul de stări create poate ajunge să fie de ordinul sutelor de mii, în cazuri nefericite.

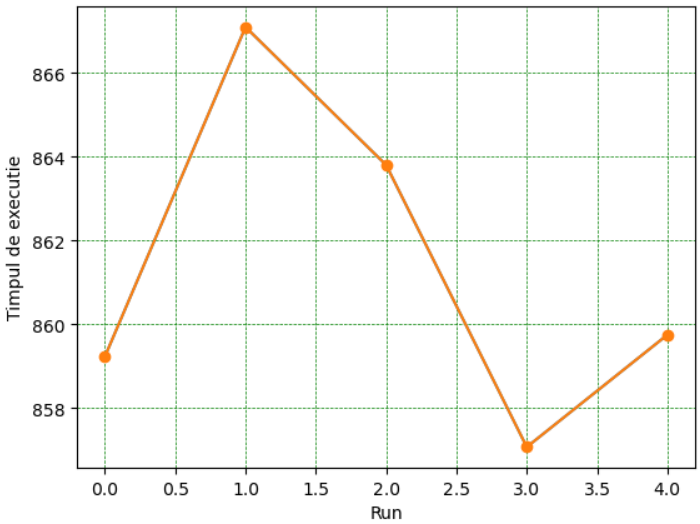




3.5 orar_constrans_incalcat

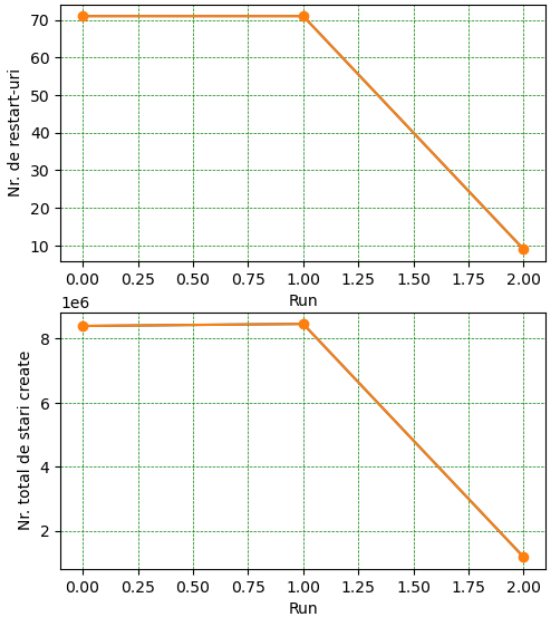
Pe testul *orar_constrans_incalcat*, analiza s-a realizat pe 5 rulări, din cauza timpul de execuție, care este foarte mare. Acest fapt se datorează faptului că, atunci când algoritmul nu obține soluție de cost 0, continuă să facă restart-uri până ajunge la limita maximă. Astfel, la toate cele 5 rulări a fost nevoie de toate restart-urile, iar numărul de iterații a ajuns la 2860. Numărul de stări create a ajuns sa fie de ordinul milioanelor. Totuși, în toate cele 70 de restart-uri, niciuna dintre rulări nu a reușit să obțină cost mai mic de 2.

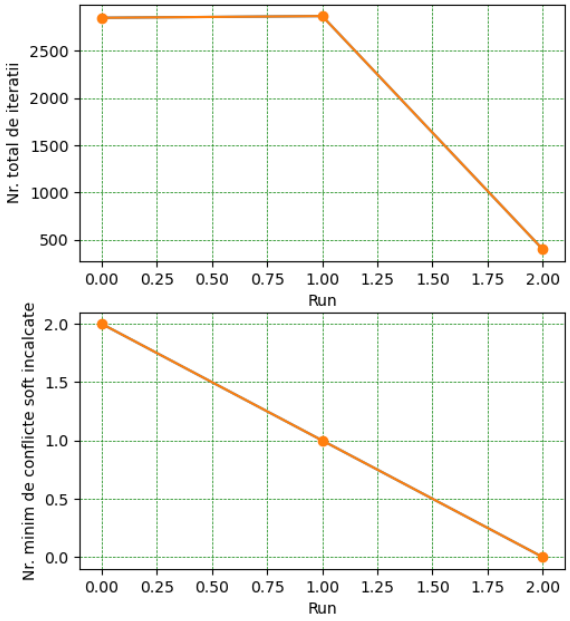


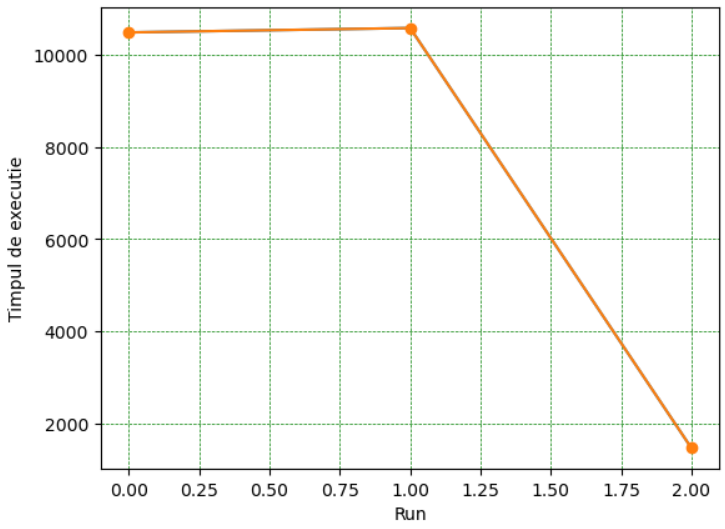


3.6 orar_bonus_exact

Pe testul *orar_bonus_exact*, analiza s-a realizat doar pe 3 de rulări. S-a dorit să se ruleze de 5 ori, însă timpii au fost extrem de mari. Norocul a fost că, la cea de-a treia rulare, s-a obținut o soluție de cost 0 după doar 9 restart-uri, spre deosebire de primele 2, în care s-a rulat până la limita de 70 de restart-uri, obținând costuri de 2 sau 1. Numărul stărilor a depășit 8 milioane, în cazurile nefavorabile. Aceste rezultate sunt cauzate de faptul că numărul restricțiilor a crescut considerabil, intervenind și restricțiile de pauze.







3.7 Rezultatele testelor

Pe lângă cele prezentate mai sus, se pot analiza rezultatele afișate în fișierele din directorul *output/*, pe baza cărora s-au realizat graficele cu ajutorul comenzii:

```
> python3 orar.py hc <nume_fisier> <numar_rulari>
```

De asemenea, se pot face pe loc alte rulări, pentru care rezultatul se va regăsi în directorul */output-test*, cu ajutorul comenzii:

```
> python3 orar.py hc <nume_fisier>
```

4 Concluzii

În urma analizei fiecărui grafic, se observă ca s-a reușit obținterea soluției de cost 0 pe toate testele, cu excepția celui menit să accepte soluția de cost nenul. S-au făcut compromisuri pe celelalte părți, pe timp și spațiu. Consider că, pe testele de bază, am reușit sa obțin timpi destul de buni, dat fiind faptul că am făcut diverse optimizări, atât la generarea stării inițiale, cât și la generarea stărilor succesoare. Pe de altă parte, memoria aduce un overhead considerabil, ținând cont ca ajung să creez stări de ordinul milioanelor. Acest lucru s-ar fi evitat dacă aș fi impus mai multe constrângeri la generarea stărilor vecine, cu riscul de a elimina stări aparent proaste, dar cu potențial mare de reușită.

P.S.: Motivul pentru care nu am implementat și A* este că, doar la aceasta parte de temă s-a lucrat minim 60 de ore și nu am avut cum fizic să fac mai mult de atât. Mi-a plăcut, totuși!

5 Utilizare ChatGPT

Am utilizat ChatGPT pentru indicații de utilizare a bibliotecilor *time* si *matplotlib.pyplot*. De asemenea, m-am folosit de el și pentru ajutor cu paginarea în LaTeX.

Bibliografie

1. Forum Tema 1 CA
2. Forum Tema 1 CB
3. Forum Tema 1 CC
4. Forum Tema 1 CD
5. Laborator Search HillClimbing
6. ChatGPT