Inteligență Artificială: Tema 1 – Orar

Seria CD

Andrei Olaru, Cătălin-Mihail Chiru, Andrei Dugăeșescu,
Vlad-Constantin Lungu-Stan, Andrei Mihalea

Ultima modificare: 5 aprilie 2024

1 Descrierea problemei

Bazându-vă pe cunoștințele dobândite în cadrul cursurilor și laboratoarelor de IA, sarcina voastră este să implementați o soluție în limbajul de programare Python pentru generarea automată a unui orar fără conflicte / cu un număr cât mai mic de conflicte, pe structura oralelor cu care v-ați întâlnit pe parcursul anilor de facultate. Problema modelează programarea unor activități de tip laborator/seminar, având următoarele elemente:

- Un slot din orar ocupă 120 de minute (e.g. 8-10) și există 6 sloturi într-o zi de lucru (de la 8 la 20) (excepție primul caz de test)
- Orele se țin de Luni până Vineri (excepție primul caz de test)
- Există un număr limitat de săli în care se pot desfășura orele. Fiecare sală are o capacitate maximă de studenți și este repartizată specific pentru anumite materii.
- Fiecare materie din orar are un număr specific de studenți pentru care trebuie să se aloce săli.
  - **Exemplu**:
    Avem o materie X cu 90 de studenți și 2 săli disponibile:
    sala A cu 50 de locuri disponibile
    sala B cu 20 de locuri disponibile
    Sunt acoperiri valide pentru materia X atât:
    2 intervale diferite în sala A ($2 * 50 = 100 \geq 90$)
    1 interval în sala A și 2 intervale în sala B ($1 * 50 + 2 * 20 = 90 \geq 90$)
- Există un număr limitat de profesori, fiecare specializat în anumite materii la care pot preda ore. Fiecare profesor are propriile preferințe referitoare la poziționarea orelor sale.
- Profesori diferiți pot preda aceeași materie în săli diferite în același interval orar.

Există două tipuri de constrângeri pe care trebuie să le avem în vedere pentru a genera un orar valid: cele implicite / hard și cele încălcabile / soft.

1.1 Constrângeri implicite / hard

Constrângerile hard sunt de ordin fizic sau logistic, și, o dată încălcate, produc orare imposibil de urmat:

- într-un interval orar și într-o sala se poate susține o singură materie de către un singur profesor.
- într-un interval orar, un profesor poate ține o singură materie, într-o singură sală.
- un profesor poate ține ore în maxim 7 intervale pe săptămână.
- o sală permite într-un interval orar prezența unui număr de studenți mai mic sau egal decât capacitatea ei maximă specificată.
- toți studenții de la o materie trebuie să aibă alocate ore la acea materie. Concret, suma capacităților sălilor peste toate intervalele în care se țin ore la materia respectivă trebuie să fie mai mare sau egală decât numărul de studenți la materia respectivă. (vezi **Exemplu** mai sus)
- toți profesorii predau doar materiile pe care sunt specializați.
- în toate sălile se țin ore doar la materiile pentru care este repartizată sala.

1.2 Constrângeri încălcabile / soft

Constrângerile soft se referă la preferințele profesorilor. Este preferabil să încălcăm constrângeri soft dacă acest lucru va genera un orar valid, decât să ajungem în imposibilitatea de a completa orarul.

Constrângerile profesorilor pot fi de următoarele tipuri:

- Preferă anumite zile sau nu doresc să predea într-o zi anume.
  - **Exemplu:**
    Luni → profesorul preferă să predea Luni
    !Marți → profesorul preferă să nu predea marți
- Preferă sau nu doresc anumite intervale orare, în oricare din zile
  - **Exemplu:**
    8-12 → profesorul preferă să predea în oricare dintre intervalele 8-10 sau 10-12
    !14-20 → profesorul preferă să nu predea în intervalele 14-16, 16-18, 18-20
- [BONUS] Preferă să nu aibă ferestre în orar mai mari de X ore
  - **Exemplu:**
    !Pauză > 0 → profesorul nu dorește nicio pauză în orarul său
    !Pauza > 2 → profesorul nu vrea ferestre mai mari de 2 ore

ATENȚIE!

Preferințele personale ale profesorilor sunt mereu exhaustive, știind pentru fiecare zi sau interval orar părerea fiecărui profesor față de ele.

2 Intrare și ieșire

Pentru a vă ajuta cu parsarea datelor de intrare, intrările vă sunt date în format YAML.

Hint

Folosiți biblioteca `yaml` pentru parsarea fișierelor de intrare.

Vom analiza structura fișierului `dummy.yaml`, pentru a detalia formatul de intrare (notă: `dummy.yaml` are un set redus de zile și intervale, pentru o rezolvare mai rapidă):

```
1  Intervale:
2  - (8, 10)
3  - (10, 12)
4  - (12, 14)
5  Zile:
6  - Luni
7  - Marti
8  - Miercuri
```

```
9  Materii:
10   DS: 100       // număr studenți pentru materia DS
11   IA: 75
12   MS: 100
```

Figura 1: Prima parte din fișierul de intrare

Avem următoarele secțiuni

- În chenarul albastru (Figura 1), dicționarele 'Zile' și 'Intervale' conțin totalitatea zilelor și intervalelor care pot apărea în orar.
- În chenarul portocaliu (Figura 1), dicționarul 'Materii' conține maparea materiilor pe care le avem de acoperit în orar cu numărul de studenți alocați la aceste materii.
- În chenarul turcoaz (Figura 2), sunt ordonați lexicografic profesorii, după prenume. Dicționarul 'Profesori' conține după cheia cu numele fiecărui profesor două intrări: una cu constrângerile profesorului respectiv și una cu materiile pe care le poate preda

```yaml
Profesori:
  Andreea Dinu:
    Constrangeri:
    - Luni          // preferă Luni și Marți
    - Marti
    - '!Miercuri'   // preferă să nu predea Miercuri
    - '!8-10'       // nu preferă intervalele 8-10, 12-14
    - '!12-14'
    - 10-12
    Materii:
    - DS            // poate preda DS și IA
    - IA
  Cristina Dumitrescu:
    Constrangeri:
    - '!Luni'
    - Marti
    - '!Miercuri'
    - '!10-12'
    - 8-10
    - 12-14
    Materii:
    - MS
    - DS
...
```

```yaml
Sali:
  EG324:
    Capacitate: 25
    Materii:
    - MS
    - IA
  EG390:
    Capacitate: 25
    Materii:
    - DS
```

Figura 2: A doua parte din fișierul de intrare

- În chenarul verde (Figura 2), dicționarul 'Săli' conține după cheia cu numele fiecărei săli două intrări: Capacitea sălii - câți studenți intră într-un interval în acea sală, și Materiile care pot fi ținute în sala respectivă

Outputul fiecărui algoritm este o atribuire de profesori și materii la săli și intervale din fiecare zi. Un exemplu de soluție pentru intrarea `dummy.yaml` este în Figura 3. (output produs cu funcția de pretty printing din `utils.py`).

```
|Interval |          Luni          |          Marti          |        Miercuri         |
-------------------------------------------------------------------------------------------
| 8 - 10  |    DS : (EG390 - EG)    |    EG390 - liber        |    DS : (EG390 - EG)    |
|         |    MS : (EG324 - RG)    |    MS : (EG324 - CD)    |    EG324 - liber        |
-------------------------------------------------------------------------------------------
| 10 - 12 |    DS : (EG390 - AD)    |    EG390 - liber        |    EG390 - liber        |
          |    EG324 - liber        |    IA : (EG324 - AD)    |    IA : (EG324 - PF)    |
-------------------------------------------------------------------------------------------
| 12 - 14 |    EG390 - liber        |    DS : (EG390 - CD)    |    EG390 - liber        |
|         |    IA : (EG324 - PF)    |    MS : (EG324 - RG)    |    MS : (EG324 - RG)    |
-------------------------------------------------------------------------------------------
```

Figura 3: Soluția pentru intrarea `dummy`.

2.1 Cazuri de test

Vom folosi următoarele cazuri de test:

- `dummy.yaml` conține o problemă simplă, cu doar 3 zile și 3 intervale orare în fiecare zi, pentru a vă oferi un mediu de testare fail-fast în implementare.
- `orar_mic_exact`, `orar_mediu_relaxat` și `orar_mare_relaxat` suportă soluții care nu încalcă nicio constrângere (au cost 0); niște posibile soluții sunt în directorul `output`.
- `orar_constrâns_încălcat` verifică faptul că rezolvarea suportă soluții care încalcă constrângerile; în `outputs/orar_constrans_incalcat.txt` se găsește o soluție de cost 1.
- `orar_bonus_exact` introduce constrângerile de tip "!Pauză > X" și suportă o soluție de cost 0, în `outputs/orar_bonus.txt`.

3 Cerințe

Veți avea de implementat 2 algoritmi dintre cei implementați la laborator, pentru problema orarului, și va trebui să comparați performanța lor. Concret, trebuie să realizați:

1. reprezentarea unei stări / a unei soluții parțiale pentru problemă (10 puncte).
2. reprezentarea implicită sau explicită a restricțiilor din problemă (10 puncte).
3. rezolvarea problemei folosind un algoritm de tip A* (30 puncte).
4. rezolvarea problemei folosind un algoritm de tip CSP (30 puncte).
5. un document PDF în care să detaliați (20 puncte):
   - reprezentarea stărilor și a restricțiilor;
   - optimizări pe care le-ați realizat pentru cei doi algoritmi, față de varianta de la laborator, specifice acestei probleme;

- comparația între cei doi algoritmi din punct de vedere al următorilor indicatori:
  - timp de execuție;
  - număr de stări construite;
  - calitate a soluției (număr de restricții încălcate).

[BONUS] pentru bonus, suportați restricții legate de intervalul de pauză pentru profesori (20 puncte).

NOTĂ: chiar dacă nu reușiți să obțineți soluții de cost 0 pentru toate cazurile de test, realizați comparația și explicați la prezentare motivele pentru care credeți că algoritmii implementați de voi produc anumite outputuri.

ATENȚIE

Tema este individuală! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.

ATENȚIE

Este important să vă concentrați pe reprezentarea problemei și modelarea sa în subprobleme, algoritmii nefiind conceptual diferiți față de implementările din laborator, o dată ce aveți problema exprimată în seturi de variabile și valori, sau stări și tranziții de stări.

4 Trimiterea temei

Tema se trimite ca o arhivă `.zip` care conține cel puțin 2 fișiere:

- un fișier Python `orar.py` *sau* Jupyter Notebook `orar.ipynb` care conține implementarea algoritmilor:
  - pentru fișierele Python, programul va primi ca argumente în linia de comandă algoritmul (`astar` sau `csp`) și fișierul de intrare. Programul va afișa soluția la output (pretty-printed).
  - pentru fișierele Jupyter Notebook, fișierul va conține rularea și rezultatele pentru ambii algoritmi ceruți și pentru toate cazurile de test care apar în analiza comparativă.
- un fișier PDF care prezintă elementele cerute la cerința 5.

5 Hints

În fișierul `utils.py` aveți câteva funcții ajutătoare pentru

- citirea fișierelor YAML;

- pretty printing pentru un orar.

Indicii generale:

General Hints

- Pentru slicing eficient, lucrați cu array-uri din numpy. Din experiența noastră, implementări folosind biblioteca pandas suferă de overhead din punct de vedere al complexității temporale.
- Încercați să folosiți mecanisme de copiere a stării cu care sunteți familiari din laboratoare, nu scrieți lucruri "de mână", C-like. Fiind o problemă de recursivitate într-un spațiu foarte mare, orice procesare suplimentară sau orice îmbunătățire se fac simțite imediat în timpul așteptat la rulare.
- Rulați întâi pe testele mici (dummy, mic și mediu) pentru a evita pierderea de timp.
- Dacă algoritmii voștri nu obțin soluții de cost 0, nu faceți overfitting pe testele noastre, concentrați-vă să aveți implementarea / implementările funcționale și explicați în analiza comparativă de ce credeți că nu obțineți cost 0.
- Încercați să folosiți euristici duc la producerea soluțiilor invalide în cât mai puțin pași (fail-fast): Ex. Acoperim întâi materia cu cel mai mic număr de studenți, cu cei mai puțini profesori, sau cu cele mai puține intervale în care pot fi predate.
- Asigurați-vă că vă folosiți de constrângeri, fiindcă multe dintre ele sunt menite să reducă spațiul de căutare. De exemplu, nu căutați acoperirea unei materii într-o sală ce nu acceptă acea materie.
- În momentul în care ați terminat de acoperit o materie, puteți trece direct la altă materie.
- În momentul în care aveți o acoperire completă, puteți opri complet căutarea.

Pentru algoritmii care se folosesc de exprimarea problemei ca o căutare în spațiul stărilor, recomandăm:

State Based Hints

- Abordați o manieră greedy în momentul în care vă stabiliți euristicile.
- Cum ar putea diferi starea curentă de stările copii care derivă din ea?
- Cât de mare este spațiul stărilor? Este tractabil să le verificați pe toate? Dacă nu, cum ați putea să le filtrați în așa fel încât să explorați nodurile cele mai promițătoare?

- Vă poate ajuta și în alte contexte condiția de Hill Climbing de oprire din explorare când obțin un cost mai mare decât best_cost?
- În situații stocastice (aleatoare), completarea unui slot ați putea să nu o modelați pur aleator, ci să stabiliți probabilități de explorare în funcție de potențialul alternativelor.
- Puteți încerca să faceți din preferințe (constrângerile încălcabile) restricții hard și dacă nu se mai pot alege acțiuni care să le respecte, puteți alege mai întâi stările care încalcă cele mai puține preferințe.