Inteligență Artificială: Tema 1 – Sokoban

Seria CA

Deadline: 25.01.2025 ora 23:59

Cătălin-Mihail Chiru, Andrei Dugăeșescu, Mihai Nan, Alexandru-Victor Baciu, Vlad Matei Drăghici, Bogdan-Andrei Sprîncenatu, Andrei Olaru

Ultima modificare: 1 aprilie 2025

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 soluțiilor jocului de Sokoban încercând să explorați un număr cât mai mic de stări și folosind cât mai puține acțiuni de tip pull.

1.1 Sokoban

În jocul de Sokoban vă aflați pe o hartă bidimensională, cu obstacole, unde trebuie să împingeți cutii. Sunteți un jucător marcat cu un disc roșu (Figura 1), și trebuie să împingeți una sau mai multe cutii (pătrate verzi cu pătrate albastre în interior) pe niște poziții țintă (pătrate galbene cu X-uri verzi deschis în interior).

Resurse utile pentru a înțelege și a vă juca sunt: Sokoban OpenAI Gym, Sokoban Hard Gameplay.

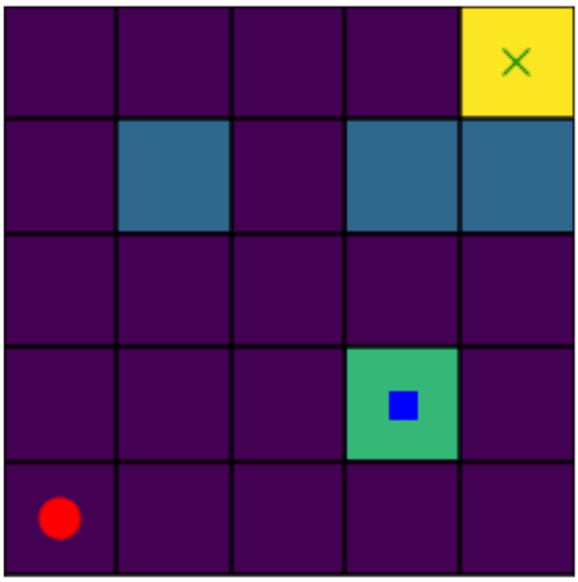


Figura 1: Jocul de Sokoban - Harta easy_map1

Tradițional, dificultatea principală a jocului de Sokoban constă în ireversibilitatea împingerii greșite a cutiilor, care forțează jucătorul să o ia de la început. (Anumite stări ale jocului original sunt ireversibile).

Pentru a nu fi frustrant pentru voi, am implementat în cadrul mediului nostru de joc acțiunea de tragere a cutiilor spre jucător (en. pull), astfel asigurând că orice stare a jocului este reversibilă.

ATENȚIE!

Pentru a vă aprecia efortul implementărilor eficiente, **Algoritmii care nu folosesc mișcări de tip pull vor fi punctați bonus!**

2 Cerințe

Veți avea de implementat 2 algoritmi dintre cei parcurși la laborator sau la curs, din categoria problemelor de căutare în spații de stări, și va trebui să comparați performanța lor.

ATENȚIE!

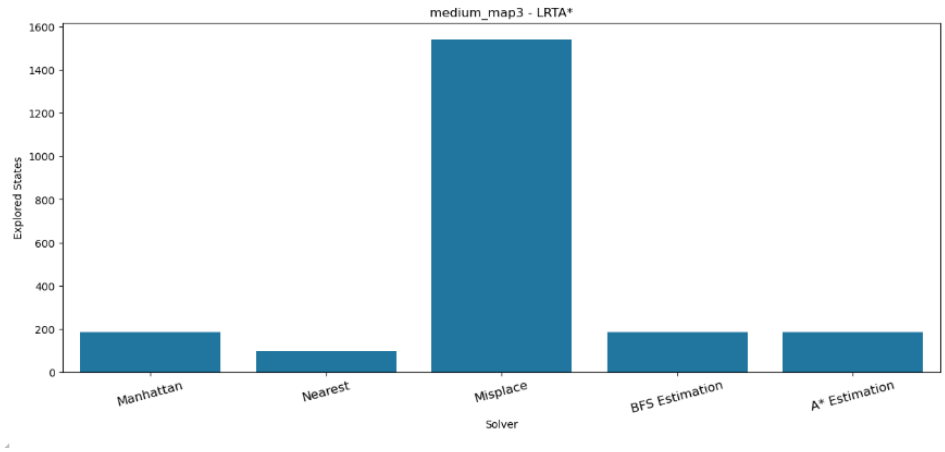
Deoarece o mare parte din efortul de modelare a temei se află deja în schelet, accentul va fi pus pe gândirea voastră critică și capacitatea voastră de analiză. Vă vom aprecia eforturile depuse în căutarea unor euristici, referențierea resurselor folosite, precum și parcursul străbătut până a ajunge la cele mai bune soluții ale voastre.
Dorim să vă lăsăm libertatea să explorați diverse idei. Astfel, punctarea maximă a fiecăruia dintre algoritmii rezolvați de voi se va face fie prin obținerea soluțiilor pe toate hărțile propuse într-un timp tractabil (sub 5 de minute per hartă) fie prin explorarea exhaustivă a minim 2 euristici inteligente (e.g mai complexe decât cele din laboratorul A*) și explicația pe baza lor a neajunsurilor care au determinat blocarea jucătorului pe drum.

ATENȚIE!

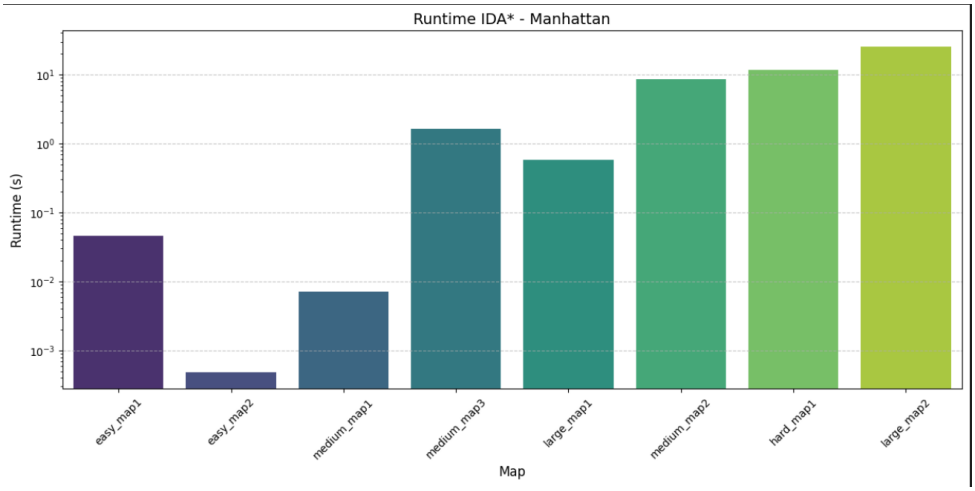
Totodată, accentul în punctaj se pune pe susținerea intuițiilor voastre pe analize calitative (cum parcurge jucătorul jocul de Sokoban) și cantitative (analiza timpilor computaționali, a numărului total de stări expandate, grafice care să susțină vizual comparații între euristici - vezi Figura 2).

Concret, trebuie să realizați:

1. Rezolvarea problemei folosind un algoritm de tip LRTA* (25 puncte).
2. Rezolvarea problemei folosind un algoritm de tip Beam-Search (25 puncte).
3. un document PDF în care să detaliați (50 puncte):



(a) Bar-plot referitor la stările explorate pentru euristica:



(b) Grafic cu evolutia timpilor de rulare pentru toate testele pentru un algoritm si o singura euristica

Figura 2: Exemplu de grafice cantitative

ATENȚIE!

Punctarea documentului PDF este condiționată de implementarea ambilor algoritmi și rularea acestora pe minim 2 teste pentru fiecare - nu ar avea sens comparația fără elementele de comparat!.

- Jurnalizarea ideilor pentru euristicile folosite în cei 2 algoritmi. (e.g. Am început rezolvarea temei folosind ca euristică distanța Euclidiană, pentru că în testul X jucătorul oscilează între două stări pentru algoritmul Y, am analizat evoluția euristicii în acele stări și am decis să fac Z schimbare crezând că va aduce o îmbunătățire. Comportamentul adus de Z a fost următorul...)
- Optimizări pe care le-ați realizat pentru cei doi algoritmi, față de variantele de la laborator sau curs, specifice acestei probleme;
- Comparația între cei doi algoritmi din punct de vedere al următorilor indicatori:
  - timp de execuție; (sprijinit de grafice)
  - număr de stări construite; (sprijinit de grafice)

– calitate a soluției (număr de mișcări de pull).

[BONUS] Pentru bonus, dezvoltarea unor euristici foarte informate, bazate pe idei inovative + cantitativ: obținerea unor soluții cu 0 mișcări de pull (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

Nu este imperios necesară folosirea bibliotecilor matplotlib sau seaborn pentru obținerea graficelor cantitative. Dacă vă sunt mai familiare alte medii de analiza datelor, puteți încărca în prezentare grafice realizate cu acestea (e.g. blender, excel, R-studio)

3 Structura scheletului

În arhiva atașată temei veți observa următoarea structură:

```
|- images                 <- Vor fi create imagini PNG dacă selectați opțiunile corespunzătoare în metodele
↪  de afișaj din map
|- search_methods         <- Aici veți adăuga codul pentru implementările algoritmilor ceruți pentru
↪  rezolvarea jocului Sokoban.
|  |- lrta_star.py          <- LRTA*
|  |- ida_star.py          <- IDA*
|  |- beam_search.py     <- Beam Search
|  |- simmulated_annealing.py     <- Simmulated Annealing
|  |- heuristics.py      <- Modul comun sugerat unde să puneți euristicile fiindcă sunt independente de
↪  algoritmul de căutare folosit
|  \-- solver.py          <- Interfața Solver
|
|- sokoban                <- Structura mediului pentru Sokoban oferită ca bibliotecă
|  |- box.py             <- Clasa ce dezvoltă logica pentru Cutie
|  |- dummy.py           <- Clasa părinte pentru obiecte ce se pot mișca pe hartă
|  |- gif.py             <- Metode pentru a converti soluțiile calitative în GIFuri
|  |- map.py             <- Logica mediului de joc
|  |- moves.py           <- Mișcările date ca MACROuri
|  \-- player.py          <- Clasa ce dezvoltă logica pentru Jucător
|
|- tests                  <- definiții YAML pentru mediile de testare
|- main.py           <- Mainul dacă decideți să lucrați cu scripturi
\- main.ipynb    <- Mainul dacă decideți să lucrați cu Jupyter Notebooks
```

Figura 3: Structura fișierelor corespunzătoare temei.

3.1 Intrare și ieșire

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

Hint

Folosiți Map.from_yaml pentru parsarea hărților din testele noastre, în spate se folosește biblioteca `yaml`.

Outputul vostru trebuie să urmărească două componente:

- **Componenta calitativă** - cu ajutorul metodelor din clasa Map, puteți afișa, fie prin text fie prin imagini, evoluția stărilor parcurse într-o iterație a algoritmului vostru. Pentru vizualizarea cu imagini, aveți funcțiile din gif.py care permit transformarea acestora într-un videoclip de tip gif.
- **Componenta cantitativă** - va trebui să scrieți propriul cod care să realizeze grafice sau să memoreze tabele cu rezultate (se poate realiza cu biblioteci precum pandas sau pickle), pentru a asigura reproductibilitatea rezultatelor voastre. Graficele vă sunt absolut necesare în rezolvarea temei, însă le puteți genera și în afara python, dacă doriți altfel de vizualizări.

Hint

În momentul în care scrieți euristici stocastice, pe lângă memorarea tabelelor de rezultate, putem asigura reproductibilitatea rezultatelor fixând explicit seed-ul generatorului pseudo aleator pe care îl folosim.
Fiecare bibliotecă ce are o componentă aleatoare prezintă metode pentru a asigura acest lucru, spre exemplu în biblioteca random apelăm: **random.seed(0)** - urmând ca operațiile realizate de algoritmii stocastici să se execute mereu identic pentru seed-ul respectiv la rulări succesive.

3.2 Cazuri de test

Vom folosi următoarele cazuri de test:

- `easy*.yaml` conțin probleme simple, din perspectiva numărului de cutii și a dimensiunii spațiului de căutare.
- `medium*.yaml` conțin mai multe cutii decât în easy, spațiul de explorare este în continuare restrâns.
- `hard*.yaml` conțin mai multe sau un număr egal de cutii cu testele medium, însă diferențierea se face prin nevoia de mișcare de pull la o singură mișcare de push greșită.
- `super_hard_map1.yaml` harta cea mai dificilă din testele noastre interne, o vom considera pentru bonus, are rolul de mediu de debug pentru euristicile mai complexe.
- `large*.yaml` conțin probleme mai ușoare decât problemele din hard, dar spațiul de explorare este mai mare.

4 Trimiterea temei

Tema se trimite ca o arhivă `.zip` similară cu cea a scheletului atașat:

- un fișier sau fișiere Python *sau* un Jupyter Notebook care conține implementarea algoritmilor:
  - pentru fișierele Python, programul va primi ca argumente în linia de comandă algoritmul (`lrta*` sau `beam-search`) și fișierul de intrare. Programul va afișa soluția la output sau o va salva ca imagini intr-o ierarhie de fișiere ca în exemplul dat.
  - 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 3.

5 Hints

Indicii generale:

General Hints

- Rulați întâi pe testele mici (easy*, medium1) pentru a evita pierderea de timp.
- Dacă algoritmii voștri nu obțin soluții cu 0 mișcări de pull, sau nu aduc toate cutiile în poziția finală, nu faceți overfitting pe testele noastre, concentrați-vă să aveți implementările funcționale și explicați în analiza comparativă de ce credeți că nu ați obținut rezolvările așteptate.
- Încercați să folosiți euristici care duc la producerea soluțiilor invalide în cât mai puțini pași (fail-fast).
- Dacă aveți neclarități nu ezitați să folosiți Forumul temei și vom mai adăuga aici alte aspecte de ajutor.
- Abordați o manieră greedy în momentul în care vă stabiliți euristicile.

Indicii particulare

Algorithm-Based Hints

- Pentru a nu se bloca într-o situație de ciclare între 2 noduri similare, algoritmul Beam Search are nevoie de niște relaxări. Dintre acestea, adăugarea stocasticității în alegerea mișcării următoare poate ușura ieșirea din impas.
- Tot pentru Beam Search, ați putea implementa un mecanism de restart dacă vedeți că ciclați între niște stări prea mult timp. Ne întoarcem în start, sau într-o stare intermediară mult înaintată din care să reluăm căutarea.