Tema Sokoban Solver

1. Dezvoltarea si logica euristicilor folosite:

Abordarea initiala a fost contruirea unei soluți funcționale, pentru intelegerea jocului Sokoban, dar si cum algoritmii ar performa in cautarea si analizarea starilor. Prima alegere a fost o euristică simplă, bazată pe distanța Manhattan dintre cutii și ținte. Ideea era că, în lipsa altor informații, cea mai apropiată țintă ar trebui să fie ținta ideală pentru fiecare cutie.

Această variantă (simple_heuristic) s-a dovedit eficientă pentru cazurile foarte simple, dar în testele complexe am început să observ probleme majore:

- Jucătorul oscila între două stări similare fără să progreseze.

- Cutiile ajungeau adesea blocate lângă pereți sau în colțuri fără cale de ieșire (deadlock-uri).

- Soluțiile conțineau multe mutări inutile.

Observând aceste blocaje, mi-am dat seama că distanța brută nu este suficientă. Aveam nevoie de o euristică mai "inteligentă", care să înțeleagă și cazurile „blocante” sau redundante ale hărții.

Îmbunătățiri treptate ale euristicilor

Am început să adaug penalizări în funcție de starea cutiilor:

- Dacă o cutie era blocată într-un colț, adăugam o penalizare mare (deadlock corner).
- Dacă o cutie era prinsă între doi pereți sau într-un tunel îngust, aplicam o penalizare moderată.
- Dacă o cutie bloca accesul spre alte cutii, aplicam o penalizare suplimentară.
- Această abordare a dus la dezvoltarea euristicilor matching_heuristic și apoi ida_star_heuristic, care au devenit din ce în ce mai complexe.
- În paralel, pentru Simulated Annealing, unde mutările sunt mai haotice și nu urmează întotdeauna un traseu logic, am dezvoltat target_matching_heuristic. Această euristică este foarte sensibilă la modificări mici ale stării, ceea ce a ajutat algoritmul să evite capcane locale și să îmbunătățească continuu soluția.

În concluzie, evoluția euristicilor mele a urmat această traiectorie:
simplu ➔ mai precaut ➔ penalizare inteligentă ➔ euristici hibride adaptate algoritmului.

2. Optimizări realizate față de variantele de laborator/curs

Pentru Simulated Annealing:

Simulated Annealing, fiind un algoritm stocastic, are nevoie de ajutor suplimentar pentru a nu rămâne blocat într-un minim local. Așadar:

- Am implementat restarturi adaptive: după fiecare căutare nereușită, algoritmul încearcă dintr-o stare ușor perturbată față de inițială.

- Numărul de perturbări nu este fix, ci se adaptează în funcție de scorul soluției anterioare — dacă ultima soluție a fost foarte proastă, perturbez mai agresiv.

- Am introdus un mecanism de validare a soluției: uneori, din cauza naturii aleatorii a mutărilor, o secvență poate duce la o stare invalidă. Înainte de a

accepta orice soluție, o simulez complet și verific dacă într-adevăr rezolvă harta.

Pentru IDA*:

- Implementarea IDA* folosește o reconstrucție eficientă a drumului pe baza unui dicționar parent, astfel evitând reținerii inutile în memorie.

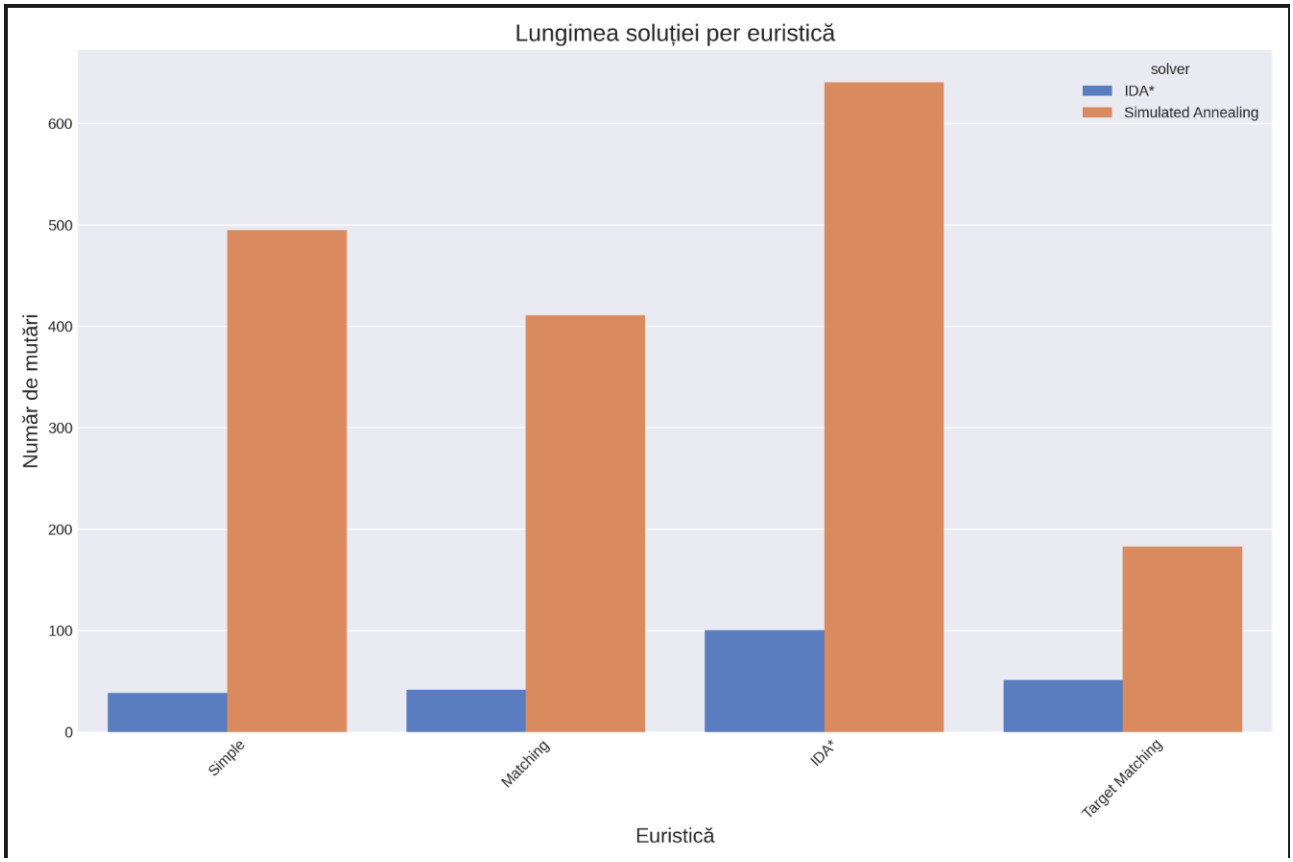
- În loc de o limită de timp brută, IDA* se oprește dacă costul f(n) > limită, ceea ce reduce dramatic numărul de explorări inutile.

- Am folosit euristici penalizatoare pentru a ghida căutarea spre stări promițătoare și a evita cât mai devreme stările imposibile (ex: cutie blocată între pereți sau cutie care este blocata lateral de un perete (is_side_blocked)).

3. Comparația între cei doi algoritmi

Analizele noastre comparative între IDA* și Simulated Annealing evidențiază diferențe semnificative în performanță pe baza mai multor criterii, după cum arată graficele de mai jos.

3.1 Lungimea soluției (număr de mutări)

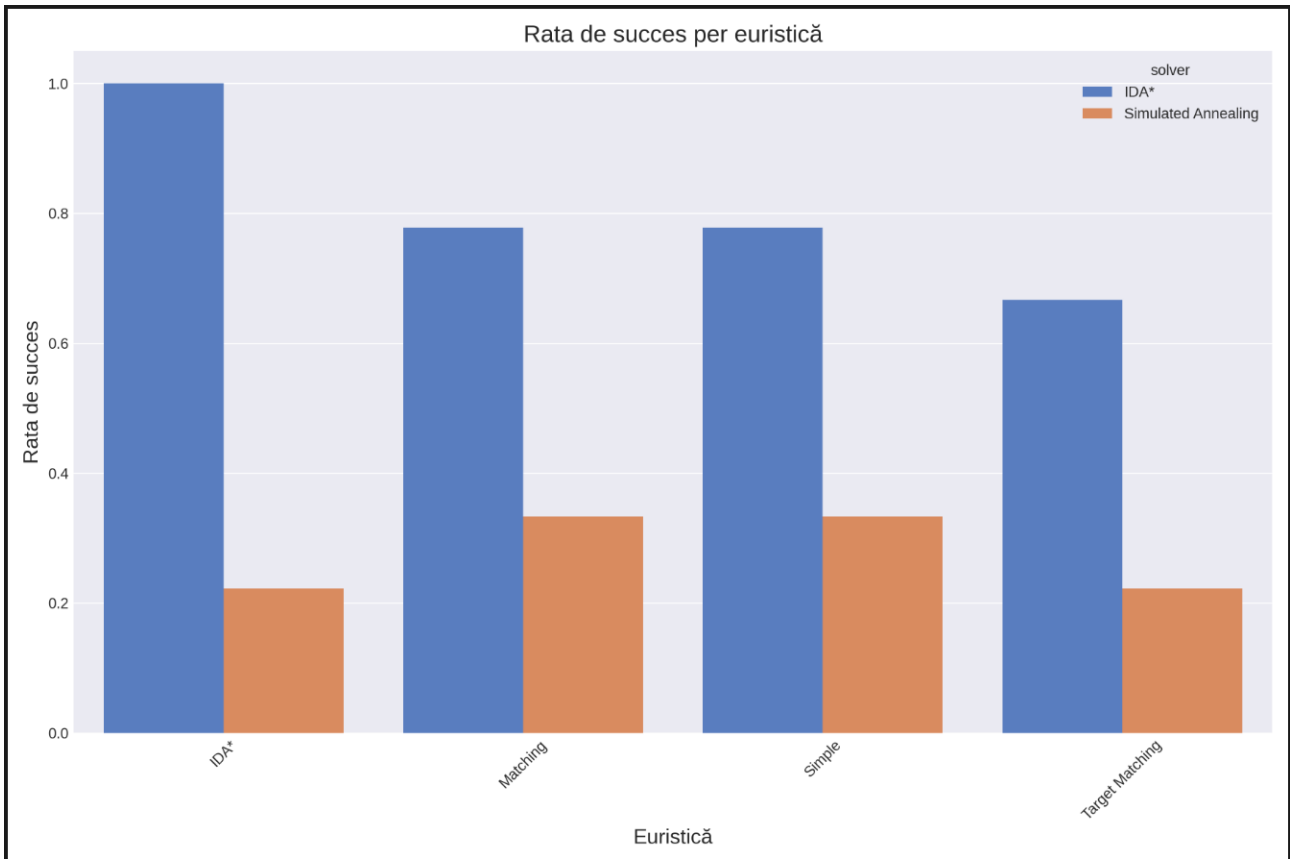


După cum se observă în primul grafic "Lungimea soluției per euristică":

- IDA* generează soluții semnificativ mai scurte (cu mai puține mutări) comparativ cu Simulated Annealing pentru toate cele patru euristici testate: Simple, Matching, IDA* și Target Matching.
- Simulated Annealing produce soluții considerabil mai lungi, în special pentru euristica IDA*, unde diferența este cea mai pronunțată (aproximativ 100 mutări pentru IDA* față de aproape 650 pentru Simulated Annealing).
- Cea mai bună performanță pentru Simulated Annealing se obține cu euristica

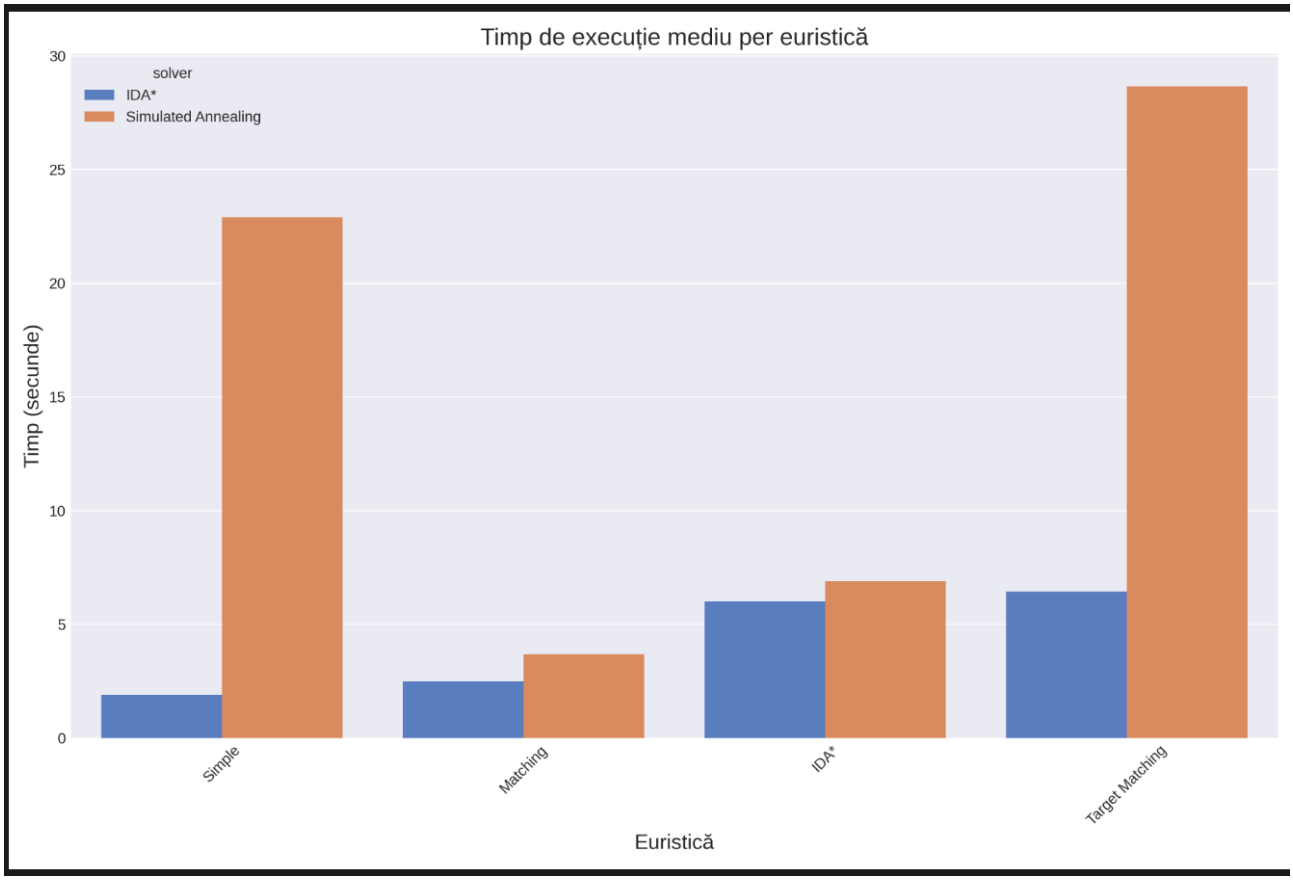
Target Matching (aproximativ 180 mutări).
- Pentru IDA*, diferența între euristici este relativ mică, toate generând în jur de 40-100 mutări.

3.2 Rata de succes:



- IDA* are o rată de succes semnificativ mai mare în găsirea soluțiilor pentru toate euristicile.
- IDA* atinge o rată de succes perfectă (1.0) cu euristica finala aleasa pentru ea și rate de peste 0.75 pentru Matching și Simple.
- Simulated Annealing are rate de succes mult mai reduse, variind între 0.22 și 0.33, ceea ce înseamnă că eșuează în găsirea soluției în majoritatea cazurilor.
- Cea mai bună performanță pentru Simulated Annealing se obține cu euristicile Matching și Simple (aproximativ 0.33).

3.3 Timp de execuție mediu:



- Simulated Annealing are timpi de execuție semnificativ mai mari comparativ cu IDA* pentru toate euristicile.
- Cea mai extremă diferență apare la Target Matching, unde Simulated Annealing necesită aproape 29 secunde față de aproximativ 6 secunde pentru IDA*.
- Pentru euristica Simple, Simulated Annealing necesită aproximativ 23 secunde, în timp ce IDA* finalizează în aproximativ 2 secunde.
- IDA* menține timpi de execuție relativ constanți între 2 și 6 secunde pentru toate euristicile.
- Matching este euristica cu cea mai apropiată performanță între cei doi algoritmi (2.5 vs 3.7 secunde).

3.4 Concluzii generale:

Pe baza graficelor, putem concluziona:

- IDA* este superior Simulated Annealing în toate cele trei aspecte analizate: lungimea soluției, rata de succes și timpul de execuție.
- Contrar așteptărilor inițiale, Simulated Annealing nu doar că generează soluții de calitate inferioară (mai lungi), dar și consumă mai mult timp.
- Euristica are un impact semnificativ asupra performanței ambilor algoritmi, dar IDA* este mai puțin sensibil la alegerea euristicii.
- Target Matching oferă cele mai scurte soluții pentru Simulated Annealing, dar cu cel mai mare cost de timp.
- Euristica Matching pare să ofere cel mai bun echilibru pentru Simulated Annealing între timp și calitatea soluției.

Aceste rezultate sugerează că pentru problema Sokoban, caracterul exhaustiv și deterministic al IDA* îl face mai eficient, probabil datorită naturii structurate a spațiului de stări din acest joc. Simulated Annealing, deși teoretic capabil să evite minimele locale, nu reușește să exploreze eficient

spațiul de soluții pentru această problemă specifică.

O posibilă explicație este că problema Sokoban beneficiază mai mult de o explorare sistematică decât de una aleatorie, iar funcțiile euristice folosite ghidează mai eficient algoritmul IDA* decât controlul temperaturii din Simulated Annealing.

4. [Bonus] Dezvoltarea de euristici informate

- Rezolvarea jocului fără miscari de pull (a fost comentat în map.py partea de cod care permite aceste mutari).

- Am combinat distanțe, blocaje, deadlock-uri și mișcările jucătorului într-o singură funcție de cost.

- Am adăugat o penalizare pentru numărul de stări explorate în timpul jocului, astfel încât să încurajez soluțiile rapide, fără oscilații inutile.

- Am aplicat penalizări variabile în funcție de cât de "grav" este un blocaj: colț → penalizare mare; tunel → penalizare medie.

Observație:
Chiar dacă nu am obținut soluții perfecte, am observat îmbunătățiri clare față de variantele naive.