Tema 1 IA – Implementarea IDA* și Simulated Annealing pentru Sokoban

Tunaru Ștefania Emilia

331 CB

1. Jurnalizarea ideilor pentru euristici și algoritmi

Tema a presupus implementarea celor doi algoritmi studiați în cadrul cursului de Inteligență Artificială, IDA* și Simulated Annealing, alături de dezvoltarea de euristici compatibile cu aceștia pentru rezolvarea jocului Sokoban, descris în cerința temei și în scheletul de cod.

Am început rezolvarea temei prin crearea fișierelor corepunzătoare, heuristics.py, ida_star.py și simulated_annealing.py.

- **Implementarea inițială a funcției euristice**

Am început implementarea funcției euristice utilizând distanța Euclidiană, preluată din laboratorul 2, pentru a o utiliza în primă instanță drept euristică în algoritmii de căutare în spațiul stărilor. Am adaptat funcția din laborator astfel încât să calculeze într-o manieră greedy, după cum era precizat în hint-ul din cerința temei. Am calculat astfel distanța minimă dintre o cutie și o destinație, fiecare cutie fiind asociată cu cea mai apropiată destinație disponibilă, în cazul hărților ce aveau mai multe cutii și destinații. De asemenea, am ținut cont de asocierile deja făcute pentru a nu a atribui aceeași detinație mai multor cutii, evitând astfel și conflictele de ciocnire. După ce am implementat funcția euristică am trecut la implementarea algoritmilor de căutare.

- **Implementare IDA***

Am început implementarea algoritmilor IDA* și Simulated Annealing, urmărind resursele din cursuri/laboratoare/cerință. Astfel, pentru implementarea algoritmului IDA* am utilizat link-ul pus la dispoziție în cerința temei **[1]**, urmărind pseudocodul algoritmului prezentat la finalul articolului, și adaptându-l la scheletul de cod pentru a funcționa pentru jocul de Sokoban. Am folosit în cadrul algoritmului euristica dezvoltată anterior.

După testare, algoritmul găsea soluții doar pentru hărțile easy, iar pentru harta easy_1 rularea lui dura mai mult, creând mai multe stări din cauza complexității amplasării obiectelor pe hartă, făcând căutarea mai dificilă comparativ cu harta easy_2. Am decis astfel să optimizez funcția euristică, prin abordarea aceleiași strategii de tip greedy, pentru a calcula distanța minimă de la player la fiecare cutie. După o căutare pe internet despre optimizarea funcțiilor euristice am aflat de tehnica de înmulțire a rezultatului euristicii cu un factor (weight > 1) care să pondereze funcția și să acorde prioritate mișcării cutiilor spre destinațiile asignate decât jucătorului către cutii **[2]**. Aceste optimizări ale euristicii nu au contribuit la optimizarea algoritmului și, prin urmare, nici la găsirea soluțiilor pe restul hărților. Playerul se bloca în minime locale și explora repetitiv aceleași stări, algoritmul rulând la infinit.

Am decis astfel să caut tehnici de optimizare ale algoritmului IDA*, am citit pe internet despre dezavantajele acestuia **[3]** și am decis să implementez un mecanism de monitorizare a stărilor deja

vizitate în cadrul căutarii pentru a reduce deplasările redundante. Acest mecanism a fost implementat cu ajutorul unui dicționar în cadrul căruia am reținut un tuplu fomat din poziția jucătorului și pozițiile cutiilor sortate pentru a asigura unicitatea instanțelor. Verific dacă starea a fost deja vizitată cu un cost egal sau mai mic, iar în cazul în care starea este nouă sau a fost vizitată cu un cost mai mare, aceasta este reînregistrată cu costul curent (costul minim de până acum). Dicționarul este resetat la fiecare iterație a algoritmului pentru a permite regăsirea unor stări cu un threshold mai mare la fiecare iterație.

După această optimizare a algoritmului, am rulat testele pe rând, obținând rulări foarte rapide și puține stări explorate pentru testele easy-medium, însă începând cu hărțile large, timpul de execuție al algoritmului depășea 5 minute, soluția fiind găsită cu greu din cauza complexității spațiului de căutare, mai ales din cauza poziționării inconvenabile a obstacolelor. Începând cu harta hard_2, algoritmul rămânea blocat, fapt ce m-a determinat să îmi regândesc strategia euristicii.

Am inspectat din nou conținutul laboratorului 2, și mi-am amintit de faptul că euristica ce utiliza distanța Manhattan era mai rapidă decât cea ce utiliza distanța euclidiană pentru problema prezentată în laborator, așa că am decis să implementez o astfel de euristică. Am pronit din nou de la informațiile din laborator, adaptându-le pentru jocul Sokoban. Folosind funcția euristică euclidiană construită anterior, am înlocuit distanța euclidiană cu cea Manhattan, păstrând optimizările deja existente ale euristicii mele, și am înlocuit-o și în algoritmul IDA*. În urma rulării, atât numărul de stări, cât și timpii de execuție au fost mici, rezultând într-un rezultat mulțumitor, algoritmul meu a reușit să găsească soluții pentru fiecare dintre hărți.

Am încercat apoi limitarea mișcărilor de pull, prin modificarea fișierului map.py din directorul ”sokoban” al scheletului de cod, înlocuind toate instanțele de BOX_DOWN cu DOWN. Algoritmul a rulat fără probleme având în vedere chiar și această limitare, și a generat soluții optime fără utilizarea mișcărilor de pull pe toate hărtile de input.

- **Implementare Simulated Annealing**

Pentru implementarea algoritmului Simulated Annealing, am eliminat limitarea mișcărilor de pull, apoi am parcurs laboratorul 3 în care am studiat implementarea algoritmului și metoda de optimizare prin softmax. Am preluat astfel codul din laborator și l-am ajustat pentru a putea funcționa în cadrul temei cu scopul de a putea rezolva jocul de Sokoban, alături de optimizarea oferită de funcția softmax. În implementarea algoritmului Simulated Annealing, în care am utilizat euristica dezvoltată anterior ce utiliza distanța Manhattan, funcția softmax este utilizată pentru a transforma costurile calculate prin euristică în probabilități de selecție pentru nodurile vecine. Softmax primește valorile negative ale costurilor (-costs) împărțite la temperatura curentă și factorul alpha, convertindu-le într-o distribuție de probabilități, această abordare probabilistică fiind esențială pentru echilibrul între explorare și exploatare. La temperaturi ridicate, distribuția este mai aproape uniformă, permițând o explorare mai largă, în timp ce la temperaturi scăzute, distribuția favorizează nodurile cu cost minim. Astfel, nodurile cu cost mai mic au șanse mai mari să fie selectate, dar algoritmul păstrează capacitatea de a explora ocazional stări aparent suboptimale pentru a evita blocarea în minime locale. Selecția finală se realizează probabilistic, ceea ce permite algoritmului să exploreze diverse căi spre soluție.

La o primă rulare, în ciuda optimizării aduse de softmax, algoritmul realiza prea multe mișcări redundante care nu duceau la soluție, explorând aceleași stări de foarte multe ori și realizând un număr mare de mișcări de pull, rezultând astfel un timp de execuție îndelungat. După ce am testat algoritmul pe ambele hărți de tip easy, deși algoritmul a găsit soluția pentru ambele, următoarele teste (medium) aveau un timp de execuție mult prea îndelungat, generând mult prea multe stări, și prin urmare multe imagini pentru gif-ul final. Pentru restul hărților, algoritmul genera 0 stări, 0 mișcări de pull și nu găsea nicio soluție. Așadar, am decis că este nevoie de o optimizare mai avansată a algoritmului pentru a genera rezultatele de rulare dorite.

Astfel, drept o primă optimizare m-am inspirat din cursul Local Search, în care era menționat faptul că se pot folosi reporniri prin revenirea la stări de energie mică obținute în trecut, dacă am parcurs un anumit număr de iterații cu energie semnificativ mai mare decît minima găsită. Astfel, am implementat un mecanism periodic de verificare la fiecare 100 de iterații. Algoritmul compară costul stării curente cu cel mai bun cost identificat anterior, iar dacă acesta s-a deteriorat (current_cost > best_cost), resetează complet căutarea la cea mai bună stare găsită până în acel moment. Se copiază starea optimă, costul acesteia și secvența de mutări corespunzătoare. Această tehnică eficientă permite algoritmului să evite blocarea în explorări neproductive și să-și concentreze eforturile în regiuni mai promițătoare ale spațiului de căutare.

În ciuda acestei optimizări, încă au existat hărți pentru care algoritmul nu găsea soluție, respectiv easy_1, medium_2, large_1 și hard_2 și super_hard, așadar, pe hărți care necesitau secvențe mai complexe de mutări prin amplasamentul obstacolelor și existența coridoarelor ce limitau accesul cutiilor.

Am continuat astfel să caut soluții pentru optimizare algoritmului meu. Utilizând informații din mai multe surse am decis să abordez implementarea de restarturi pe care am înțeles-o dintr-o lucrare stiințifică găsită pe internet **[4]**. De asemenea, m-am inspirat și din hint-urile unei implementări Sokoban găsită pe GitHub **[5]**. Așadar, strategia de restarturi execută algoritmul de 20 ori independent, pornind de la aceeași configurație inițială, dar explorând căi diferite datorită naturii probabilistice a selecției vecinilor. După fiecare execuție completă, algoritmul reține doar soluția validă cu cel mai mic număr de mutări, maximizând astfel șansa de a descoperi soluția optimă globală prin explorarea diversificată a spațiului de căutare și evitarea blocării în minime locale.

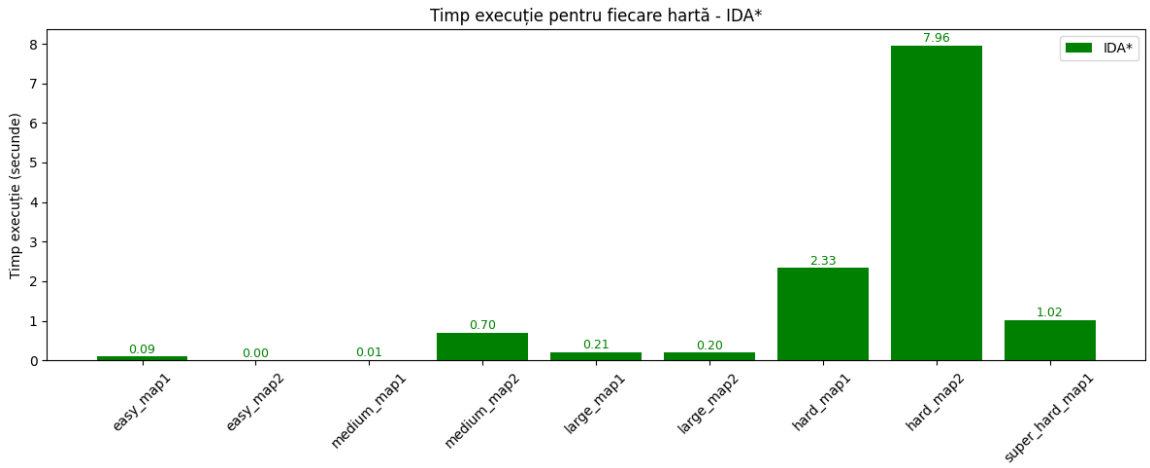
În continuare, după aceste modificări, algoritmul a reușit să găsească o soluție și pentru hărțile easy_1, medium_2 și large_1, însă tot nu funcționa pentru harta hard_2 și super_hard. Am decis să încerc modificarea parametrilor algoritmului, valoarea acestora fiind inițial preluată din laborator, încercând să restricționez comportamentul explorator al jucătorului doar în prima parte a execuției algoritmului. Am modificat valoarea temperaturii inițiale de la 700 la 10000 pentru a permite jucătorului să exploreze mai agresiv la început, făcând mișcări aparent irațională, împingând cutii în direcții neașteptate. Scăderea temperaturii finale de la 1.0 la 0.01 face ca, spre final, jucătorul să devină extrem de precis, selectând doar mișcări care optimizează direct pozițiile cutiilor. Creșterea ratei de răcire de la 0.995 la 0.999 menține comportamentul explorator al jucătorului pentru mult mai multe iterații, permițându-i să testeze secvențe variate de mutări înainte de a se angaja la o strategie. Modificarea alpha de la 0.02 la 0.97 face ca jucătorul să accepte mai ușor mutări temporar suboptimale, împingând cutii în direcții aparent greșite pentru a debloca configurații viitoare mai avantajoase. Modificarea parametrilor a transformat

comportamentul jucătorului dintr-o strategie directă și prudentă într-una mult mai exploratorie și flexibilă, dispusă să facă sacrificii pe termen scurt prin mutări aparent irațională pentru a descoperi soluții neconvenționale și a evita blocajele în puzzle-urile complexe, iar aceste modificări au dus la succesul găsirii unei soluții pe toate hărțile.

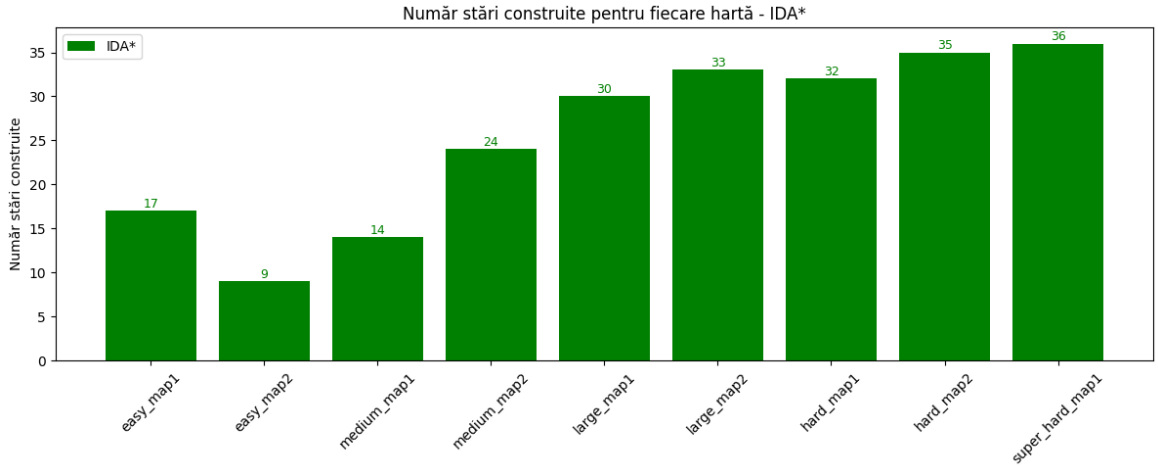
Din păcate, algoritmul meu Simulated Annealing nu funcționa fără mișcări de pull atunci când am încercat să îi impun această limitare, decât pe primele trei hărți. Am încercat construirea unei euristici care să folosească penalizări pentru mutările care duceau la blocarea cutiilor, respectiv la necesitatea mișcărilor ulterioare de pull, însă acesta nu a funcționat decât pe câteva dintre hărți și nu am reușit să găsesc o soluție pentru a-l face să funcționeze în totalitate.

2. Rezultatele rulării IDA* fără limitarea mișcărilor de pull

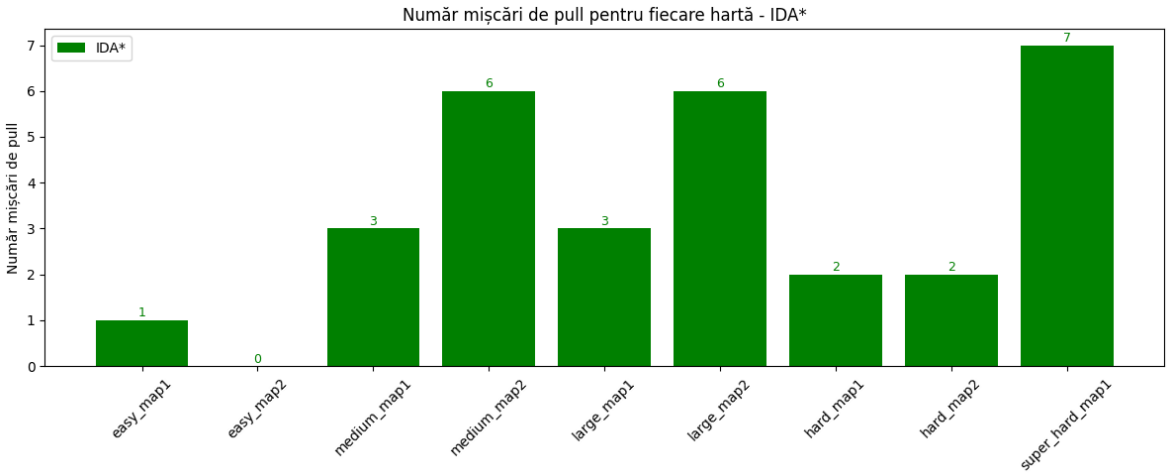
- **Din punctul de vedere al timpului de execuție**



- **Din punctul de vedere al numărului de stări parcurse**



- **Din punctul de vedere al numărului de mișcări de pull**

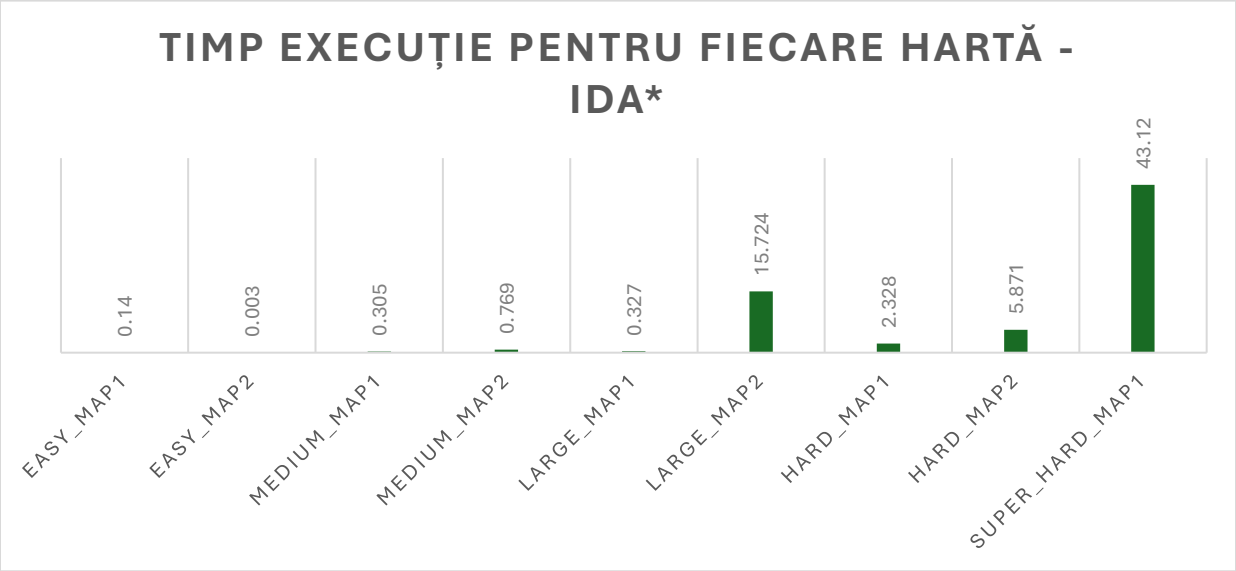


Concluzii:

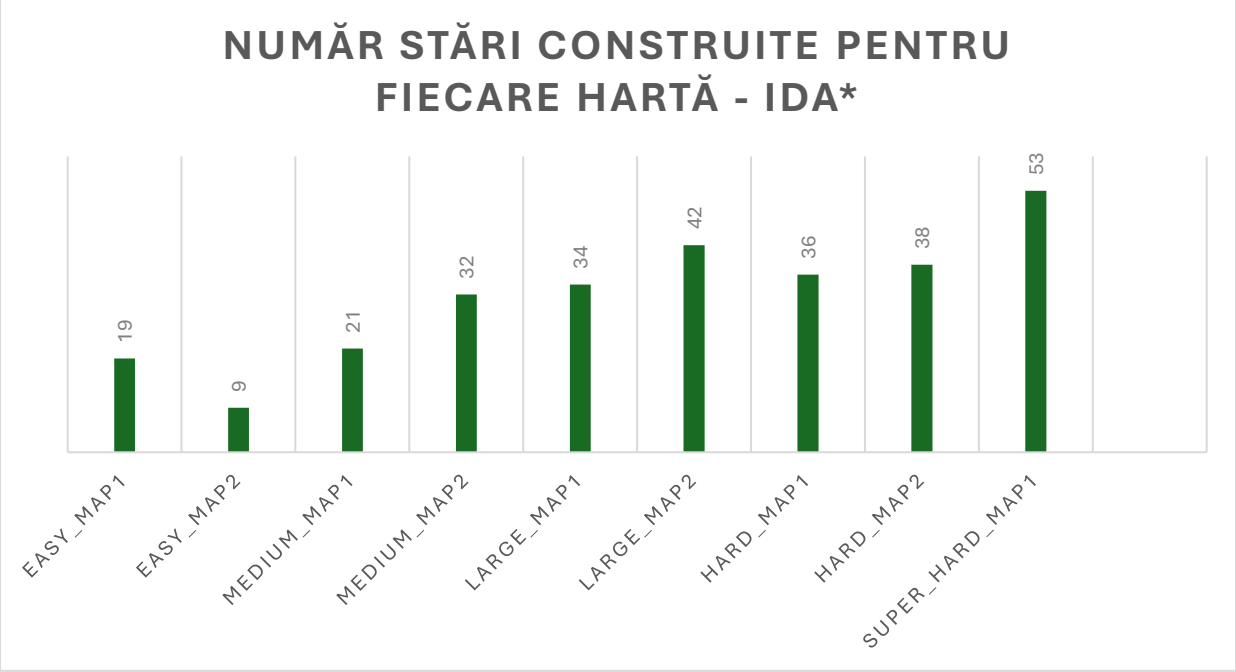
- Pe hărțile de tip easy și medium, IDA* a avut timpi de execuție foarte mici, datorită complexității scăzute a problemelor și a spațiului restrâns de căutare. Pe hărțile de tip large și hard, timpul de execuție a crescut moderat, însă datorită folosirii mișcărilor de pull, a reușit să evite blocajele și a ajuns la un timp de execuție mulțumitor. Pe super_hard_map1, timpul de execuție a fost considerabil, dar în parametrii acceptabili având în vedere dificultatea hărții.
- În mod similar, numărul de stări generate a fost mic pe hărțile de tip easy și medium, a crescut pe hărțile de tip large și hard și a fost foarte mare pe super_hard_map1. Acest comportament reflectă creșterea normală a numărului de stări explorate odată cu sporirea complexității mediului explorat.
- Soluțiile au conținut un număr rezonabil de mișcări de pull. IDA* a apelat la pull doar când era necesar să evite blocaje, fără a abuza de această opțiune.

3. Rezultatele rulării IDA* cu limitarea mișcărilor de pull

- **Din punctul de vedere al timpului de execuție**



- **Din punctul de vedere al numărului de stări parcurse**

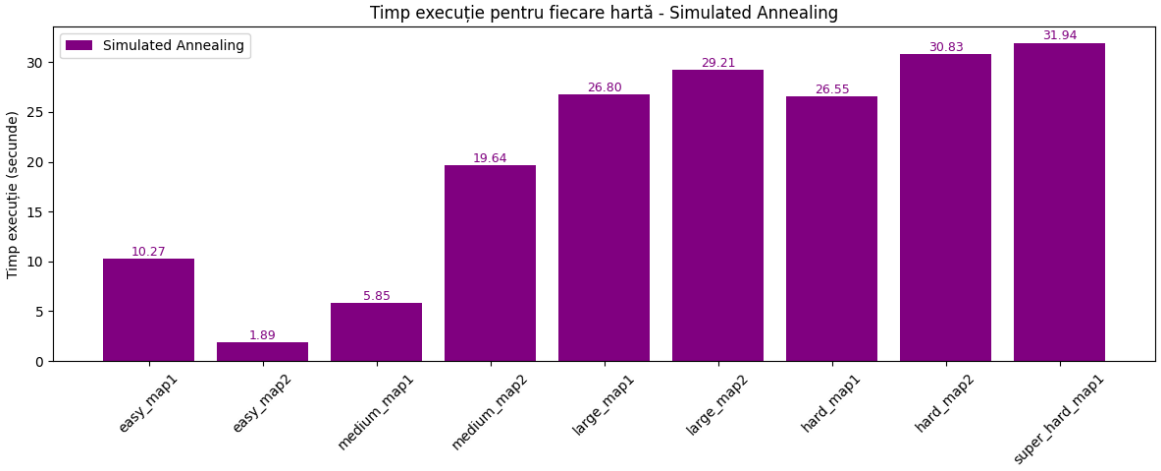


Concluzii:

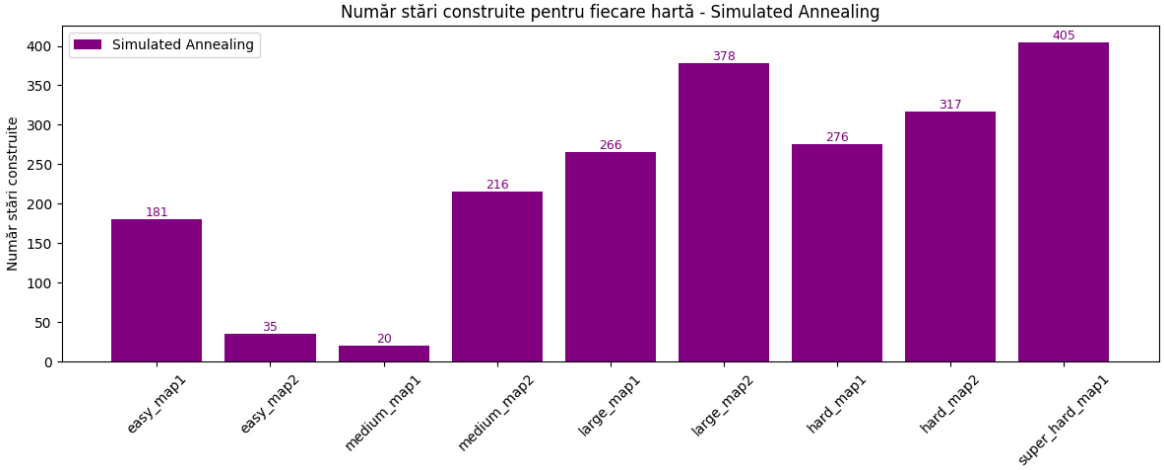
- Fără posibilitatea de a realiza mișcări de pull, timpul de execuție a crescut, în special pe hărțile de tip large și hard, unde traseele alternative sunt mai greu de găsit. Cu toate acestea, IDA* a reușit în majoritatea cazurilor să găsească soluții în timpul impus.
- Numărul de stări parcurse a crescut considerabil, deoarece lipsa pull-ului a restricționat posibilitățile de mișcare și a forțat algoritmul să caute trasee mai lungi.

4. Rezultatele rulării Simulated Annealing fără limitarea mișcărilor de pull

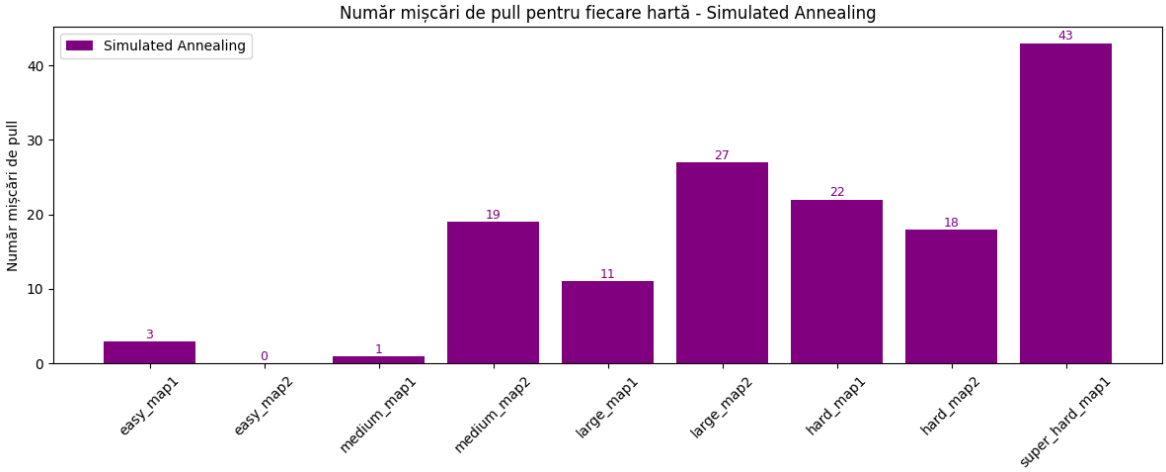
- **Din punctul de vedere al timpului de execuție**



- **Din punctul de vedere al numărului de stări parcurse**



- **Din punctul de vedere al numărului de mișcări de pull**

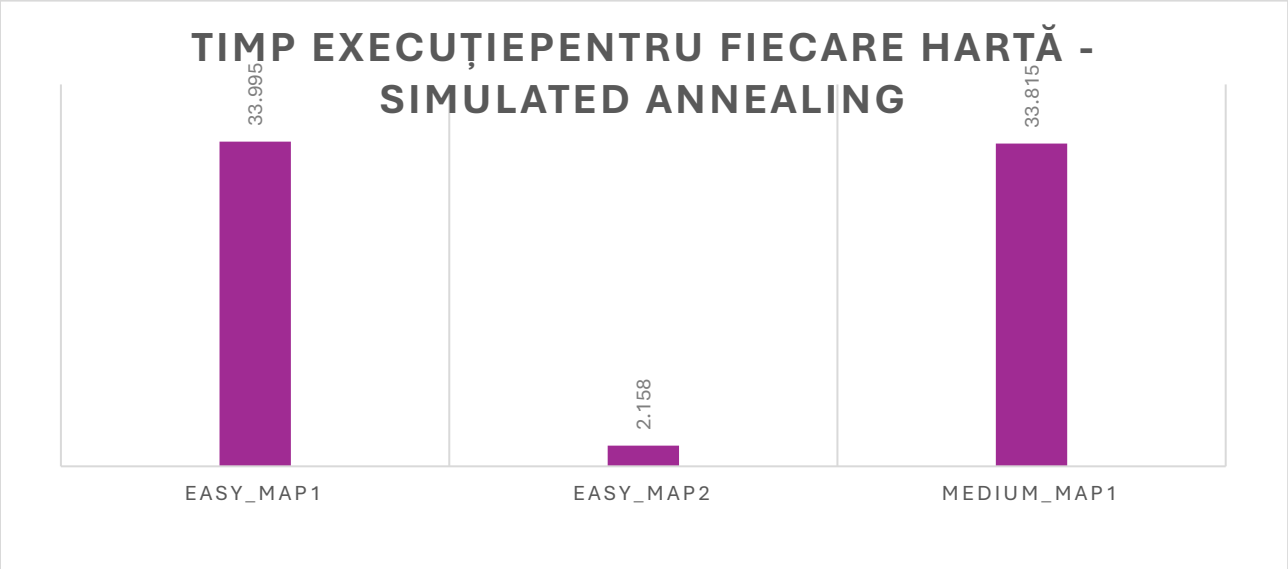


Concluzii:

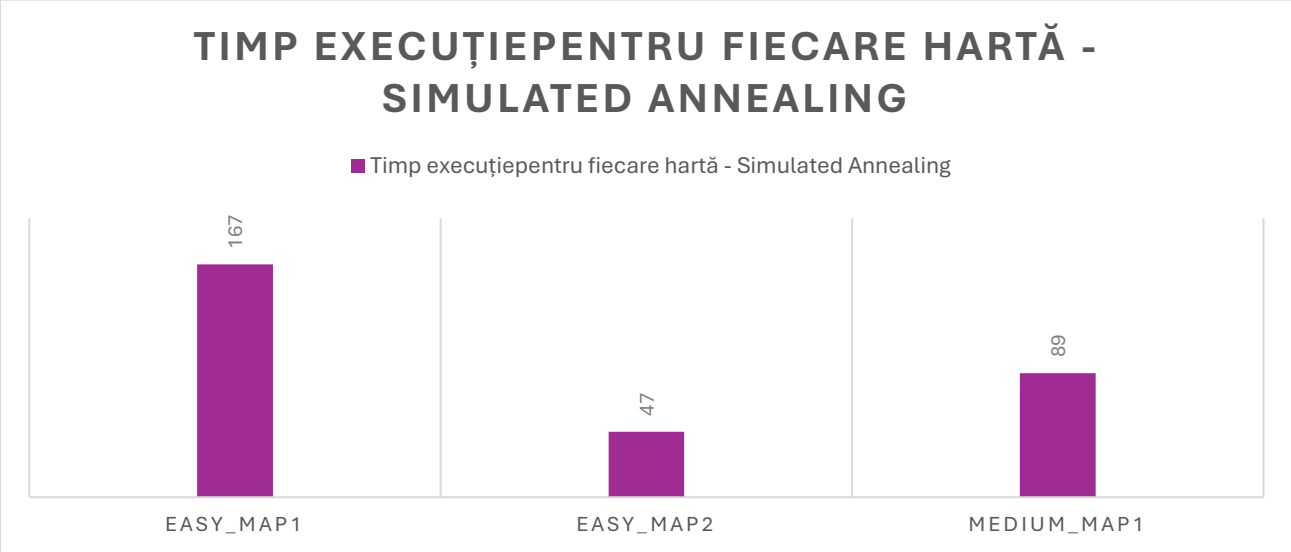
- Algoritmul Simulated Annealing a avut timpi de execuție mai mari față de IDA*, mai ales pe hărțile de tip medium, hard și pe harta super_hard1. Natura probabilistică a algoritmului l-a determinat să exploreze multe trasee suboptimale înainte de a ajunge la o soluție.
- Numărul de stări explorate a fost mult mai mare comparativ cu IDA*, în special pentru hărțile de dificultate ridicată, ceea ce a fost de așteptat din cauza explorării aleatorii.
- Soluțiile produse de Simulated Annealing au implicat mai multe mișcări de pull decât cele ale IDA* din cauza accentului pus pe explorare.

5. Rezultatele rulării Simulated Annealing cu limitarea mișcărilor de pull

- **Din punctul de vedere al timpului de execuție**



- **Din punctul de vedere al numărului de stări parcurse**

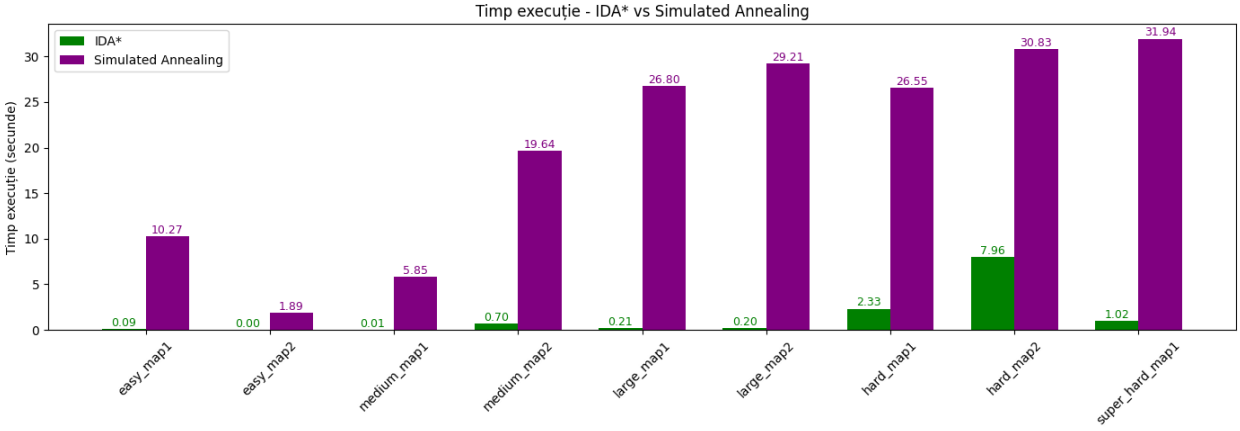


Concluzii:

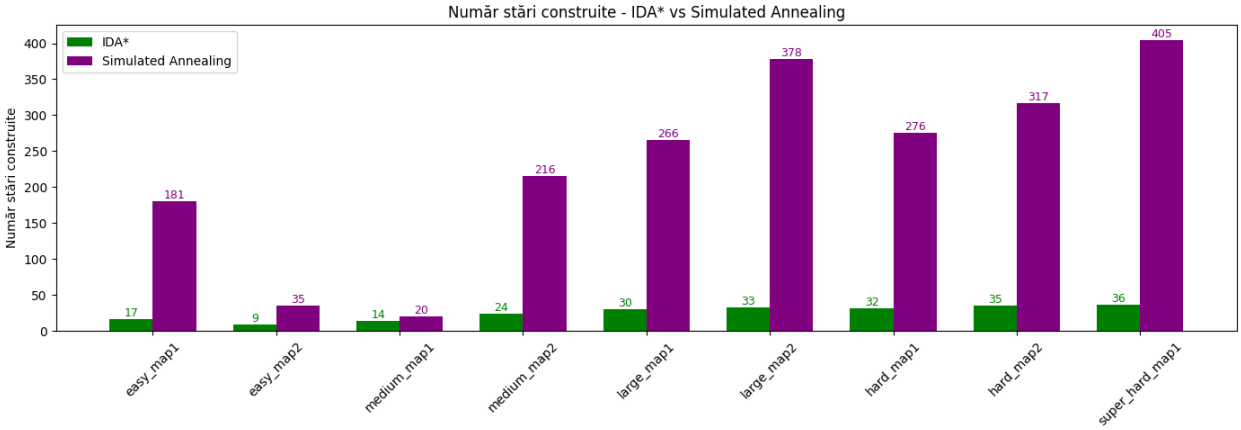
- Limitarea mișcărilor de pull a afectat dramatic performanța algoritmului Simulated Annealing. Începând chiar de la hărțile de tip medium, algoritmul nu a reușit să găsească soluții.
- Numărul de stări parcurse a crescut drastic, reflectând dificultatea algoritmului de a găsi soluții fără ajutorul flexibilității oferite de pull.

6. Compararea rulării celor doi algoritmi

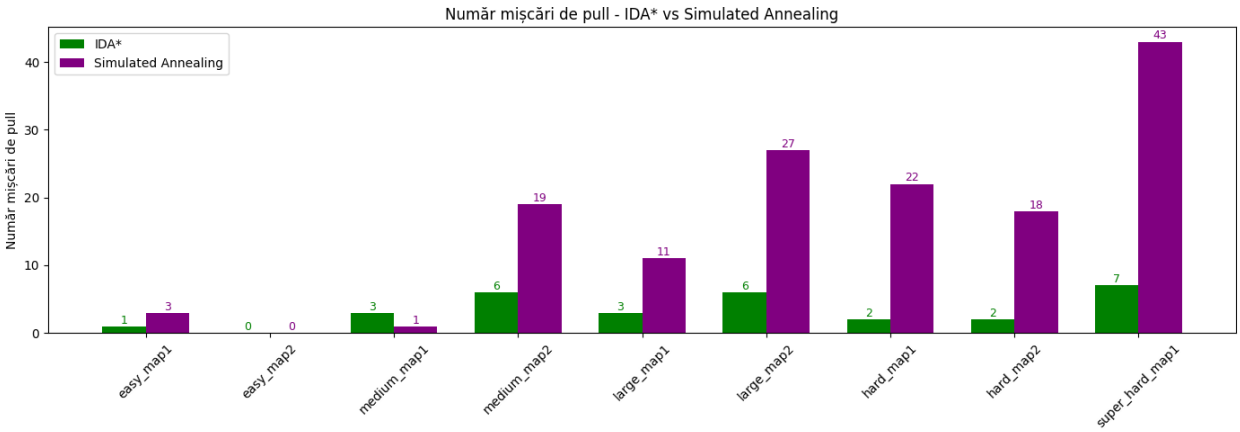
- **Din punctul de vedere al timpului de execuție**



- **Din punctul de vedere al numărului de stări parcurse**



- **Din punctul de vedere al numărului de mișcări de pull**



Concluzii:

- IDA* este în mod evident mult mai eficient ca timp pe toate hărțile, în comparație cu Simulated Annealing.
- Simulated Annealing a generat mult mai multe stări, mai ales pe hărțile complexe (hard, large, super_hard). IDA* a rămas constant mai eficient în gestionarea spațiului de căutare.
- IDA* a produs soluții cu mai puține pull-uri, în special după optimizarea euristicii și implementarea strategiei de evitare a stărilor deja vizitate. Simulated Annealing, fiind un algoritm probabilistic, a generat soluții mai greoaie, cu multe mișcări de pull.

În concluzie, algoritmul IDA* este mai potrivit pentru Sokoban decât algoritmul Simulated Annealing, oferind un compromis excelent între viteză, calitatea soluției și număr de stări explorate. Algoritmul Simulated Annealing poate aduce soluții surprinzătoare, mai ales în cazurile extrem de complexe, dar necesită multă ajustare din punctul de vedere al euristicii utilizate în cadrul acestuia și este mai puțin eficient în general.

RESURSE

[1] - IDA-Star(IDA*) Algorithm in general « Insight into programming algorithms

[2] - https://www.sciencedirect.com/science/article/pii/S0004370215000545

[3] - https://webdocs.cs.ualberta.ca/~holte/Publications/fringe.pdf

[4] - https://www.mdpi.com/2227-7390/9/14/1625

[5] - https://github.com/gabrielarpino/Sokoban-Solver-AI/blob/master/tips.txt