Inteligenta Artificiala: Tema 1 – Sokoban

1. Introducere

Sokoban este un joc puzzle in care jucatorul trebuie sa impinga toate cutiile de pe o harta pe anumite pozitii semnalizate ca tinte. Jocul clasic include doar operatia de impingere a cutiilor, ceea ce poate fi problematic prin faptul ca anumite operatii sunt ireversibile. Implementarea temei include si functionalitatea de tragere a cutiei, ceea ce scapa de problema anterioara.

2. Algoritmi

➉ IDA* (Iterative Deeepening A*)

Iterative Deepening A* (IDA*) este un algoritm de cautare informata pe grafuri, realizat pentru a gasi cel mai scurt drum intr-un graf echilibrat. Metoda combina **algoritmul clasic A*** cu **algoritmul DFS**. Mai exact sunt folosite o functie euristica de aproximare a drumului ce prioritizeaza drumurile cu un cost mai bun (A*) cu o cautare in adancime (DFS). Toata ideea algoritmului este pragul threshold folosit pentru a stii cat de in adancime trebuie sa cautam solutia. Acest prag porneste de la costul aproximat de inceput si este reglat la fiecare pas, si este marit daca raman doar drumuri scumpe, fara sa fi gasit solutia.

➉ Euristica IDA* si etapele de implementare

Am pornit pentru inceput de la euristica bazata pe **distanta Manhattan**, deoarece ne aflam pe harta ce implica deplasarea pe axe de coordonate, adica avem posibile doar 4 mutari (sus-jos, stanga-dreapta). Repede am realizat faptul ca in acest joc, distanta Manhattan nu aproximeaza corect costul daca este aplicata simplu intre cutii si tinte din mai exact 2 motive: **player-ul poate sa se deplaseze pe pozitii mai indepartate de cutii, distanta cutie-tinta ramanand aceeasi** si **atribuirea fiecarei cutii pe o tinta se poate face prost** (ex: 2 cutii atribuite pe aceeasi tinta daca se afla la distante egale de aceeasi tinta).

Am realizat aceste schimbari, ceea ce a dus la timpi mult mai buni pe teste. Am facut de la bun inceput o **asociere clara cutie-tinta,** asociind fiecarei cutii cea mai apropiata tinta disponibila. De asemenea, in euristica folosita, costul meu implica **mai multe distante Manhattan: suma tuturor distantelor player-cutie si suma tutror distantelor cutie-tinta**.

In incercarea de a trece testele hard si super-hard, am incercat modificarea euristicii de asocierie cutie-tinta prin **realizarea tutror permutarilor tupletilor (cutie, tinta)** si selectarea celei mai bune sume. De asemenea, am incercat **schimbarea euristicii Manhattan in BFS, deoarece BFS ia in considerare si blocajele de pe traseu, in timp ce Manhatta nu**.

➉ Beam-Search

Beam-Search este un algoritm euristic de cautare informata pe grafuri in care este folosita tehnica BFS limitata. L afiecare treapa a BFS, in loc sa luam in calcul toate ramurile din acel punct, alegem sa **cercetam doar cele mai bune beam-width dintre ele**. Acest agoritm este foarte util in sisteme mari cu foarte multe stari, unde memoria este limitata. Din pacate,

acest algoritm nu este exact, putem pierde solutia optima , daca aceasta se afla pe una din ramurile ignorate la un moment dat.

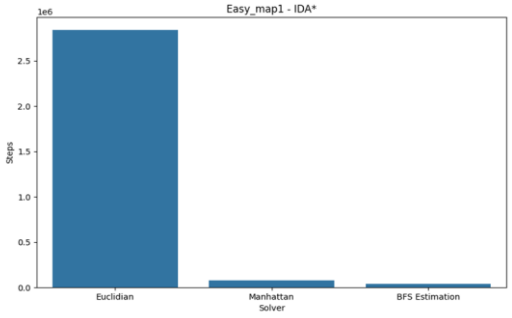
❿ Euristica Beam-Search si etapele de implementare

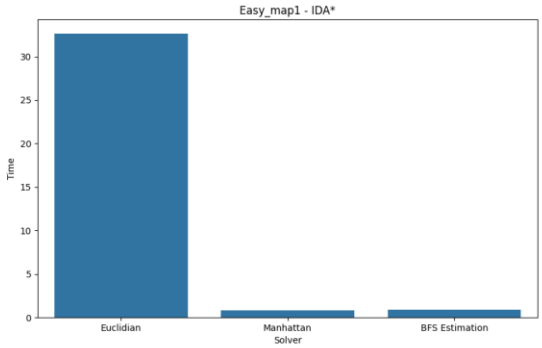
Am pornit pentru inceput de la euristica folosita la IDA* prin care calculez costul ca fiind **suma tuturor distantelor Manhattan player-cutie si suma tutror distantelor cutie-tinta**. Dupa aceea am realizat un BFS limitatat la beam_width ramuri, la fiecare etapa. La acest algoritm am incercat si prioritizarea fiecarei cutii in parte, adica **player-ul sa rezolve e rand fiecarea cutie, separat**. De asemenea, **dupa ce finalizam plasarea unei cutii pe o tinta, o marcam ca finalizata** si o eliminam din lista de cutii luate in considerare la calcularea costurilor.

Algorimtul si euristica au dat rezultate foarte bune, rezolvand intr-un timp mai mult decat rezonabil toate testele, inclusiv cele hard si super-hard.

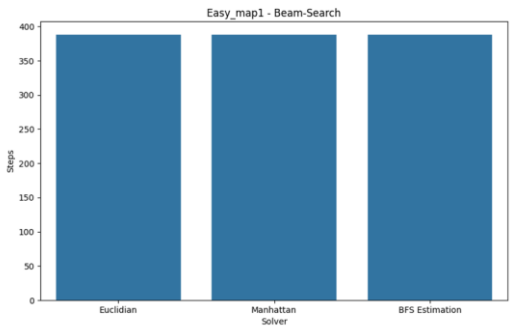
3. Date si tabele

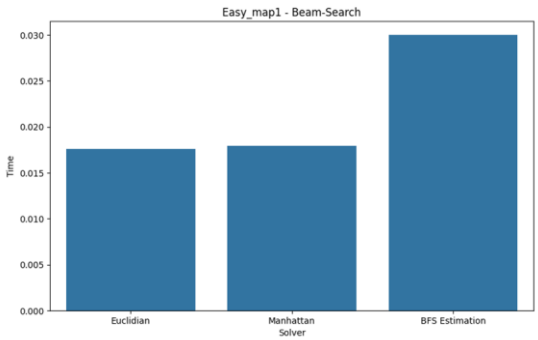
Pentru algoritmul IDA*, se poate observa cum distanta euclidiana este foarte costisitoare in timp si numar de pasi fata de BFS si Manhattan care sunt aproximativ egale.





Pentru Beam-Search se poate observa cum el functioneaza cam la fel pentru toate euristicile. Asta se datoreaza faptului ca noi limitam din start numarul de ramuri pe care le cercetam. Totusi se poate observa cum timpul e mai mare pentru euristic abfs decat pentru celelalte deoarece la bfs chiar parcurg aborele in timp ce la euclidian si manhattan doar fac un clacul.





Daca compara algoritmii intre ei, putem observa ca Beam-Search are mai putini pasi, fapt ce se datoreaza limitarii impuse de beam_width.

4. Referinte

https://en.wikipedia.org/wiki/Iterative_deepening_A*
https://en.wikipedia.org/wiki/Beam_search
https://www.geeksforgeeks.org/iterative-deepening-a-algorithm-ida-artificial-intelligence/
https://www.geeksforgeeks.org/iterative-deepening-a-algorithm-ida-artificial-intelligence/