Nume:Logofatu Patricia

Grupa:335CC

1.Reprezentarea unei stari a problemei-Sokoban

Pentru rezolvarea temei am considerat initial abordarea jocului Sokoban ca o problema de cautare in spatial starilor.O stare a fost reprezentata prinr-o instanta a clasei ***Map,*** definita in scheletul oferit.

- *Componentele unei stari:*

Fiecare obiect de tip Map contine:

- Pozitia jucatorului - pastrata sub forma unui obiect Player ce include coordonatele (x,y) ale acestuia;
- Pozitiile cutiilor - sub forma unui dictionar de obiecte Box;
- Pozitiile tintelor - o lista de tupluri (x,y) corespunzatoare locatiilor finale ale cutiilor;
- Pozitiile peretilor - memorate in lista walls_pos, cu tupluri ce defines zonele inaccesibile.
  Acestea sunt folosite, de exemplu, in functiile euristice pentru a detecta stari de deadlock, adica unde o cutie este blocata definitiv intr-un colt.

- Pentru a genera succesorii/pentru a explora spatial starilor, am folosit metoda *filter_possible_moves() ,* care returneaza doar acele mutari valide intr o anumita stare, adica cele care nu duc in pereti sau in pozitii imposibile.
- Fiecare mutare este aplicata cu *apply_move(),* care returneaza o noua stare rezultata din aplicarea mutarii respective.
- Pentru a pastra integritatea starii curente, am folosit metoda copy(), care produce o copie independenta a starii, esentiala pentru algoritmii de cautare care evalueaza mai multi vecini in paralel.

2. LRTA*

- In implementarea algoritmului LRTA* am folosit o abordare pas cu pas, in care jucatorul exploreaza spatial starilor si isi actualizeaza functia euristica pe masura ce inainteaza.Ca functionare, la fiecare pas folosesc metoda *filter_possible_moves()* pentru a determina mutarile valide, adica acelea care nu

duc in pereti sau in pozitii imposibile.Aplic fiecare mutare cu *apply_move()* si evaluez starea rezultata printr-o functie euristica. Retin valorile in dictionarul **H**, folosind cheia *str(state)* pentru a putea actualiza ulterior valoarea starii curente conform fomulei. In final, selectez starea succesoare cu cel mai mic cost estimate si continui explorarea de acolo.

- Optimizarea adusa algoritmului o reprezinta faptul ca am impus o limita de explorare *(max_iterations)* pentru a preveni blocarea in bucle infinite.Am adaugat un set de stari vizitate (*visited_states*) pentru a detecta si evita ciclurile.

3. Simulated-Annealing

- Acest algoritm este folosit pentru optimizarea functiilor intr-un spatiu mare si neregulat, iar in implementarea lui am definit cativa pasi principali.
  -Am inceput cu initializarea starii curente (harta de pornire) si o temperatura ridicata (*initial_temp)*.

  -Mai departe, la fiecare iteratie se genereaza un vecin al starii curente folosind *filter_possible_moves()* si *apply_move(),* se calculeaza scorul noii stari;daca noua stare are un scor mai bun, este acceptata automat,iar daca are un scor mai slab, este acceptata cu o probabilitate ce depinde de temperature curenta(pentru a permite explorarea).

  -Temperatura se raceste gradual prin *cooling_rate*, reducand astfel probabilitatea de a accepta mutari suboptime pe masura ce algoritmul avanseaza.
- In varianta implementata, am folosit o strategie de selectie a vecinilor denumita *“weighted”* , in care mutarile de tip push sunt preferate in proportie de 80% fata de cele de tip pull, pentru a ghida algorimul spre solutii curate si eficiente.Daca nu exista mutari de tip push disponibile, se recurge la selectie aleatorie.Aceasta abordare simplificata a contribuit la reducerea mutarilor pull si la obtinerea unor trasee mai bune in unele instante.

4. Descrierea euristicilor

- ***manhattan_distance*** – pt a estima costul de deplasare intre 2 puncte(cutie->tinta, jucator->cutie etc.)

Am testat individual aceasta eurstica pe harta *easy_map1.yaml*, unde s-a comportat correct si efficient.Desi are o estimare simpla, toate celelalte euristici au fost construite peste aceasta functie.

- ***min_distance_to_targets*** – rezolva suma distantelor Manhattan dintre fiecare cutie si cea mai apropiata tinta
  In testul *easy_map1.yaml*, aceasta euristica a condus la solutii rapide, iar algoritmul LRTA* functiona normal.Insa, pentru testele *medium_map2.yaml* si *hard_map1.yaml,* am observant ca jucatorul tinde sa se roteasca in jurul aceleiasi cutii, chiar daca solutia era departe.Mi-am dat seama ca euristica ignora pozitia jucatorului si relatiile cutie-tinta, astfel ca am decis sat trec la o versiune mai coordonata, anume euristica de mai jos.

- ***min_matching_distance –*** am folosit o abordare in care fiecarei cutii ii este asociata o singura tinta apropiata
  Am testat aceasta euristica pe *hard_map2.yaml*, unde cutiile sunt in pozitii appropriate, dar cu tinte foarte dispersate. Rezultatul a fost vizibil mai bun:jucatorul nu mai oscila intre tinte, iar numarul de stari explorate a scazut cu 25%.

- ***box_player_distance -*** am observat ca jucatorul pierde mult timp ajungand la cutii.De exemplu, in *medium_map1.yaml*, uneori jucatorul mergea in colturi fara un scop clar. Am adaugat aceasta euristica pentru a ghida jucatorul spre cutiile care nu sunt deja pe tinta.Dupa integrarea acestei euristici, traseul jucatorului a devenit mai clar, solutiile fiind mai scurte, cu aproximativ 3 mutari in medie.

- ***deadlock_detection -*** detecteaza blocajul cutiilor in colturi.De exemplu, in testul *large_map2.yaml*, algoritmul se bloca frecvent in stari in care o cutie ajungea intr un colt si nu mai putea fi mutata.Am introdus-o pentru ca detecteaza cutiile blocate de 2 pereti si aplica o penalizare mare. Dupa adaugare, in testele *medium_map2.yaml* si *hard_map1.yaml*, numarul de stari explorate a scazut semnificativ si algoritmul a evitat traseele pierzatoare.

- ***pull_move_penalty –*** am definit o penalizare proportionala cu numarul de obstacole in jurul cutiilor care nu sunt pe tinte,pentru a descuraja pozitionarile gresite.De exemplu, pe harta *large_map1.yaml*, penalizarea a dus la reducerea *pull_moves* de la 6 la 1 in rularea SA cu euristica combinata.

- ***distance_to_goal_state -*** pentru a favoriza starile partial complete, am adaugat un scor proportional cu numarul de cutii deja plasate correct.Am testat aceasta modificare pe *super_hard_map1.yaml*, unde SA reusea adesea sa aduca 2 din 3 cutii pe tinta, dar nu stia ca e aproape de success.Dupa introducerea acestei euristici, probabilitatea de acceptare a mutarilor utile a crescut si algoritmul a gasit solutia complete in aprox. 12% mai putine iteratii.

- ***combined_heuristic -*** am definit o euristica ce aduna toate scorurile ponderate (din euristicile anterioare), iar aceasta a dat cele mai bune rezultate, mai ales in SA, reducand timpul de executie si crescand succesul.

5.Comparatia intre LRTA* si Simulated Annealing

1. Timp de executie

- In urma rularii celor 2 algoritmi, am obtinut timpi de executie semnificativ mai mici pentru LRTA* in comparatie cu SA, in special pe hartile de dimensiune mai mica si medie.

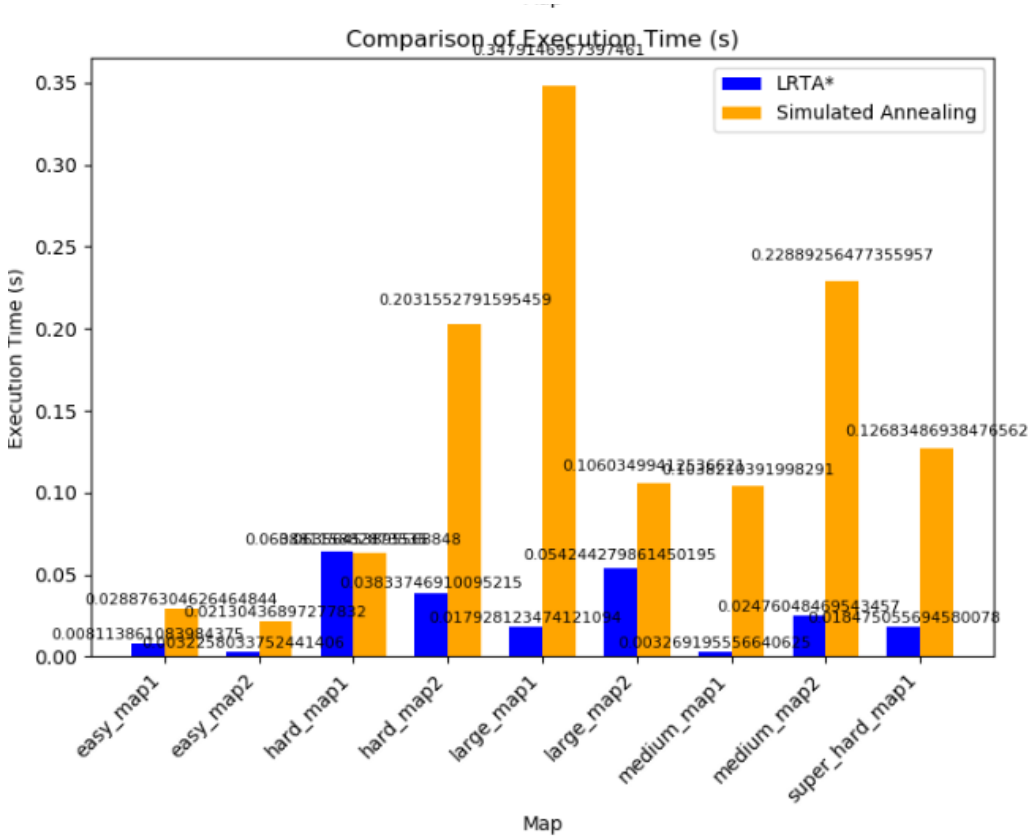
- Acest lucru se datoreaza in principal faptului ca LRTA* ia decizii local, explorand doar succesorii directi ai starii curente si actualizand valorile euristice in mod incremental.Astfel, fiecare pas este rapid si determinist, fara a evalua intreaga ramura a spatiului starilor. Pe de alta parte, SA se bazeaza pe un mecanism de explorare aleatorie ghidata de o functie de temperature si acceptare probabilistica. Acest lucru inseamna ca, uneori, algoritmul accepta mutari suboptime si ajunge in stari care nu duc imediat spre solutie, necesitand mai multe iteratii pentru a ajunge la un rezultat valid.Aceasta randomizare vine la pachet cu un cost de timp mai mare, intrucat nu toate cautarile duc imediat spre progres.

- Timpii de executie pentru LRTA*:
  -easy_map1: 0.007s
  -easy_map2: 0.003s
  -medium_map1: 0.0036s

-medium_map2: 0.0235s
-hard_map1: 0.0624s
-hard_map2: 0.0363s
-large_map1: 0.0168s
-large_map2: 0.0514s
-super_hard_map1: 0.0180s

- Timpii de executie pentru SA:
  -easy_map1: 0.0278s
  -easy_map2: 0.0201s
  -medium_map1: 0.1021s
  -medium_map2: 0.2321s
  -hard_map1: 0.0509s
  -hard_map2: 0.2171s
  -large_map1: 0.3251s
  -large_map2: 0.1243s
  -super_hard_map1: 0.1294s

- Grafic:



2. Numar de stari contruite

- In cazul LRTA*, numarul de stari construite este semnificativ mai mic comparativ cu algoritmul SA, datorita naturii sale locale. LRTA* exploreaza doar succesorii directi ai starii curente si actualizeaza estimarile euristice incremental, fara a pastra un istoric al tuturor starilor parcurse.

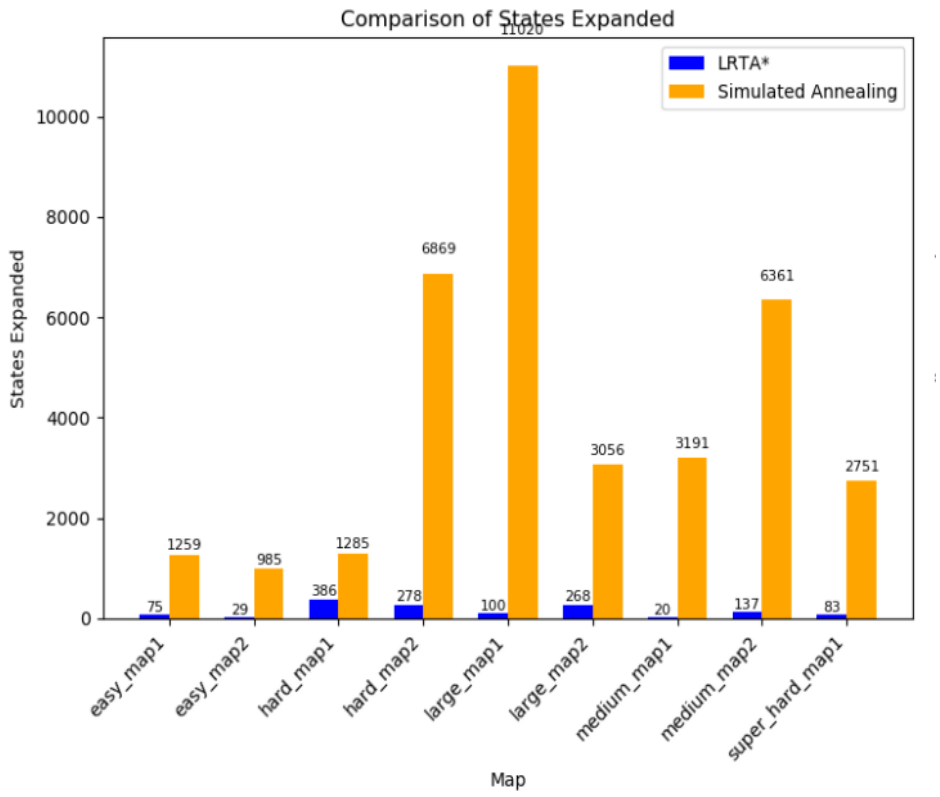
- Numarul de stari contruite LRTA*:

  -easy_map1: 75
  -easy_map2: 29
  -medium_map1: 20
  -medium_map2: 137
  -hard_map1: 386
  -hard_map2: 278
  -large_map1: 100
  -large_map2: 268
  -super_hard_map1: 83

- Numarul de stari contruite SA:

  -easy_map1: 1259
  -easy_map2: 985
  -medium_map1: 3191
  -medium_map2: 6361
  -hard_map1: 1285
  -hard_map2: 6869
  -large_map1: 11020
  -large_map2: 3056
  -super_hard_map1: 2751

- Grafic:



3. Calitatea solutiei

- LRTA* produce solutii de calitatea mai buna in ceea ce priveste numarul de mutari de tip pull, comparative cu Simulated Annealing.Datorita naturii sale deterministe, LRTA* evita adesea traseele care implica blocaje sau mutari ineficiente. Pe de alta parte, SA exploreaza frecvent cai mai putin optime in prima faza, ceea ce duce deseori la un numar mai mare de mutari de tip pull.

- Numar mutar de tip pull – LRTA*:

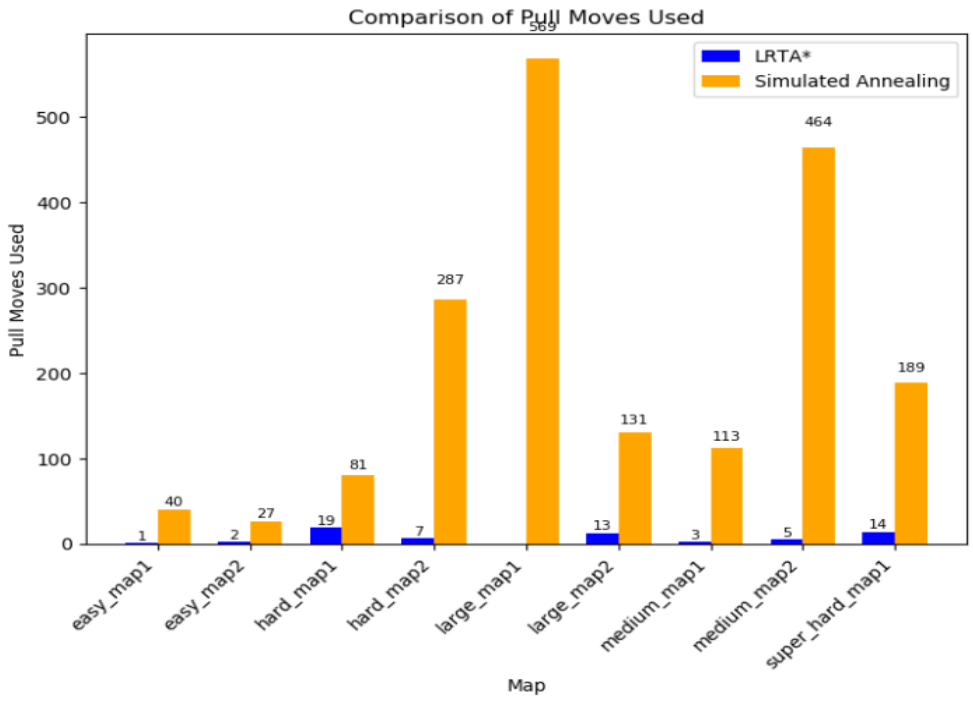
  -easy_map1: 1
  -easy_map2: 2
  -medium_map1: 3
  -medium_map2: 5
  -hard_map1: 19
  -hard_map2: 7

-large_map1: 0
-large_map2: 13
-super_hard_map1: 14

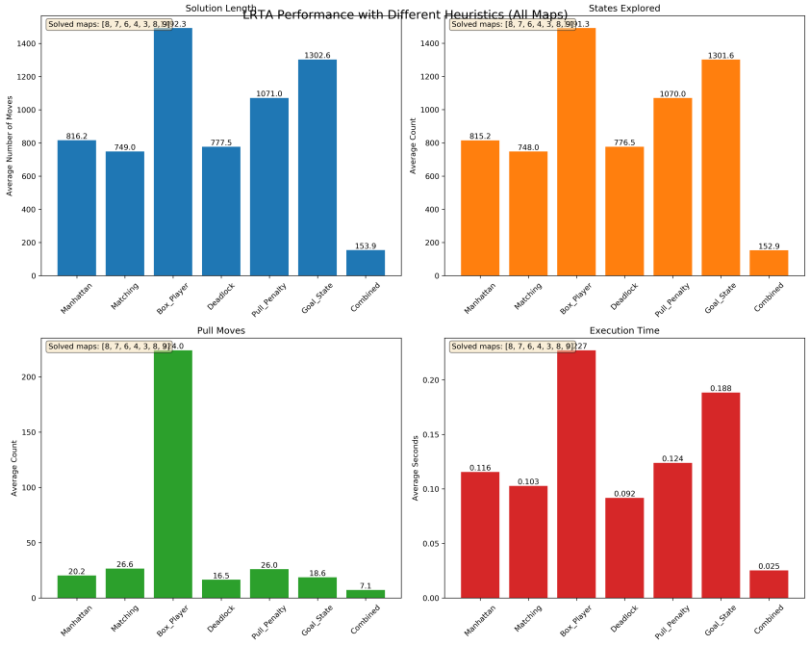
- Numar mutari de tip pull – Simulated Annealing:

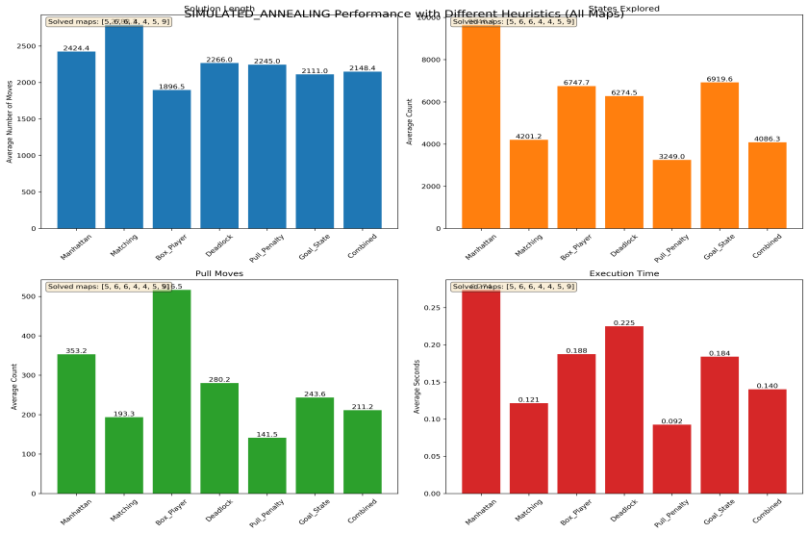
-easy_map1: 40
-easy_map2: 27
-medium_map1: 113
-medium_map2: 464
-hard_map1: 81
-hard_map2: 287
-large_map1: 569
-large_map2: 131
-super_hard_map1: 189

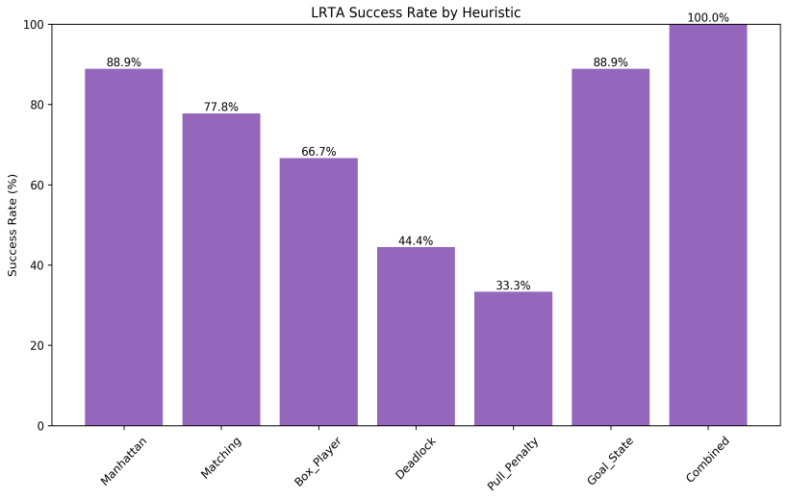
- Grafic:



6.Alte observatii

- Pe baza graficelor de mai jos, putem vedea, din nou, ca timpul de executie pentru SA este mai mare LRTA*, in special in combinatie cu euristici suboptime.De exemplu, *Deadlock* si *Box_player* aduc timpi >0.2s pentru SA.
- SA exploreaza semnificativ mai multe stari decat LRTA*,indifferent de euristica.De exemplu, pentru euristica Manhattan, media este de aprox. 9000 stari pentru SA, vs aprox. 800 stari pentru LRTA*.
- Euristica *Combined* a obtinut o rata de success de 100%, urmata de *Manhattan* si *distance_to_goal_state*(88.9%).
- 

- 



7.Rulare

- Tema poate fi rulata folosind urmatoarea comanda:
  $python3 main.py lrta*/simulated-annealing all