Inteligenta Artificiala: Tema 1 – Sokoban

-Seria CC-

Nume: Calu Andrei-Daniel

Grupa: 332CC

-> Jurnalizarea ideilor pentru euristicile folosite în cei 2 algoritmi:

Note: In general, analiza asupra euristicilor am realizat-o prin algoritmul Simulated Annealing, incat a fost primul implementat (cu ajutorul variantei de la laborator).

>Misplaced: Am inceput rezolvarea temei folosind euristica Misplaced, anume evaluarea fiecarei stari era egala cu numarul de cutii ce nu se aflau pe pozitii finale. Dupa modificarea parametrilor pentru Sim Annealing, in special rata de cooling la 0.9999 pentru a oferi mai mult timp de explorare algoritmului, am reusit sa obtin o rata de succes de 100% in rezolvarea hartilor, insa, numarul de stari explorate era foarte mare.

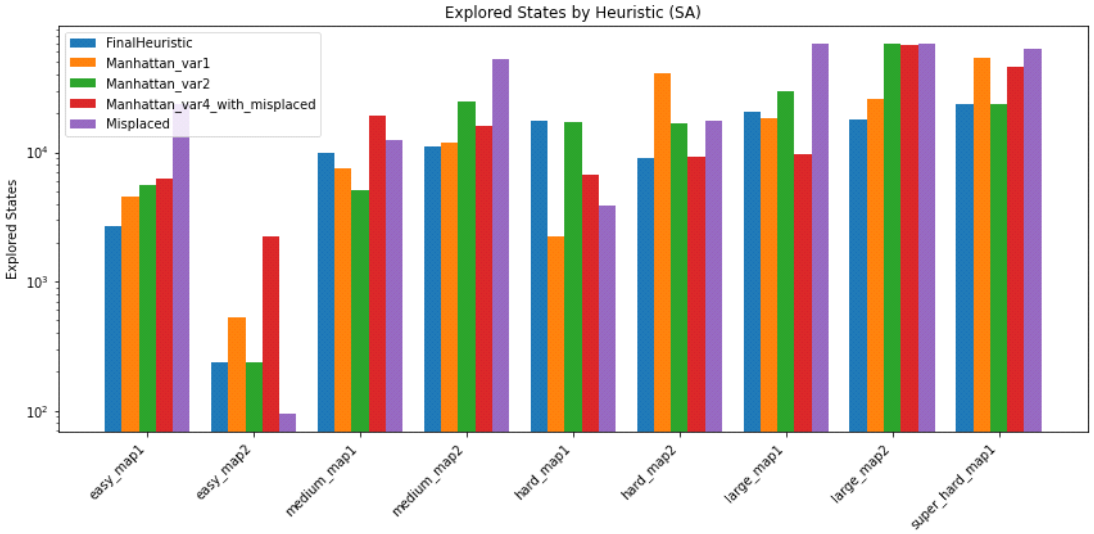
>Manhattan_var1: Pentru imbunatatirea performantei am decis sa utilizez alta euristica, anume suma distantelor Manhatten de la fiecare cutie la cel mai apropiat punct final, astfel incat sa pot evalua si starile intermediare, in care se incearca deplasarea cutiilor catre o solutie. Alegerea distantei Manhatted in loc de cea Euclidiana a fost motivata de faptul ca toate entitatile jocului se pot deplasa doar sus/jos/stanga/dreapta, nu si pe diagonala, asa ca am decis ca nu e necesara si implementarea unei euristici cu distanta Euclidiana. In mare parte a hartilor, analiza euristicii a decurs asa cum ma asteptam: numarul de starie explorate prin Simulated Annealing era mai mic acum decat prin utilizarea euristicii Misplaced. Totusi, nu a fost cazul peste tot, lucru la care nu ma asteptam, dar cred ca s-a intamplat acest lucru deoarece nu tineam cont de pozitiile barierelor spre pozitiile finale.

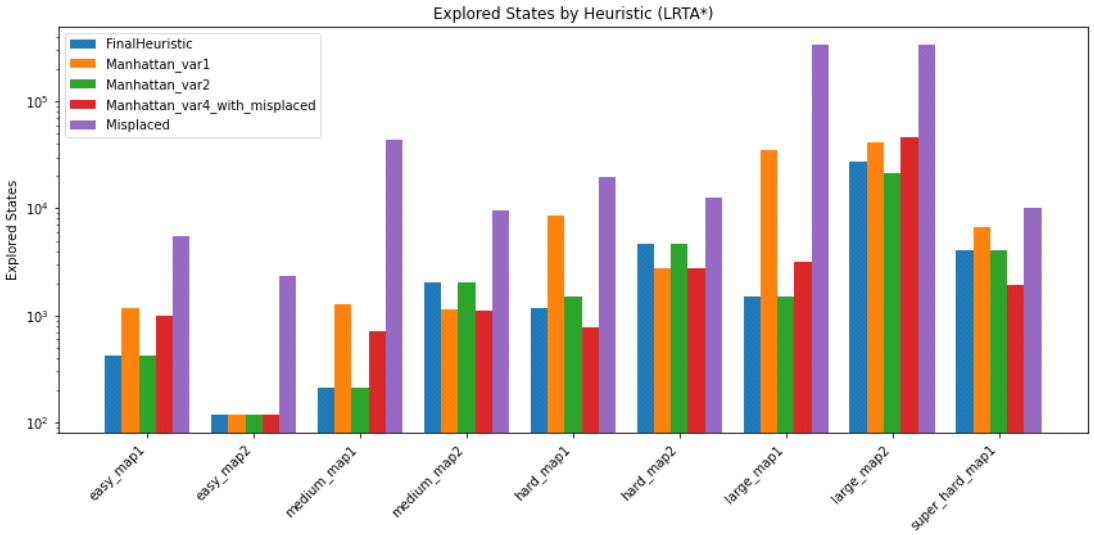
>Manhattan_var2: Incat in varianta anterioara puteam sa apreciez exclusiv stari in care cutiile erau mutate, am decis sa adaug la costul total si distanta Manhattan de la player la cea mai apropiata cutie, pentru a sti in ce directie sa se indrepte spre una (scopul fiind de a le muta in pozitii finale). Am analizat rezultatele, iar efectul a fost negativ: existau harti pentru care nu se gasea solutie si harti cu numar de pasi mai mare decat anterior. Ca urmare, am decis sa adaug aceasta valoare ponderata (distanta player->cutie / 10), incat sa prioritizez mutarea cutiilor, iar, doar in caz de tie-braker (nicio cutie nu e mutata in starea urmatoare) sa iau starea in care player-ul se apropie de o cutie. Rezultatul a fost unul mai bun: Toate hartile au primit o solutie, iar, in unele harti, numarul de stari explorate a fost mai mic decat prin varianta anterioara de euristica.

>Manhattan final: In incercarea de a minimiza numarul miscarilor de pull, am adaugat in euristica o verificare pentru deadlock: daca o cutie este blocata '(up or down) and (left or right)', atunci am ajuns la un colt/deadlock si jocul trebuie resetat. Astfel, atribui starii o valoare estimativa infinita, pentru a nu fi aleasa niciodata. Initial, am decis sa verific colturile

formate atat din peretii hartii, cat si din alte cutii sau bariere (blocuri prin care nu se poate trece). Totusi, comportamentul algoritmului SA a fost neasteptat, returnand harti pentru care nu a gasit solutii, sau harti pentru care a gasit solutii in mai multi pasi. Ca urmare, am ales o varianta mai permisiva, verificand doar deadlock-uri formate din colturile hartii. Rezultatul a fost unul mai bun. Dupa parerea mea, motivul eseculi initial apare din cauza miscarilor de pull prea reduse in acea abordare, acestea putand oferi o solutie corecta mai rapid.

>Manhattan_var4: Dupa ce am vazut ca pot aparea multe comportamente neasteptate in incercarea de a face o euristica mai buna, am decis sa incerc si sa o fac mai slaba cu gandul ca poate oferi un rezultat mai bun. In Manhattan final, am inlocuit suma distantelor de la fiecare cutie la cel mai apropiat final cu Misplaced (numarul de cutii in pozitii nefinale). Totusi, utilizarea acestei euristici a avut un rezultat destul de slab, dupa cum ma asteptam, neavand nici macar rata de 100% succes. Astfel, am decis sa pastrez varianta anterioara de euristica.





-> Optimizări realizate pentru cei doi algoritmi:

>Simulated Annealing: (Suport: laborator)

- In loc de utilizarea functiei Softmax, am decis ca la fiecare pas sa aleg o mutare random din lista de mutari posibile, dupa care sa o aplic starii curente si sa verific daca va fii acceptata (e mai buna decat cea precedenta sau nimereste probabilitatea). Astfel, algoritmul poate scapa mai usor din minime locale slabe, si, in special, evauleaza un singur vecin la fiecare pas, numarul de stari construite fiind egal cu numarul de iteratii.

- In plus, dupa un anumit numar de iteratii (4800, ales prin incercari repetate), daca algoritmul nu a gasit o noua stare minima fata de ultima, se intoarce la aceasta, incat se presupune ca merge pe un drum gresit. (Restart cu threshold)

- Nu reprezinta intocmai o optimizare, insa am adaugat un seed 42 pentru a genera aceleasi rezultate la fiecare rulare a algoritmului (avand in vedere alegerile random).

>LRTA*: (Suport: curs 3)

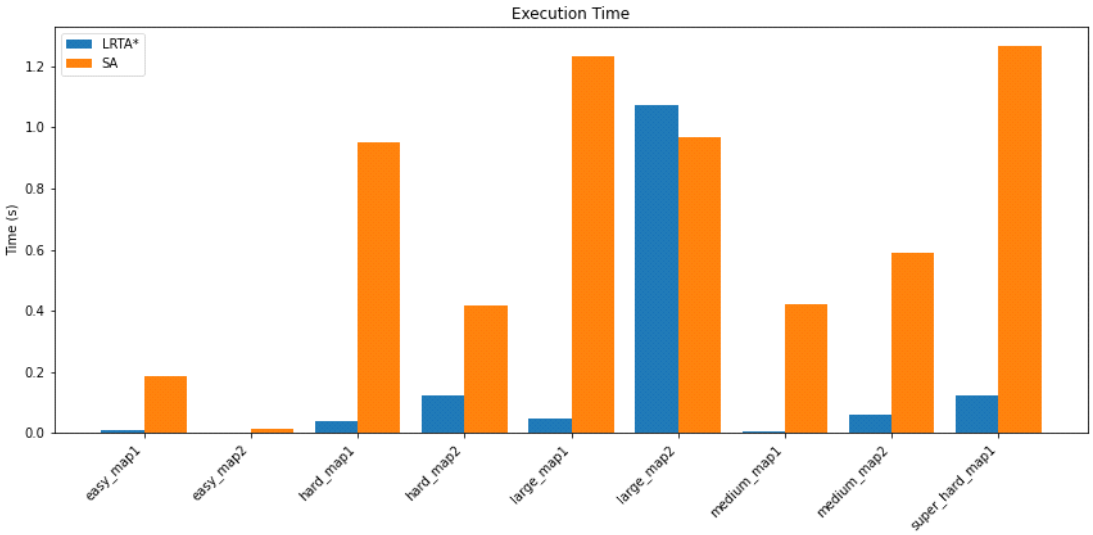
- Nu am realizat nicio optimizare in cadrul acestui algoritm. L-am implementat cu ajutorul pseudocodului oferit drept suport, in curs.

-> Comparatia între cei doi algoritmi, pe baza euristicii principale:

>Timp de executie:

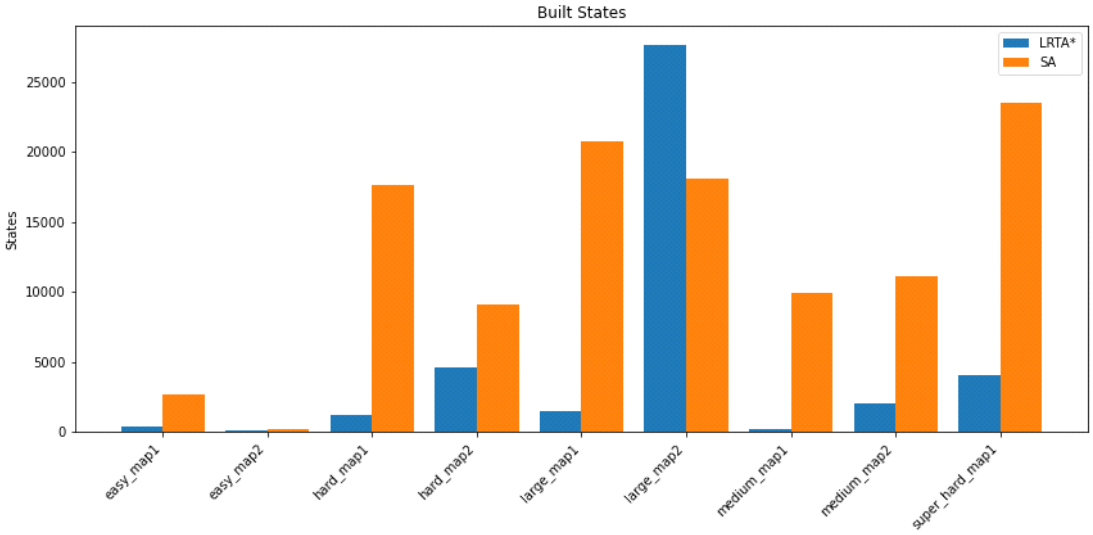
Algoritmul LRTA* prezinta un timp mult mai mic de executie fata de Simulated Annealing pentru aproape toate hartile, incat reprezinta un algoritm determinst ce alege intotdeauna cea mai buna stare si nu pierde timp in explorarea altor stari pe ideea ca pot duce catre un drum mai bun.

Timpul mai mare al alogritmului LRTA* pentru harta large_map2 este, probabil, deoarece costurile estimate pot fi inselatoare si ajunge sa mearga pe un drum gresit ce nu are o solutie finala. Astfel, trebuie sa se intoarca si sa refaca estimarile, actiune costisitoare din punct de vedere al timpului.



>Numar de stari construite:

Cei doi algoritmi priviti din punct de vedere al starilor construite prezinta un comportament asemanator precum in cazul timpilor de executie, motivele fiind cele prezentate mai sus. Am incrementat, atat in cadrul algoritmului LRTA* cat si in cadrul algoritmului Simulated Annealing, numarul de stari construite, la fiecare apel al functiei 'apply(actiune)' asupra unei stari (trecerea intr-o stare noua).



>Calitate a solutiei (numar de miscari de pull):

Algoritmul Simulated Annealing are un numar mult mai mare de miscari de tip pull fata de LRTA*, incat exploreaza multe solutii partial gresite, dupa care incearca sa le corecteze, facand astfel pull-uri fara rost pentru a reveni si a lua pe un alt drum.

Se poate observa o crestere imensa pe harta super_hard_map1. Dupa parerea mea, se datoreaza dificultatii hartii, avand un spatiu de soltuii foarte mare pe care simulated annealing incearca sa il exploreze, de cele mai multe ori intorcand-use prin miscari de tip pull.