Inteligentă Artificială: Tema 1 – Sokoban

Mărgheanu Cristina-Andreea 333 CA

Timeline rezolvare:

Am început rezolvarea temei folosind ca euristica distanta Manhattan deoarece este simplu de implementat si inteles, dar și potrivită pentru table ortogonale precum cele din Sokoban. Distanța Manhattan estimează costul minim de deplasare al unei cutii până la o țintă, fără a lua în calcul obstacolele sau constrângerile impuse de celelalte cutii sau de poziția jucătorului.

Pentru testul medium_map2.yaml, am observat comportamente diferite în funcție de algoritm deoarece avem cutii si tinte care nu se afla în niște directii clare de mutare, iar cutiile se pot bloca unele pe altele sau pot ajunge in stări în care nu mai pot sa fie mutate eficient.

Beam Search, care funcționează offline și evaluează global generarea stărilor, a reușit să găsească soluția în 30 de pasi cu beam_width = 60 având ca euristică distanța Manhattan. La testul large_map2 însă parcurgerea nu se realizeaza cu aceasta latime în timpul cerut de 5 minute. Aceasta parcurgere a hartii functioneaza doar in cazul in care aleg un beam_width considerabil mai mare, de exemplu 500. Atunci jocul se sfarseste in 2.63 secunde si pentru aceasta harta. Modificarea lui k insa afecteaza destul de mult strategia, conform informatiilor din curs. Afecteaza memoria consumata, dar pe testele complexe , o valoare mare pentru beam_width (ex. k=500) a fost necesară. O valoare mică duce la eliminarea prea devreme a stărilor promițătoare .Cautam deci o imbunatatire viitoare. Am luat în considerare modificarea euristicii cu distanța Manhattan pentru a penaliza cutiile deja plasate pe target sau pentru a include distanța dintre jucător și cea mai apropiată cutie. Totuși, pentru acest test(medium_map2.yaml), am păstrat varianta Manhattan de bază, care s-a dovedit suficientă pentru Beam Search, dar mai puțin eficienta pentru LRTA*. Dupa modificarea euristicii Manhattan simpla am implementat una imbunatatita in urma analizei testului large_map2 care imi depasea timpul. Astfel noua eursitica reduce costul dacă o cutie e deja pe un target, minimizează mișcările în plus adăugând distanța playerului la cutii si prioritizează mutarea cutiilor utile, nu pe toate. Deci pentru rularea testului large_map2 de beam search am acum un timp de 0.413 secunde.

In ceea ce privește traseul parcurs de LRTA*, acesta se realizeaza într-un număr foarte mare de pasi si am observat că algoritmul a oscilat temporar în jurul cutiei, ceea ce a dus la o creștere a numărului de pași. Acest comportament se explică prin faptul că LRTA* învață H treptat și local, fără să aibă o viziune globală asupra planului.

Diferente inițiale între algoritmi, cu euristica cu distanta Manhattan:

- *Tip de cautare*:
  Beam Search: cautare pe niveluri, extinde toata generatia curenta apoi alege cele mai bune beam_width succesoare

  LRTA: se muta imediat, rescrie din mers

- *Memorie utilizată*:

BS: pastreaza la fiecare nivel maximum beam_width stari, adica k candidiati. Are cost ridicat in functie de cate stari decidem sa memoreze.

LRTA: tine minte doar starea curenta si un dictionar H pt euristica

- *Strategie*:

La Beam Search pornim de la starea initiala si generam toate starile vecine, nivelul urmator. Le sortam dupa g + h, unde g(n) = cat am mers pana aici si h(n) = cat mai avem de mers. Pastram doar beam_width cele mai bune.

LRTA: genereaza toate mutarile posibile, le aplica, obtine succesorii, alege succestorul cu 1 + h minim

Mutare la o alta euristica: Pentru o estimare mai realistă, am experimentat și cu o euristică BFS, care în loc să estimeze distanțele dintre cutii și ținte prin Manhattan, calculează distanțele minime reale printr-un algoritm BFS de la fiecare cutie la cel mai apropiat target.

Această abordare tine cont de obstacole reale (nu doar de poziție), evitând situații în care o cutie pare aproape de un target, dar de fapt este blocată.Permite o explorare mai eficientă în hărți complexe, reducând numărul de stări inutile generate.

Pentru euristica Misplace**:** Numărul de obiecte care nu sunt la locul final.În Sokoban**:** numărul de cutii care nu sunt pe targeturi. Nu aduce neaparat niste imbunatatiri semnificative, dar m-am gandit sa o incerc deoarece in Sokoban este inutil să mișcăm cutii deja plasate corect; această euristică penalizează doar cutiile care mai trebuie aranjate, deci ghidează mai eficient spre final.

Optimizari specifice problemei conform Sokoban :am facut 2 euristici speciale aducand aceste imbunatatiri:

- Atat in euristica imbunattita pentru Manhattan cat si pentru cea de-a 2-a(Manhattan_imporved si improved_sokoban_heuristic din fisierul heuristics.py), cutiile deja pe target nu mai sunt luate in calcul, ele nu mai trebuie mutate deci avem cost 0. Sectiune de cod: movable_boxes = [b for b in boxes if b not in goals]

- In plus, in sectiunea de cod : dist_player_to_box = min(abs(player_pos[0]-b[0])+...) h += dist_player_to_box , am adaugat distanta reala de la player la cea mai apropiata cutie utila, evitand astfel situatiile in are eurstica subapreciaza costul cand playerul e izolat de cutii.

- Fallback controlat cu un cost finit 50 daca BFS nu gaseste drumul. Permite compararea starilor fara a distruge ordinea si a face un nod invizibil.

- In beam_search_solver: am implementat un restart adaptiv pentru cand h nu mai scade, pentru a impiedica blocajele. Totodata am folosit o lista pentru a pastra cele mai bune k pozitii, nu un heap si am facut taiere direct prin slicing.

- improved_sokoban_heuristic: calculeaza h care care estimează cât de departe este starea curentă de o stare soluție. Aspectele urmarite sunt: cat de aproape sunt cutiile de target-uri, folosind BFS pentru fiecare cutie. Apoi cat de aproape e jucatorul de o cutie mutabila: mutarea incepe doar daca jucatorul ajunge langa o cutie.

-LRTA : Am folosit mereu stringul starii nu obiectul Map, deoarece copierea Map e mai grea, in timp ce stringul e mai stabil.

Penalizează mișcările de tip pull folosind undo_moves.

Comparatii intre cei 2 algoritmi:

Pentru a face acest lucru am ales initial ca test de referinta super_hard_map1 si o sa folosesc aceeasi euristica pentru amandoi algoritmii, bfs_distance.

- timp de executie: LRTA 0.290 cu 13 miscari de pull, iar la BeamSearch 1.103 sec si 13 miscari de pull.

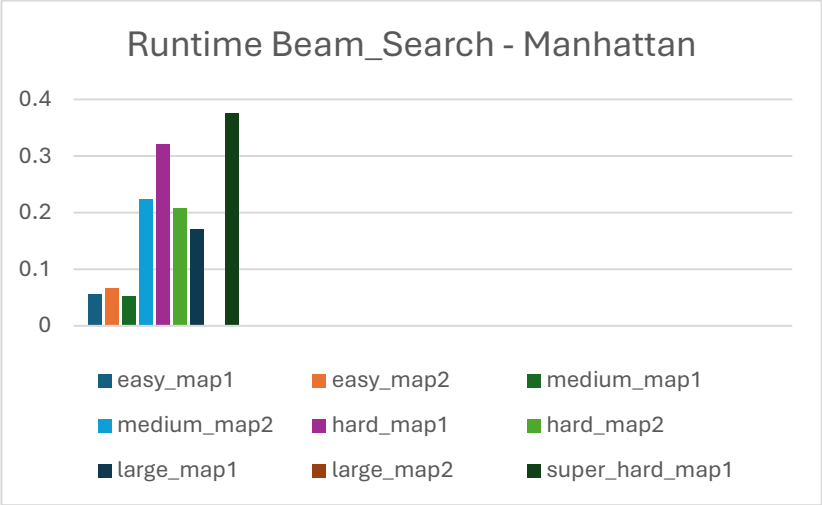
Am realizat 2 grafice pentru euristica Manhattan atat pentru Beam Search cat si pentru LRTA_star pentru toate testele.

Beam Search are timpi de execuție constanți pe majoritatea hărților:

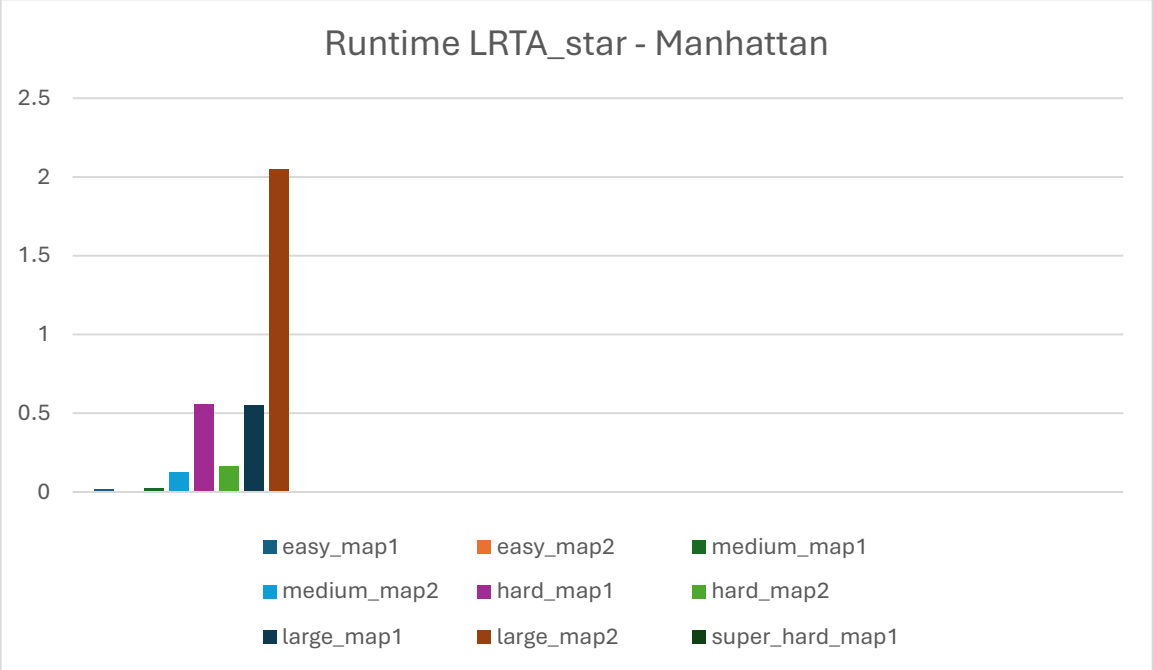
- Pentru hărțile easy_map1/2, timpul este sub 0.07 secunde, ceea ce arată eficiența algoritmului când spațiul de căutare este redus.
- super_hard_map1 are cel mai mare timp: 0.38 secunde, dar tot rămâne sub o secundă.
- În schimb, large_map2 nu este prezent pentru că execuția Beam Search a depășit limita de 5 minute cu k initial ales 60, ceea ce sugerează că euristica Manhattan nu este potrivită pentru instanțe mari și complexe. Cu euristica Manhattan imbunataita avem un timp de 0.357s. Un rezultat rezonabil se obtine si pentru un k(beam_width) mai mare.

LRTA*:

- Pe hărțile easy și medium, timpii sunt comparabili sau mai mici decât Beam Search.
- Pe hărți mai complexe precum hard_map1, large_map1/2, timpii cresc semnificativ:
    - large_map2 are un timp de 2.05 secunde, sub Beam Search (care nici nu a terminat execuția cu euristica neimbunataita pentru k = 60).
    - Acest lucru indică faptul că LRTA* e eficient pe spații mai mici, nu e extrem de bun la creșterii dimensiunii și a complexității hartilor, având nevoie de mai multe reexplorări și updateuri ale funcției H.

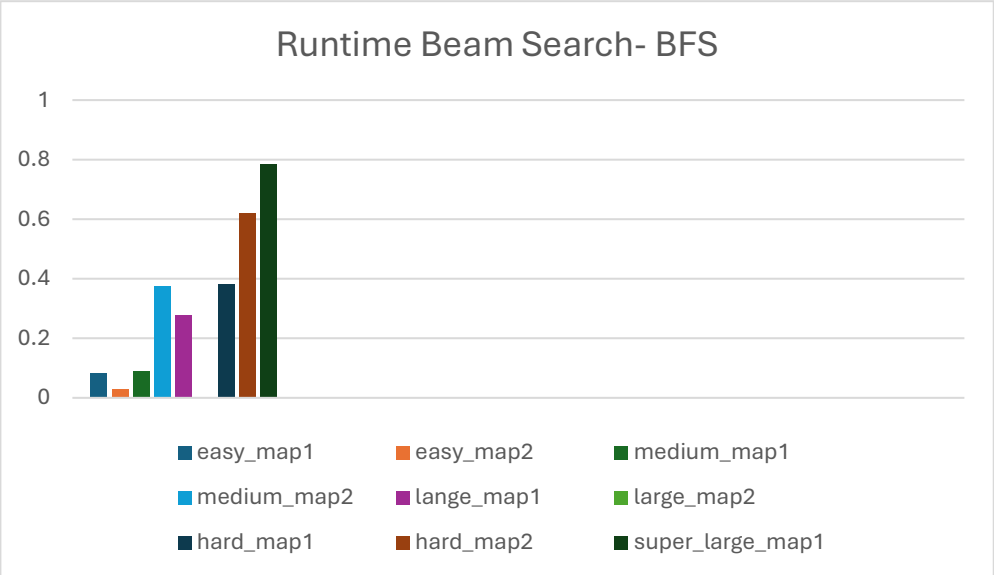


Timeout la large_map2 pentru k = 60, functional pentru k mult mai mare, sau pentru o euristica putin imbunatatita.

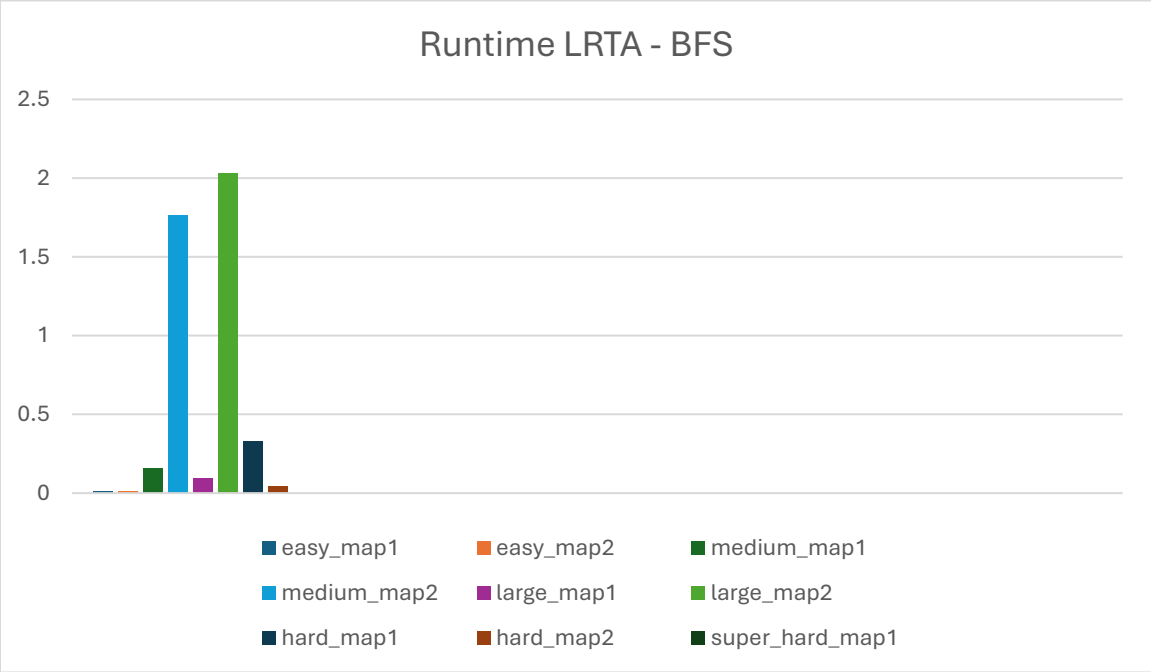


Posibile imbunatatiri: La Beam search: Verifica daca am pus o cutie pe un target si din starea curenta am scazut din costul acesteia ca sa nu mai mute cutia. Trebuie sa minimizez numarul de miscari de pull, deci la distanta de la cutie la targetul cutiei, am adunat suma dupa distanta de la fiecare cutie la targetul ei plus distanta de la player la cea mai apropiata cutie pt ca initial playerul era departe de cutie.

Realizez grafice si pentru euristica BFS pentru ambii algoritmii si timpii de rulare pe toate testele:

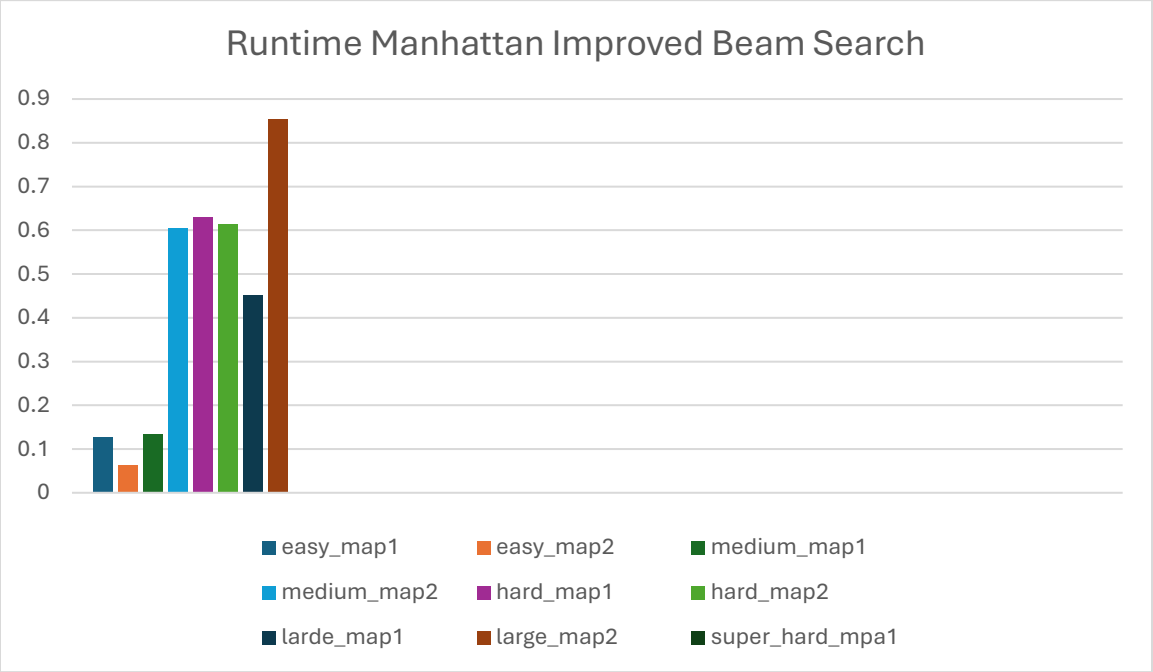


Timeout la large_map2 pentru k = 60, functional pentru k mult mai mare, de exemplu k = 500.



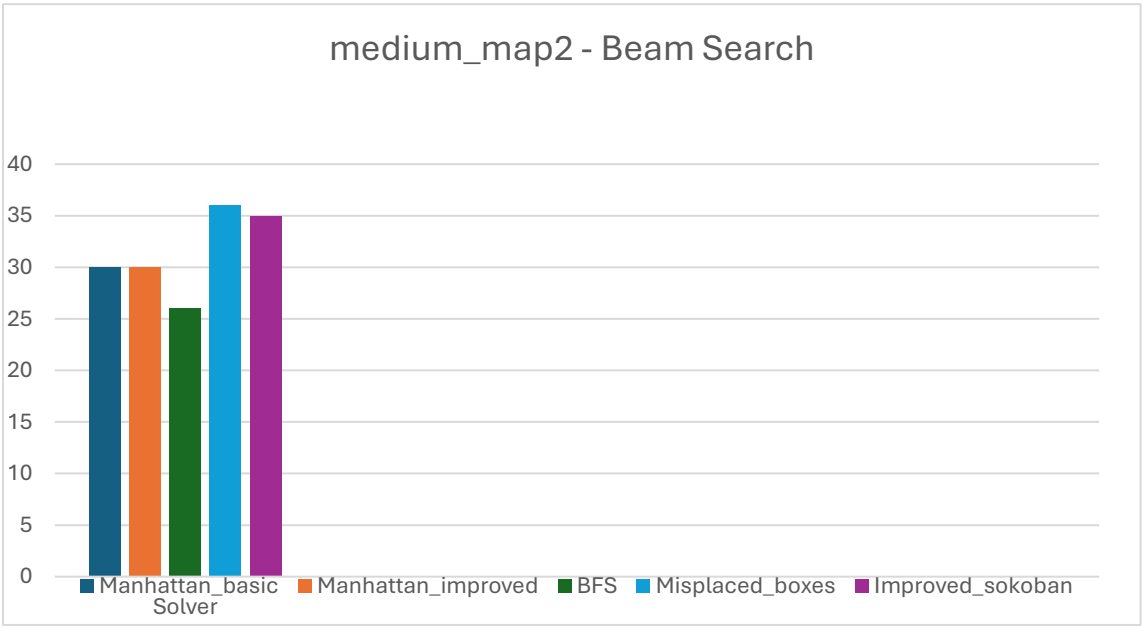
Pe aceeași euristică BFS, tabelul arată că Beam Search păstrează timpi sub o secundă pentru toate mapele mai putin pentru large_map2 pe care nu o poate parcurge in timpul limita cerut**,** crescând de la hărțile *easy* până la *super_large*, în timp ce LRTA*, deși rămâne foarte rapid pe majoritatea instanțelor, e foarte mare punctual pe hărțile cu mulți pereți și cutii (*medium_map2* și *large_map2*).

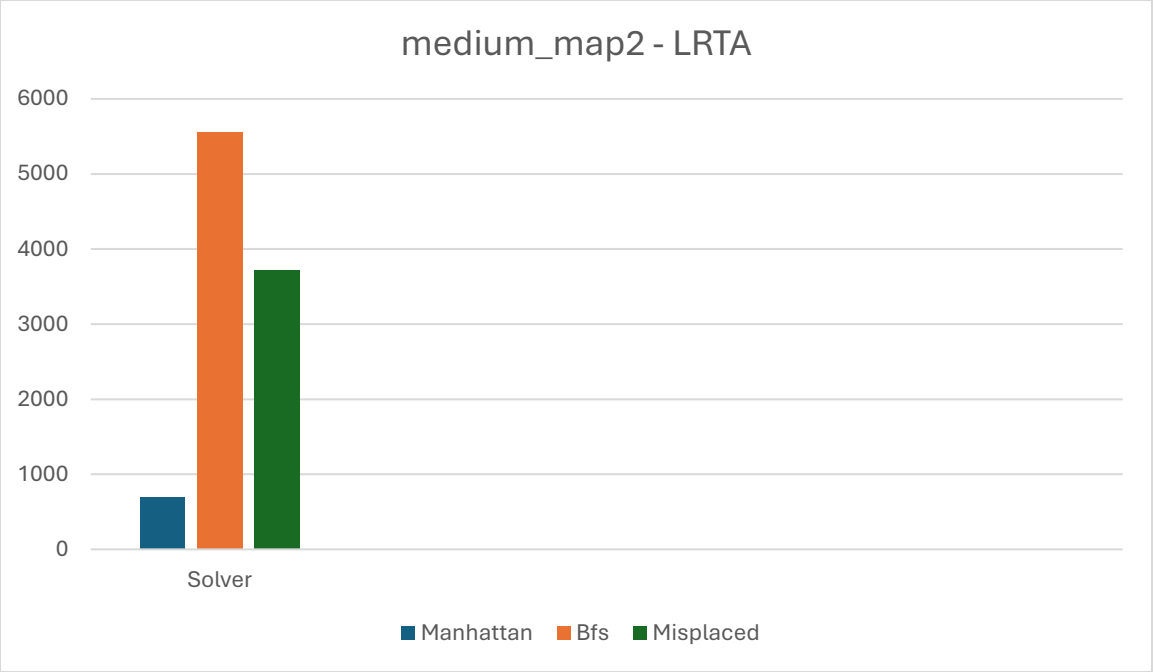
Grafice pentru euristica Manhattan Imbunatatita pentru a observa diferentele intre Manhattan simplu, fara adaugiri Sokoban:



Diferente intre acesta si Manhattan simplu: Hartile mici au in ambele cazuri timpi foarte rapizi, hartile largi si complexe au timp putin mai are pentru euristica imbunatatita deoarece avem adaugiri suplimentare de luat in considerare dar cel mai important este ca toate hartile functioneaza acum intr un timp scurt, fara modificarea lui k cu o valoarea foarte mare.

Continuam studiul comparatiilor si analizarea graficelor cu diferente pe aceeasi harta, medium_map2, pentru ambii algoritmi, cu toate euristicile posibile.



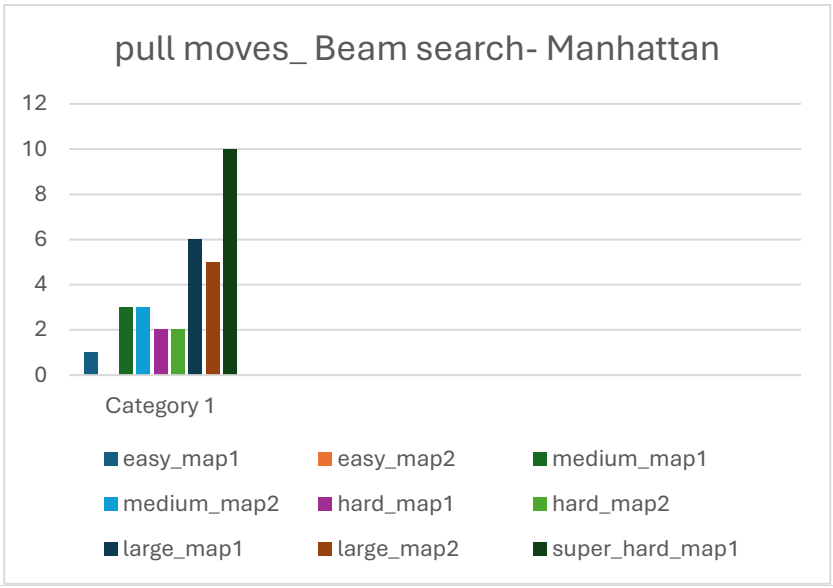


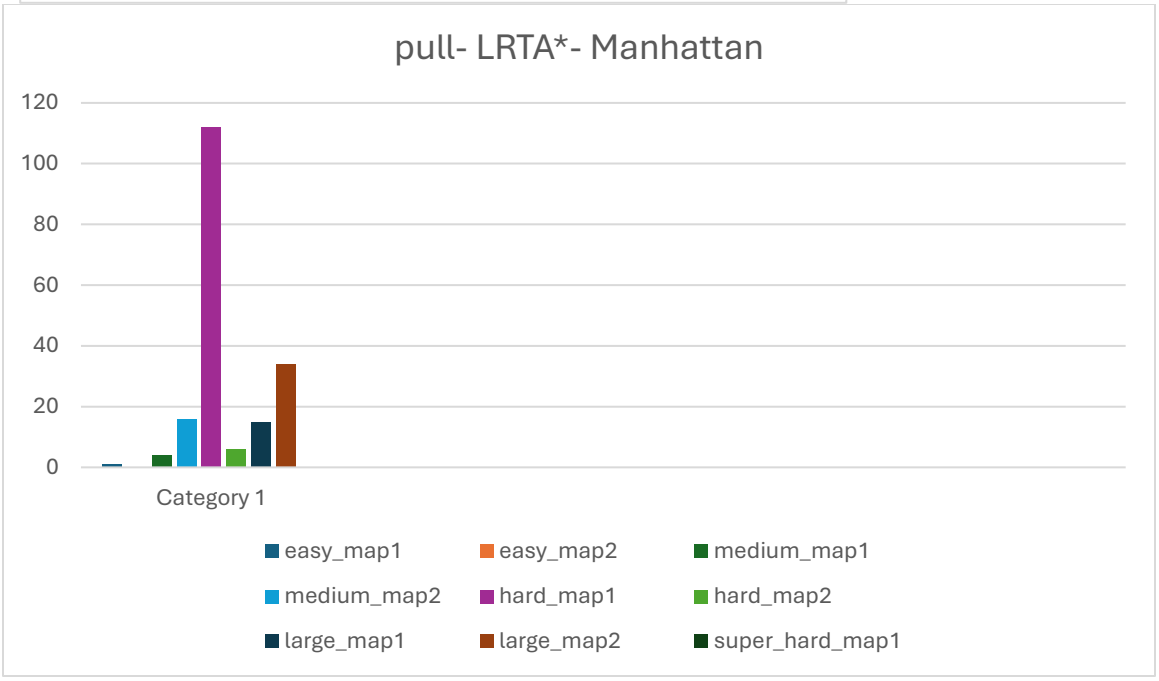
Concluzie: Toate cele 4 euristici oferă un comportament consistent și eficient pentru Beam Search, dar BFS duce la cele mai puține stări explorate.

Pentru că **LRTA*** face o învățare incrementală și actualizează costurile în timp ce explorează spațiul. Dacă euristica este slabă ,LRTA* se plimbă mai mult și poate reveni des în stări anterioare , de aici și numărul foarte mare de stări.

In plus, ceea ce am analizat la LRTA: pasi, adica o mutare facuta de player este putin diferita de un nivel de la beam search. LRTA* nu cauta solutia dinainte, fiecare pas este aplicat imediat, fara sa generezi multe variante, ci doar mergi pe cea mai buna. La Beam Search, nivelul este reprezentat de toate starile posibile dupa un numar fix de mutari, toti succesorii de adancime constanta. Pe fiecare nivel generez toate mutarile posibile pentru toti candidatii curenti. Evaluez si pastrez doar beam_width cele mai bune.

Acum comparam numarul de miscari de Pull, euristica Manhattan(imbunatatita la Beam Search pentru testul large_map2) si cea normala pentru LRTA*:





Comparatii si concluzii: Beam Search: numar mic de miscari de pull, hartile au intre 1 si 6, exceptia fiind super_hard_map1. Se menține eficient chiar și pe hărți mari precum large_map1, large_map2.

LRTA: Valori mult mai mari, mai ales pentru hartile complexe. În schimb, LRTA*, fiind un algoritm de învățare online, face multe corecții și reveniri, ceea ce duce la un număr semnificativ mai mare de *pull-uri*, mai ales în cazul hărților dificile.