Tema 1 – Cautarea in spatiul starilor

Dinca Daniel, 343C3

**1. Avantaje si dezavantaje pentru fiecare strategie in parte**

A. Breadth First Search

Avantaje BFS:

- dacă o soluție exista este garantat  ca BFS o va găsi
- dacă 2 soluții exista BFS o va găsi pe cea mai apropiata. Totuși, pentru problema noastră acest lucru nu este neaaparat un avantaj deoarece, deseori, cea mai apropiata soluție nu este soluția cu cel mai bun profit (deci cea mai calitativa)
- nu se blochează pe cai care nu duc la o soluție finala

Dezavantaje BFS:

- consuma multă memorie, deoarece toate nodurile de la un nivel sunt stocate înainte de a explora următorul nivel
- dacă soluția este departe de nodul root, atunci consuma foarte mult timp (mai ales în varianta neoptimizata)
- nu găsește soluția optima

B. Uniform Cost Search

Cost – reprezintă benzina consumata de la o stare la alta

Avantaje UCS:

- în general, atunci când costurile muchilor nu sunt egale, UCS va găsi cea mai scurta cale către un anumit nod. În cazul nostru, deci toate muchiile care reprezintă o mișcare (N, S, V, E) au costul de 1, muchiile care reprezintă o operație cu clientii (P, D) au asociat costul 0 (pentru ca nu se concuma benzina). Astfel, algoritmul UCS, aplicat pe problema noastră va favoriza mișcările de pickup sau dropoff, ceea ce poate duce la un profit mai bun.

Dezavantaje UCS:

- necesita o structura suplimentara care sa retina nodurile în funcție de cost
- dacă nu am fi avut mișcările de pickup și dropoff s-ar fi comportat exact ca BFS
- nu găsește soluția optima

C. Depth First Search

Avantaje DFS:

- nu are nevoie de așa de multă memorie
- găsește cea mai lunga soluție în cel mai puțin timp
- se comporta bine dacă are o soluție finala pe toate path-urile

Dezavantaje DFS:

- complexitatea depinde de numărul de path-uri
- se poate bloca cautand pe path-uri care nu contin o soluție
- nu se poate garanta ca se găsește o soluție
- nu se găsește soluția optima

Pentru problema nostra, DFS se comporta destul de bine, în sensul ca ajunge la o soluție finala, deoarece pe orice cale am merge, se ajunge la o cale finala deoarece se termina benzina.

D. Depth Limited Search

Avantaje DLS:

- eficient cu memoria
- dacă depth-ul dat este indeajuns de mare, atunci se comporta la fel ca DFS

Dezavantaje DLS:

- dacă depth - ul este mai mic decât depth - ul pentru cea mai apropiata soluție, atunci algoritmul nu va găsi soluția și se va consuma foarte mult timp deoarece trebuie sa exploreze toate path – urile cu acel nivel de adancime
- nu va găsi soluția optima

E. Iterative Deepening Search

Avantaje ID:

- în fiecare iteratie poți folosi un algoritm care este eficient cu memoria, de exemplu DLS

Dezavantaje ID:

- fiecare incrementare pentru depth, repeta munca iteratiei precedente
- dacă pentru un anumit depth, nu se găsește soluția, atunci va trebui sa exploreze toate starile pentru acel depth

F. Greedy Best First Search

Avantaje GBFS:

- este un algoritm de căutare informata
- găsește o soluție care are un profit mult mai bun decât solutiile găsite de algoritmii de căutare neinformata

Dezavantaje GBFS:

- deoarece este greedy, algoritmul duce la soluții mai puțin optime, în favoarea timpului de rulare mai rapid
- deoarece folosește doar functia euristica pentru a evalua un nod, nu ia în considerare informațiile acumulate precedent

G.  A* Search

Avantaje A*:

- este complet și optimal
- este optimal eficient (nu exita alt algoritm optimal care sa expandeze mai puține noduri decât a star)
- pentru a evalua un nod folosește atât functia euristica cât și informațiile adaugate precedent

Dezavantaje A*:

- viteza de execuție a algoritmului A* depinde de calitatea euristicii alese

H. Hill Climbing Search

Pentru problema noastră, algoritmul Hill Climbing search nu este unul bun, deoarece folosește doar nodul curent și evaluarea nodurilor următoare. Deoarece atunci când trecem de la un nod la următorul (prin consumarea de benzina) în principiu evaluarea starii scade, de cele mai multe ori algoritmul HCS oprindu-se deoarece alege sa pastreze benzina curenta.

I. Recursive Best First Search

Avantaje RBFS:

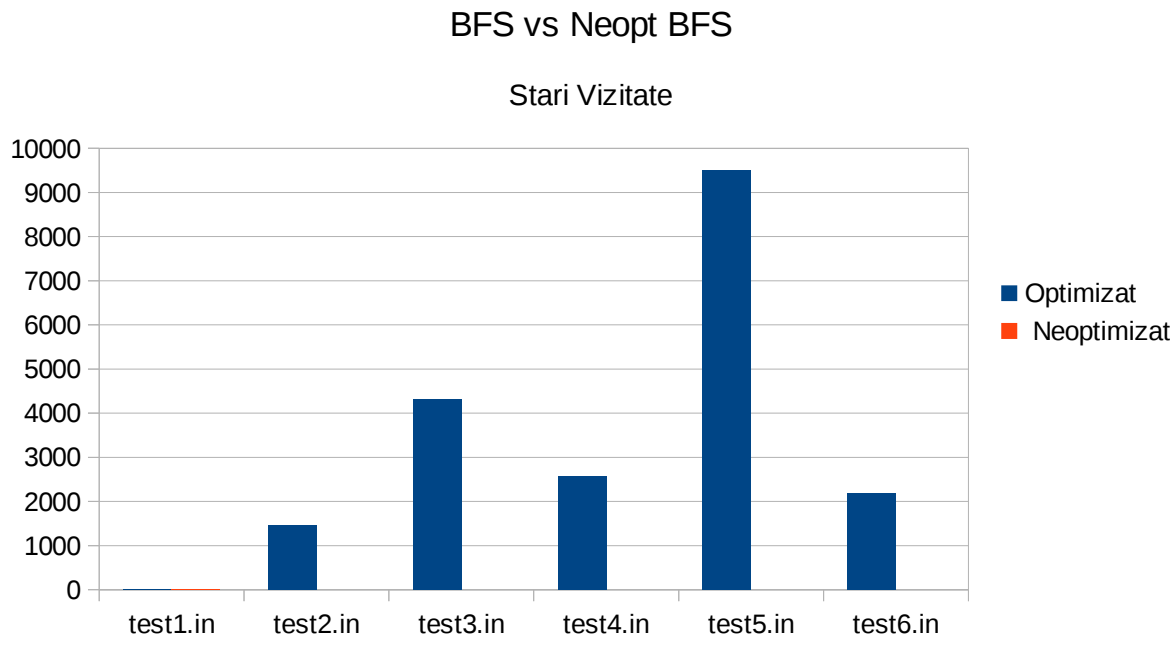
- optimal
- algoritm informat

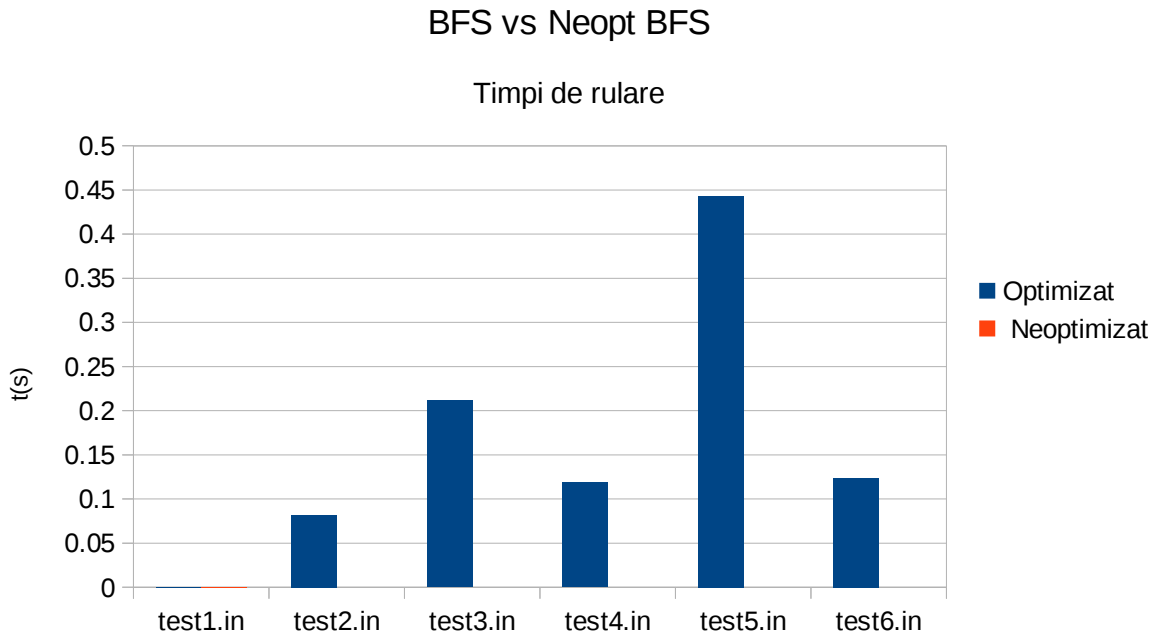
Dezavantaje RBFS:

- regenerare de noduri excesiva. Adică, atunci când se exploateaza un nod și se ajunge la un punct unde trebuie explorat altul de pe o alta cale, toată parcurgerea acelui nod este ștearsă. Cu toate acestea, dacă ajungem într-un alt punct în care nodul care a fost explorat anterior devine iarăși cea mai buna optiune, calea din acel nod va fi reexplorata.

## 2. Comparație variante optimizate cu variante neoptimizate

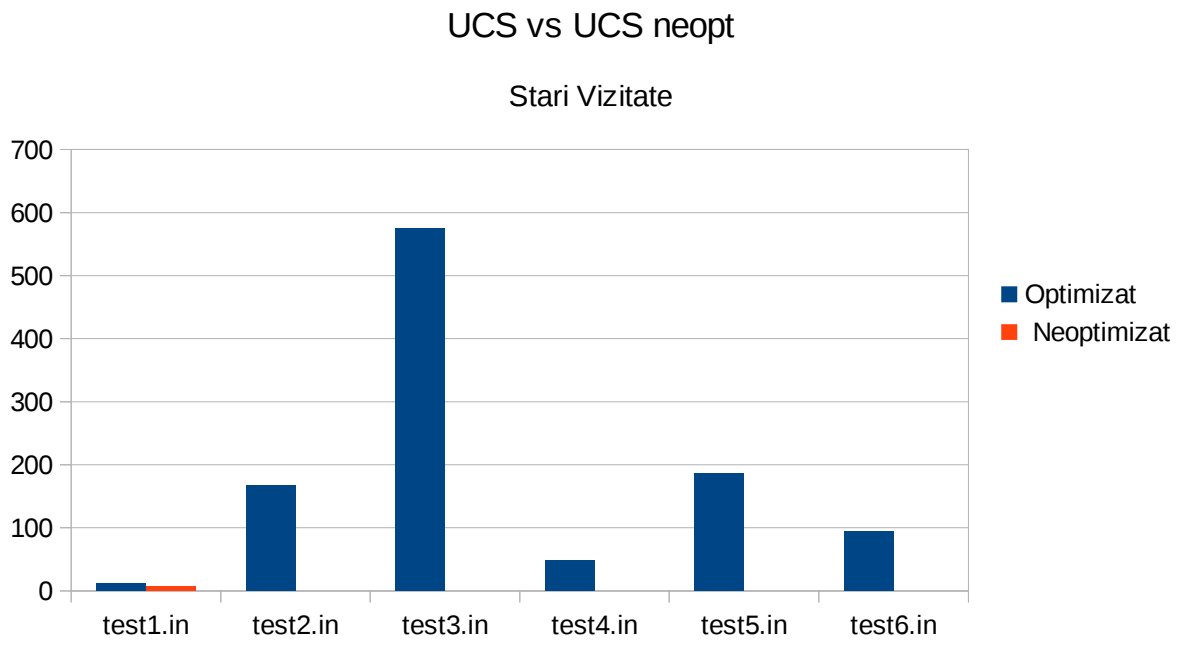
A. BFS optimizat vs BFS neoptimizat

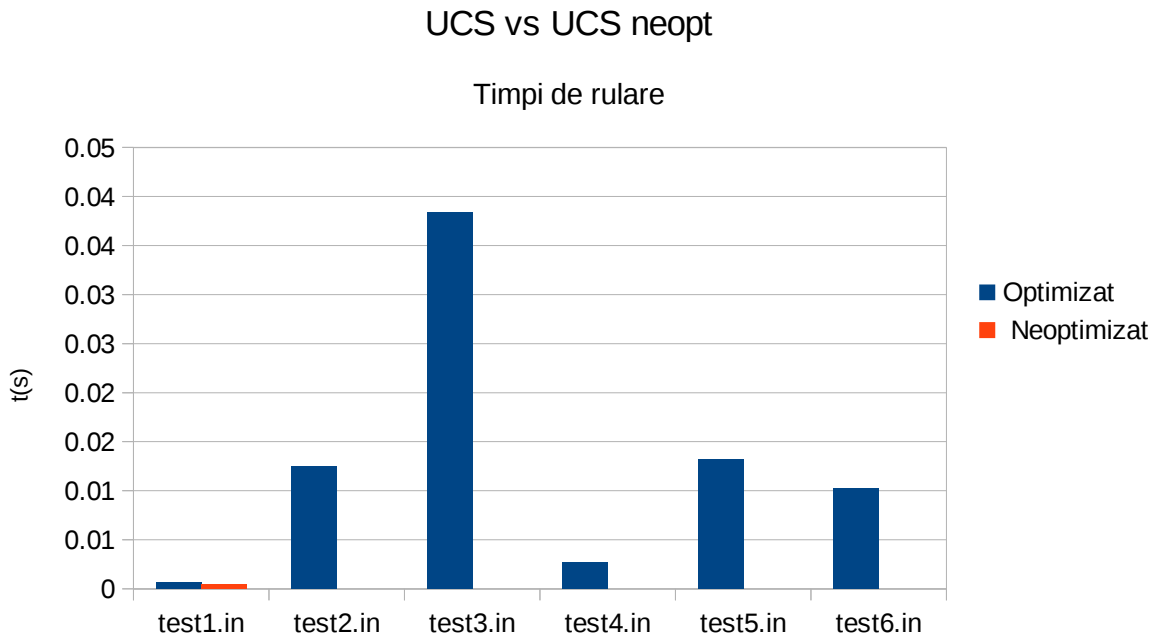


Pentru varianta de BFS optimizata se observa atât timpul de rulare cât şi numărul de stari vizitate, pe când pentru varianta neoptimizata se observa aceste 2 lucruri doar pentru testul 1. Varianta neoptimizata de BFS este atât de neeficienta încât în afara de testul 1, nici un alt test nu ruleaza într-un timp acceptabil, de aceea nu apar în graficele de mai sus.

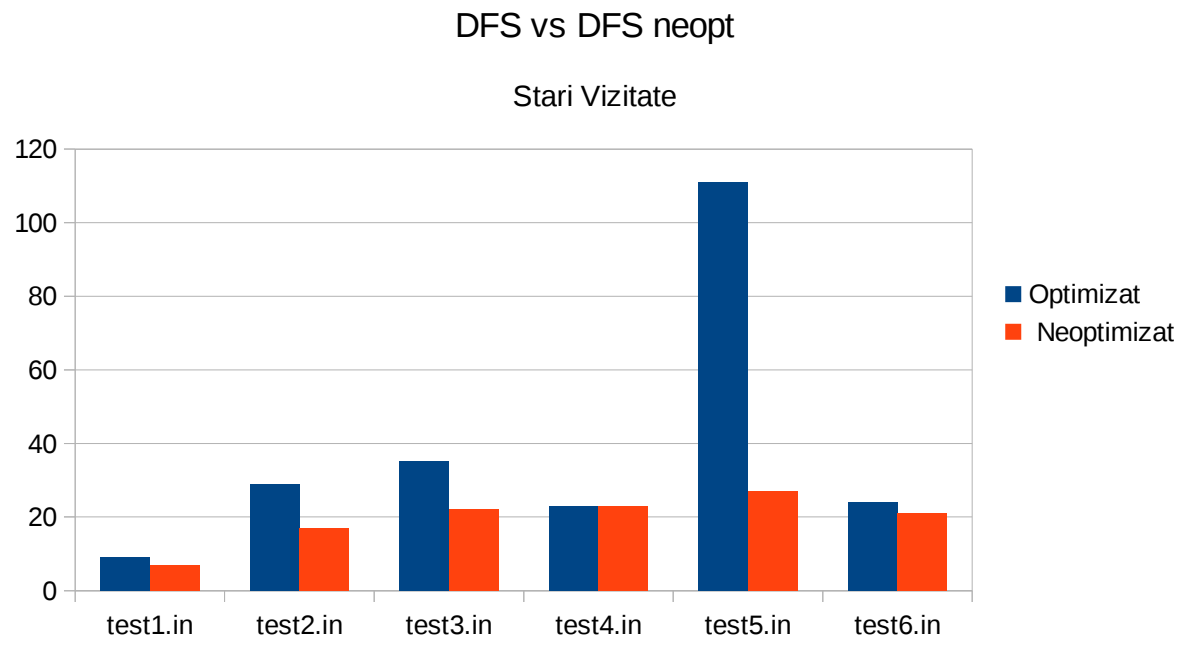
B. UCS optimizat vs UCS neoptimizat

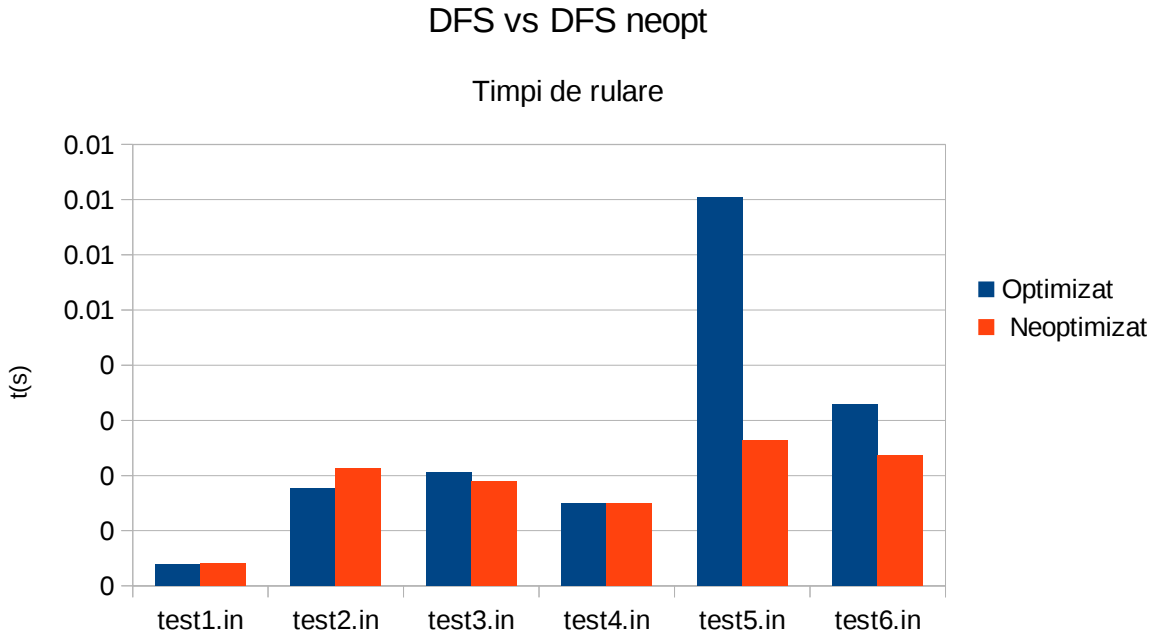


Asemănător cu cazul BFS și pentru UCS neoptimizat se termina într-un timp acceptabil doar testul 1. Deci, imbunatatirile aduse de optimizari sunt considerabile.

C. DFS optimizat vs DFS neoptimizat



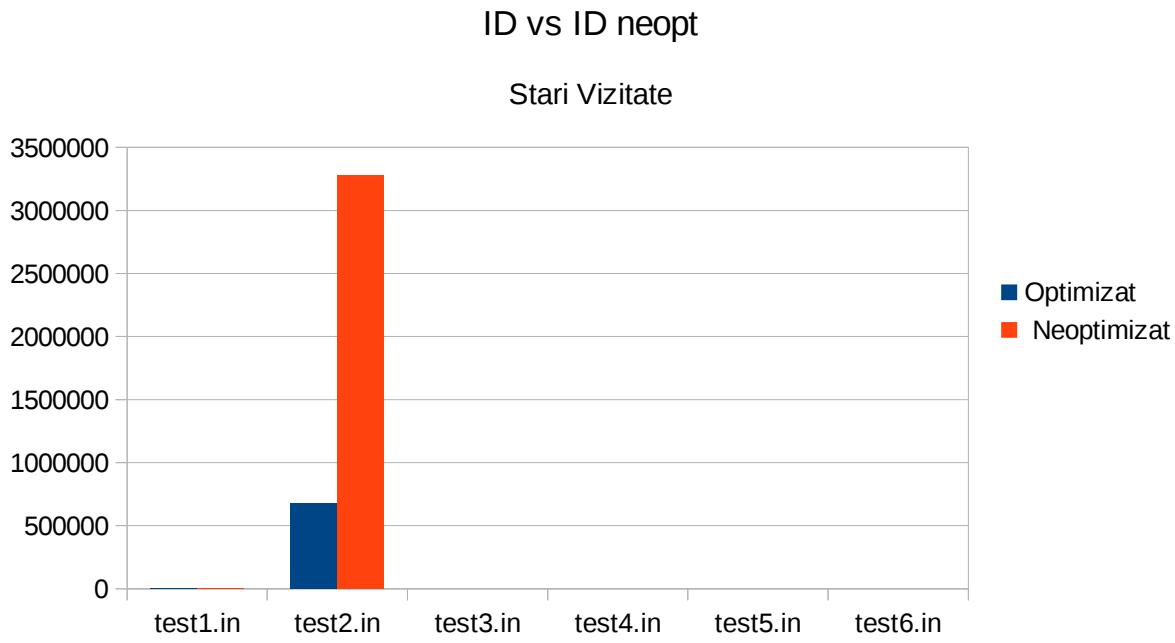



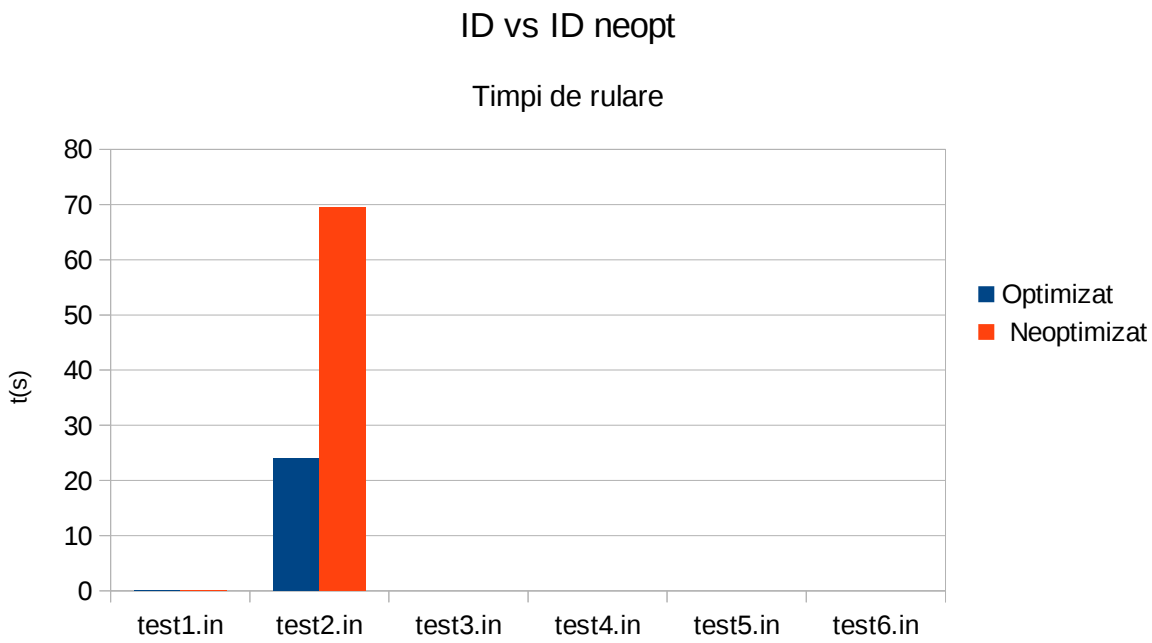
Pentru cazul DFS nu se observa o imbunatatire radicala de la varianta optimizata la varianta neoptimizata.

D. DLS optimizat vs DLS neoptimizat

Pentru cazul DLS, comparatia este identica cu cea de la DFS.

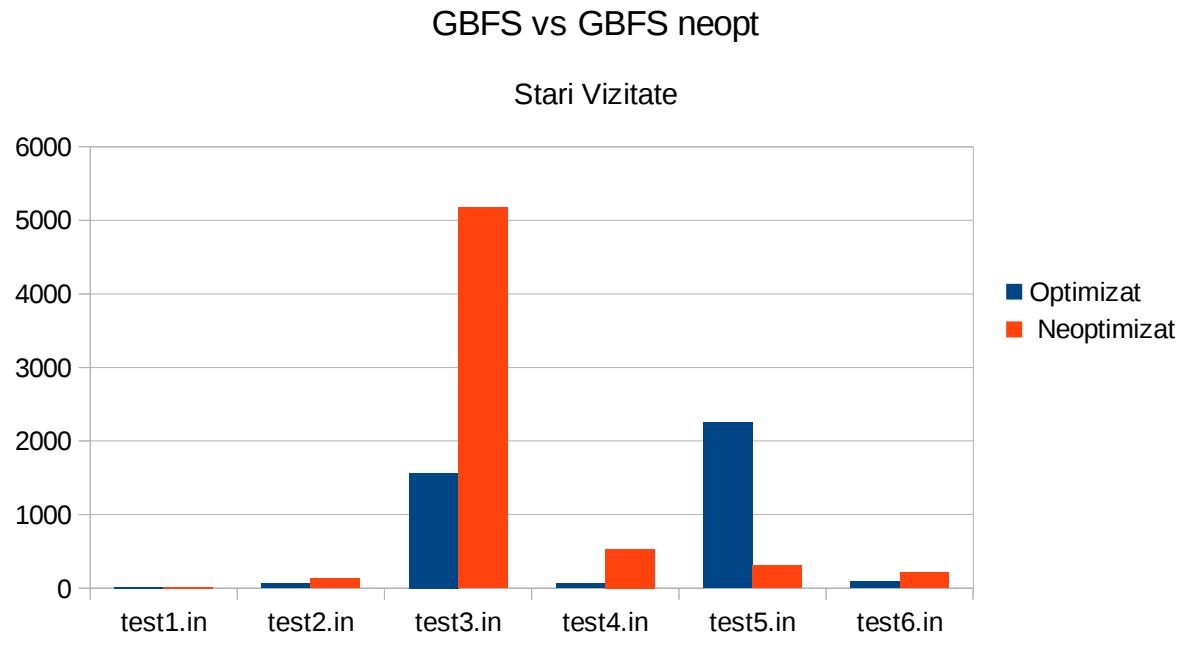
E. ID optimizat vs ID neoptimizat

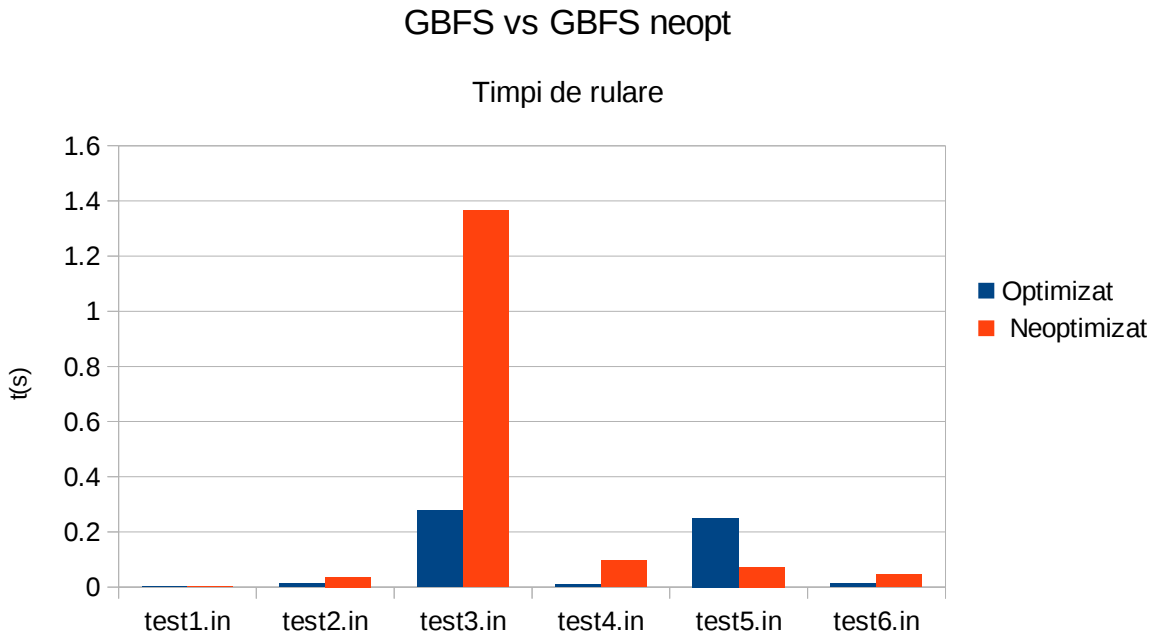


Pentru varianta ID optimizata și cea neoptimizata se observa o diferența majora intre cele 2. Se observa ca varianta optimizata, în cazul testului 2, durează de aproape 3 ori mai puțin, și vizitează de 4 ori mai puține stari. Atât varianta optimizata cât și cea neoptimizata nu ruleaza într-un timp acceptabil pentru testele de la 3 la 6. Pentru varianta optimizata, acest lucru se întâmpla deoarece exista un anumit interval de adancimi (0, n) pentru care nu exista nici o soluție disponibila și astfel trebuie parcurse toate caile din acel interval de cai (cu adâncimea data). De asemenea, la adâncimea k, se reexploreaza toate caile de la k-1, acest lucru fiind neeficient.

F. GBFS optimizat vs GBFS neoptimizat



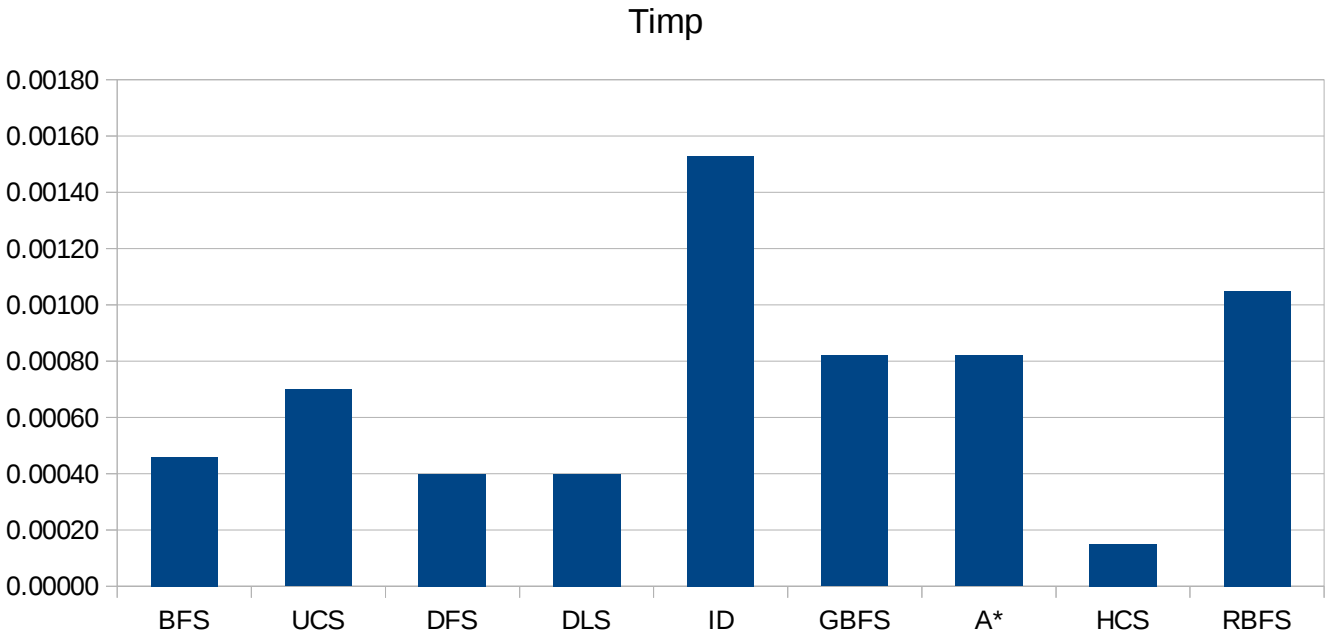



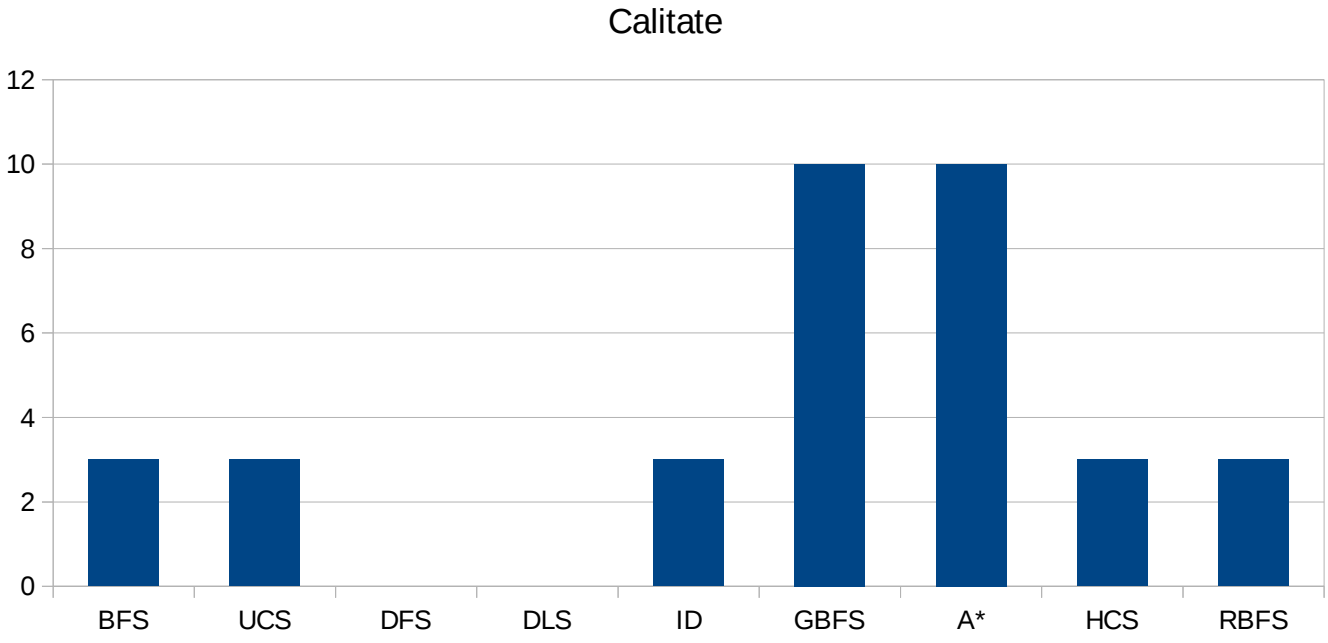
Deși și pentru GBFS neoptimizat, se termina de rulat toate cele 6 teste, se observa niște imbunatatiri considerabile pentru varianta optimizata. Numărul de stari vizitate scade foarte mult și automat si timpii de rulare.

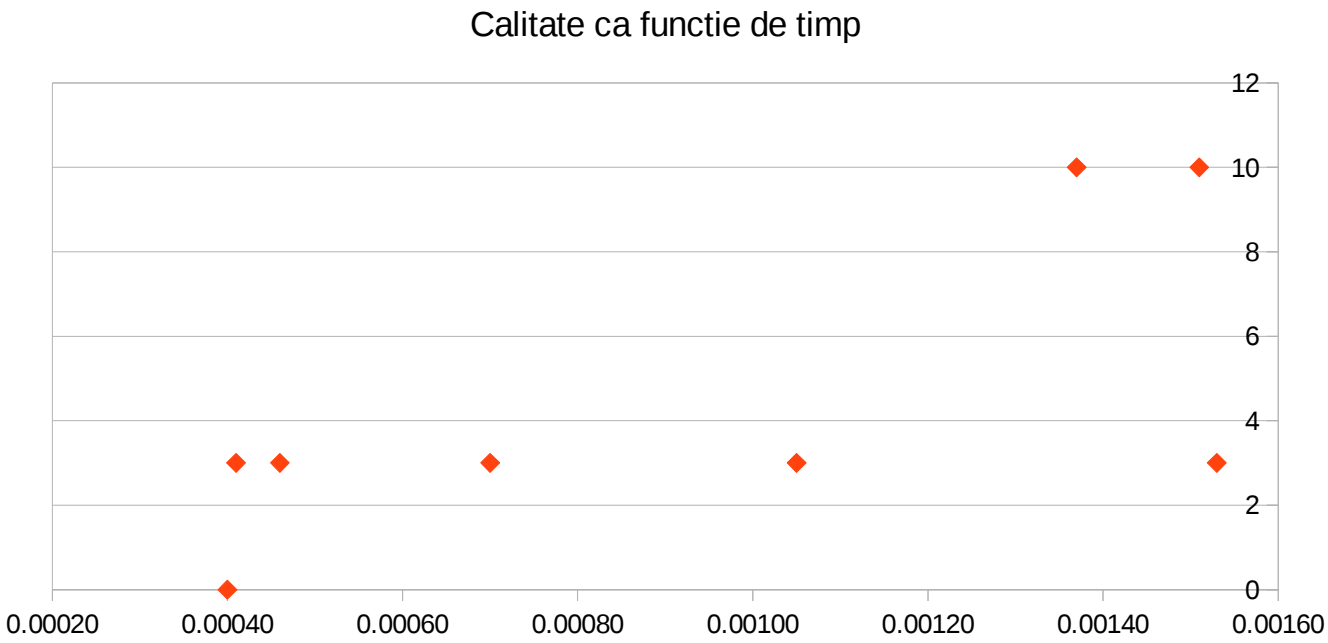
### 3. Comparare timpi de rulare intre algoritmi și comparare calitatea solutiei

În continuare vom lua în calcul doar variantele optimizate de la algoritmii neinformati cât și de la GBFS. Pentru unele teste, timpul va fi reprezentat pe o scala logaritmica deoarece diferența intre timpi ar fi prea mare (intre ID și RBFS fata de ceilalți algoritmi)

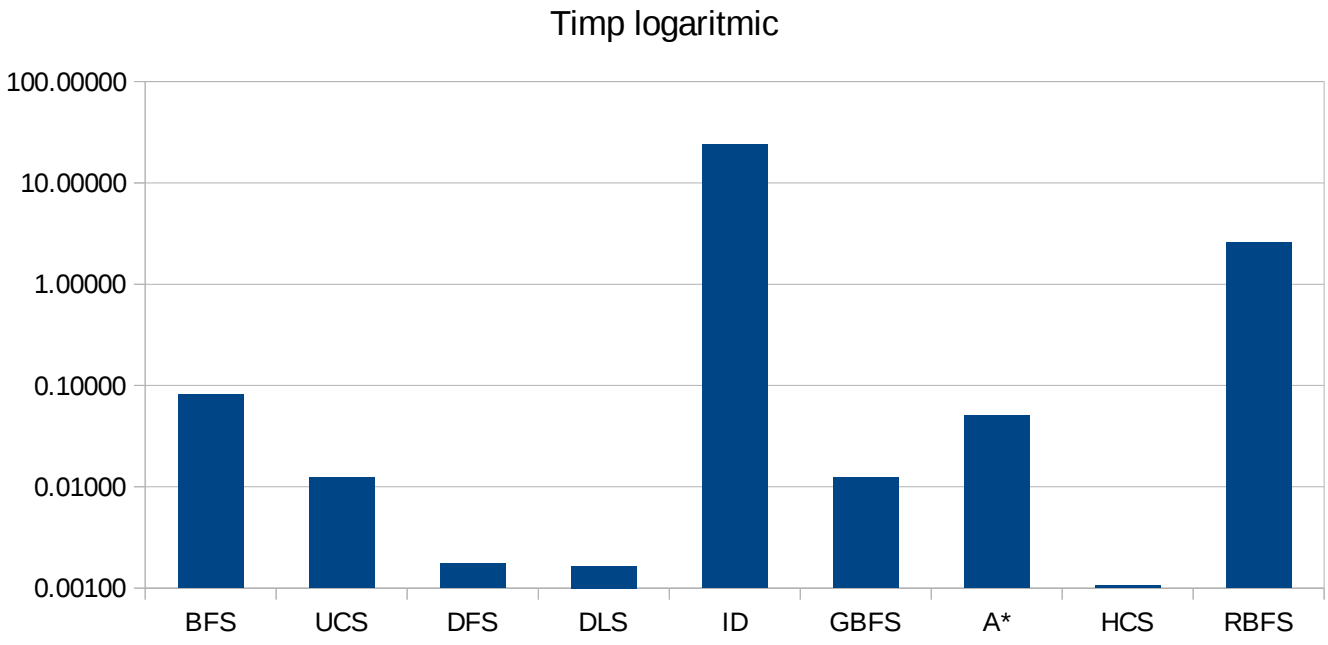
***Test1***

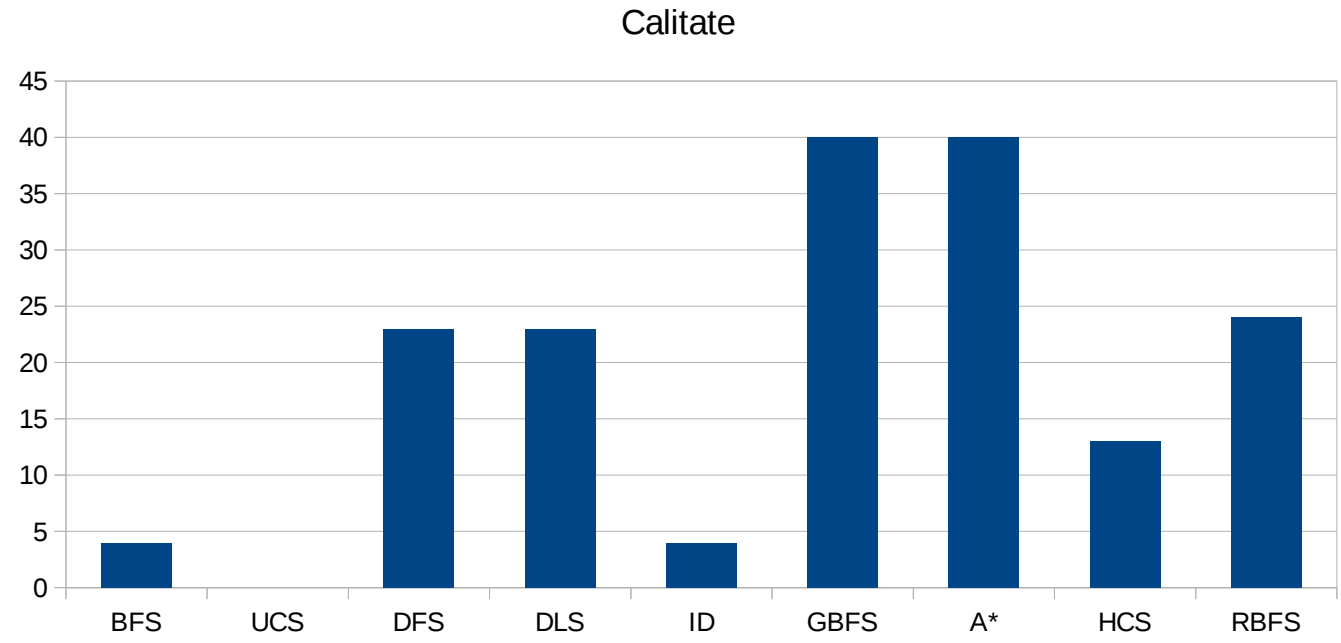


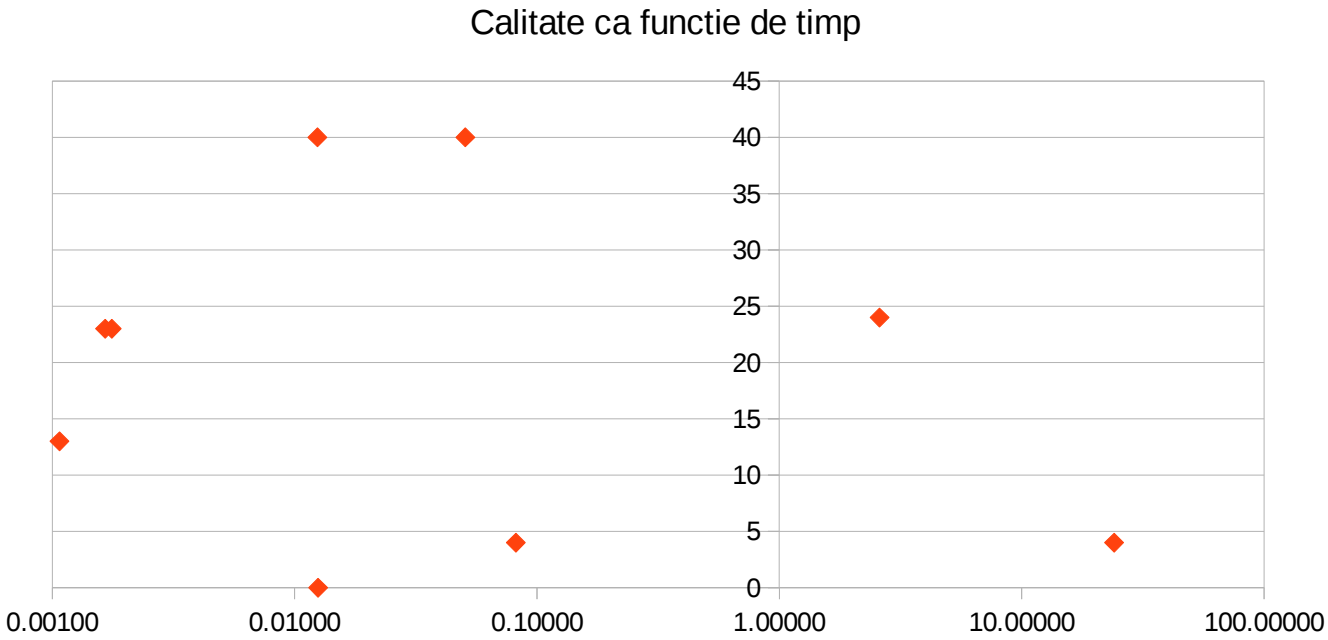




Calitate ca functie de timp
12
10
8
6
4
2
0
0.00020
0.00040
0.00060
0.00080
0.00100
0.00120
0.00140
0.00160

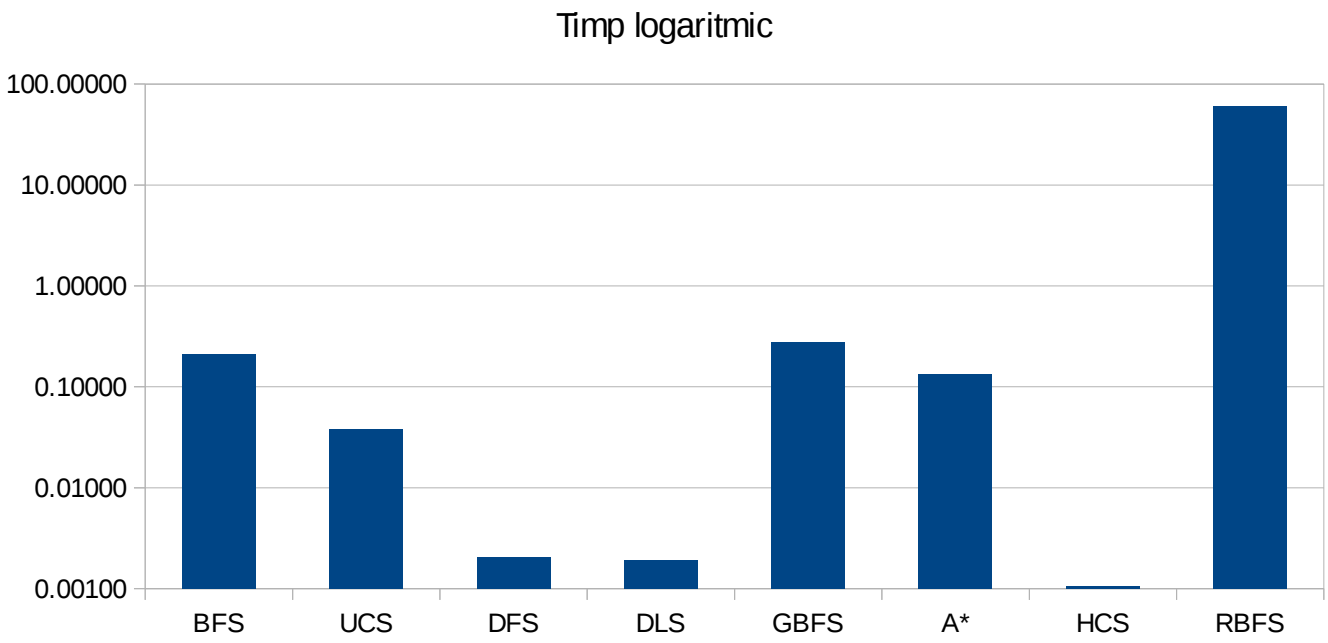
***Test2***

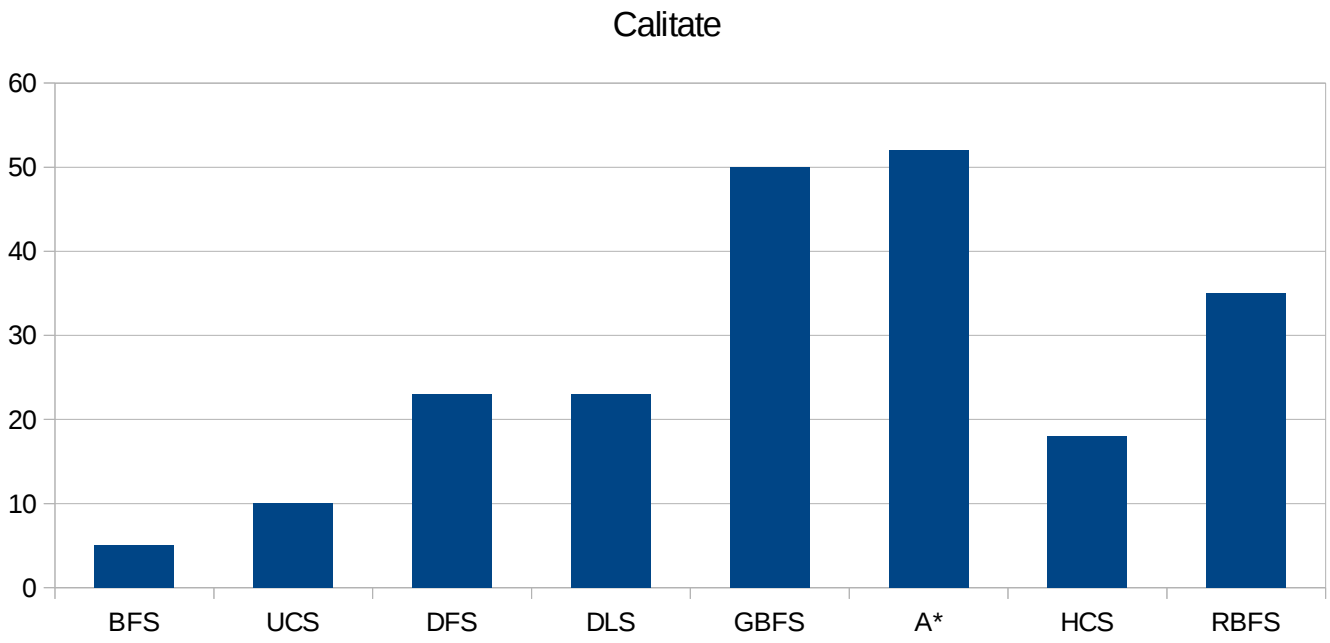

Timp logaritmic
100.00000
10.00000
1.00000
0.10000
0.01000
0.00100
BFS
UCS
DFS
DLS
ID
GBFS
A*
HCS
RBFS

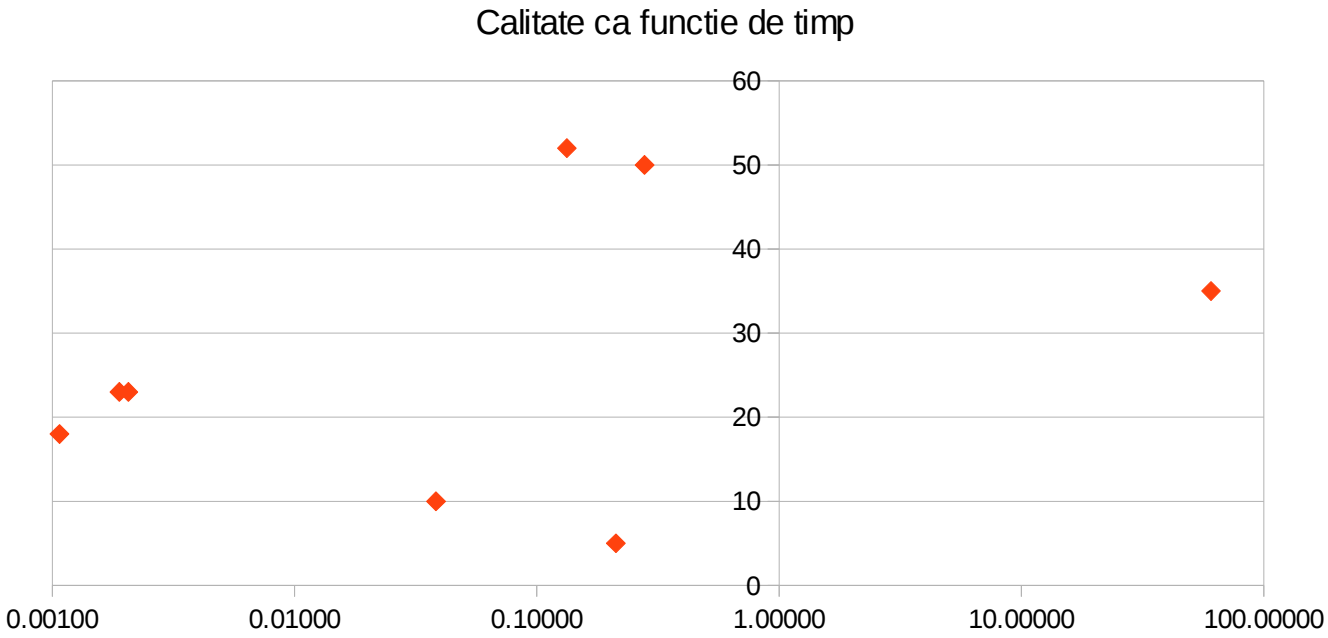

Calitate
45
40
35
30
25
20
15
10
5
0
BFS
UCS
DFS
DLS
ID
GBFS
A*
HCS
RBFS


Calitate ca functie de timp
45
40
35
30
25
20
15
10
5
0
0.00100
0.01000
0.10000
1.00000
10.00000
100.00000

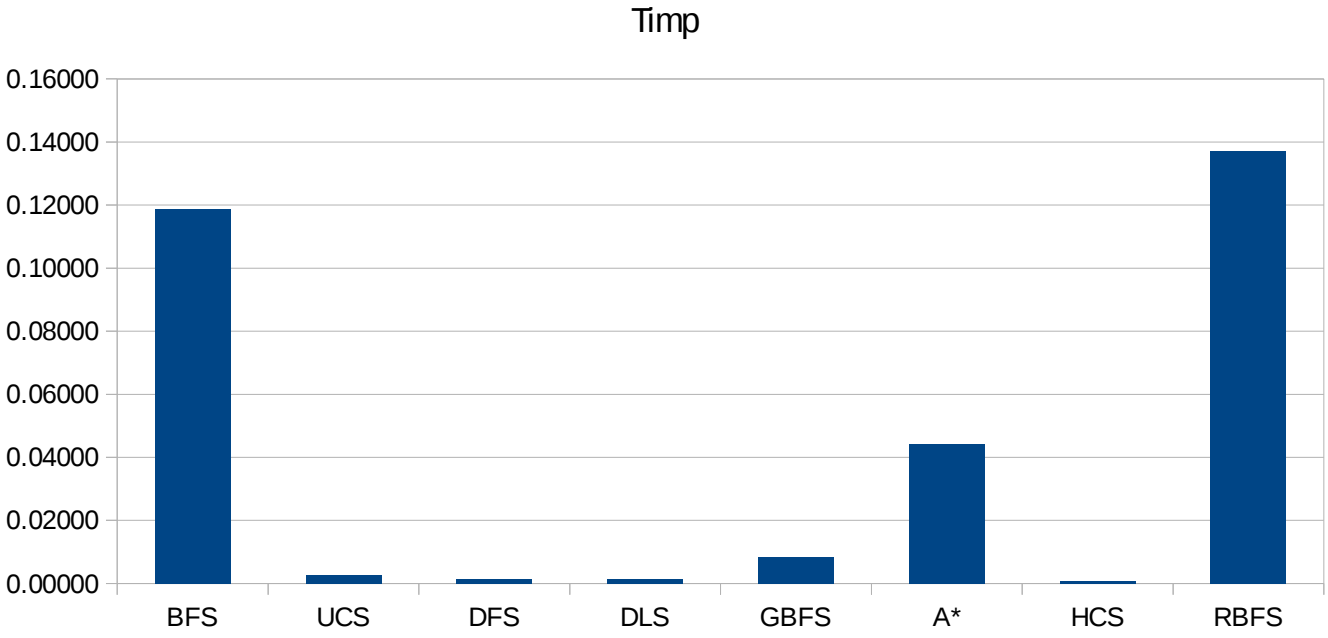
***Test3***

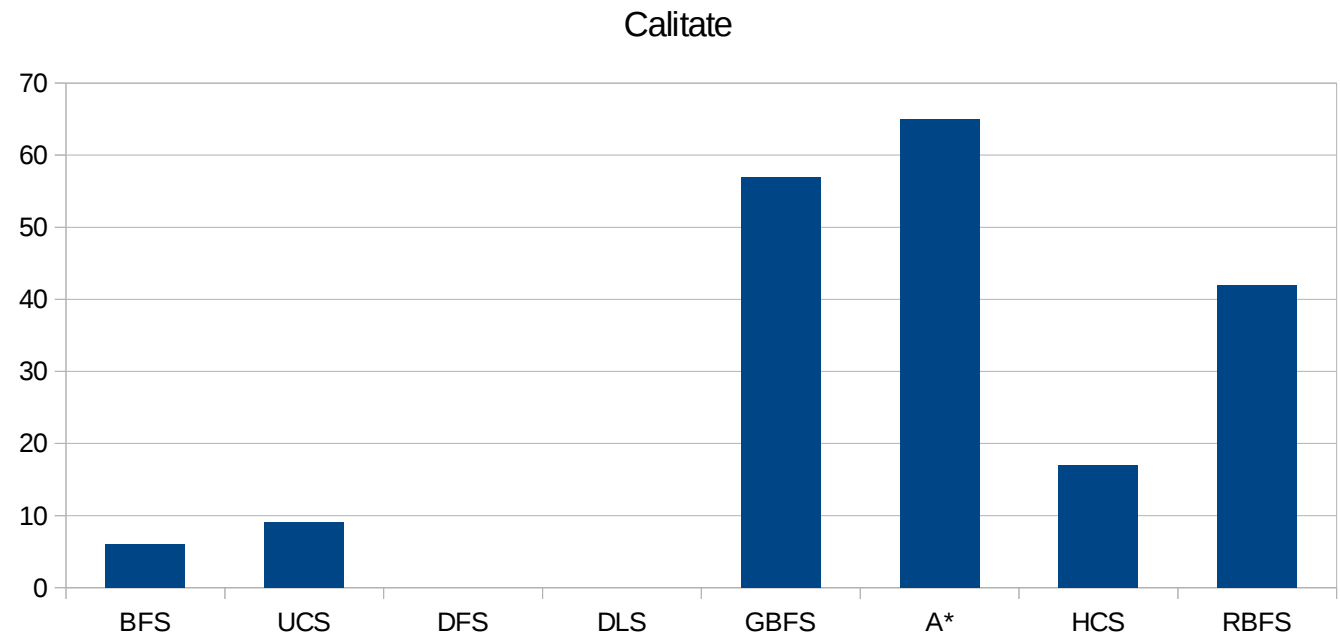

Timp logaritmic
100.00000
10.00000
1.00000
0.10000
0.01000
0.00100
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS

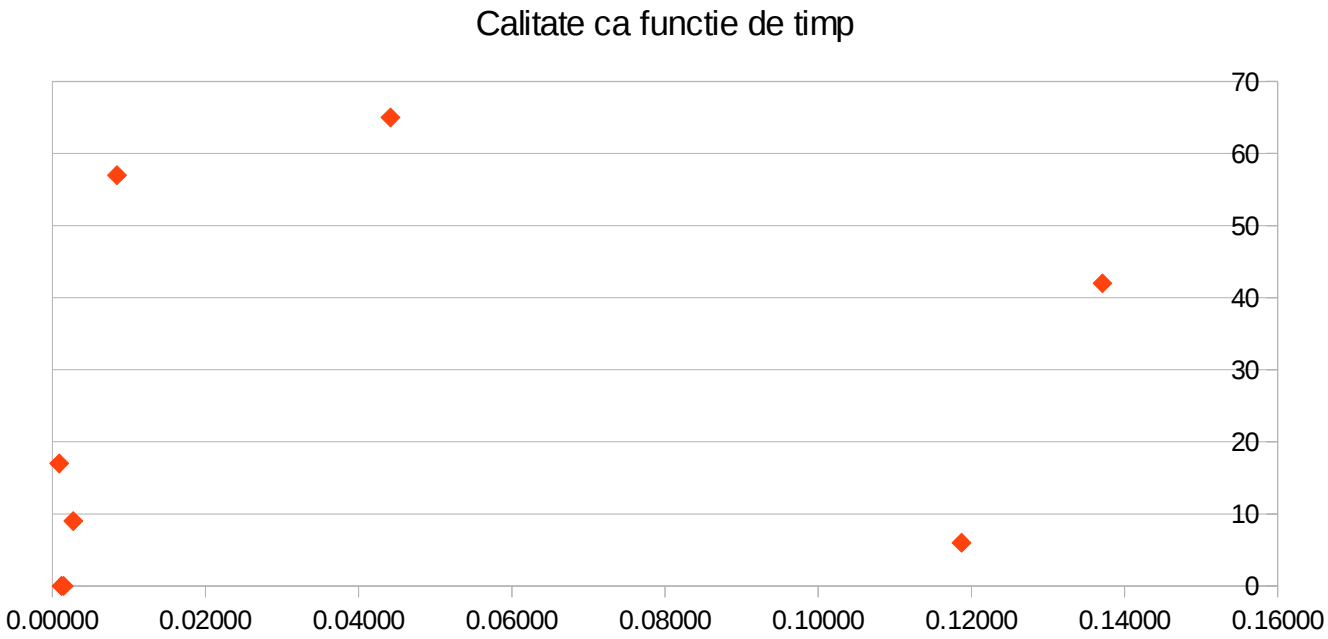

Calitate
60
50
40
30
20
10
0
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS


Calitate ca functie de timp
60
50
40
30
20
10
0
0.00100
0.01000
0.10000
1.00000
10.00000
100.00000

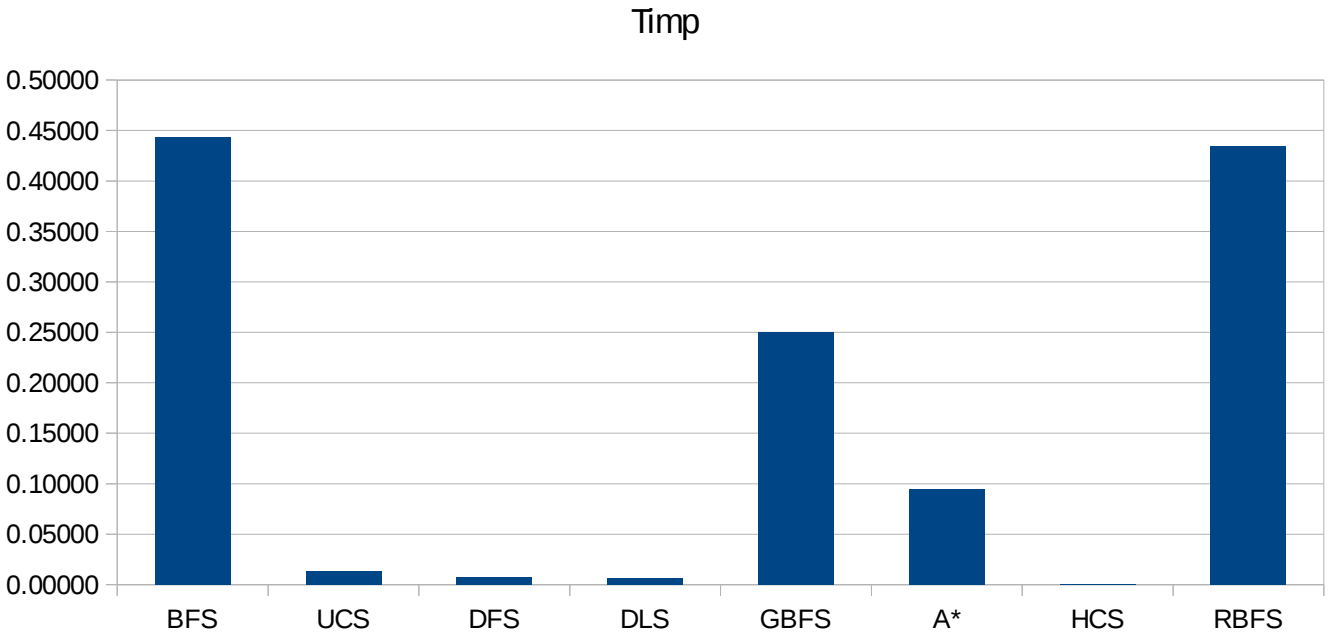
***Test4***

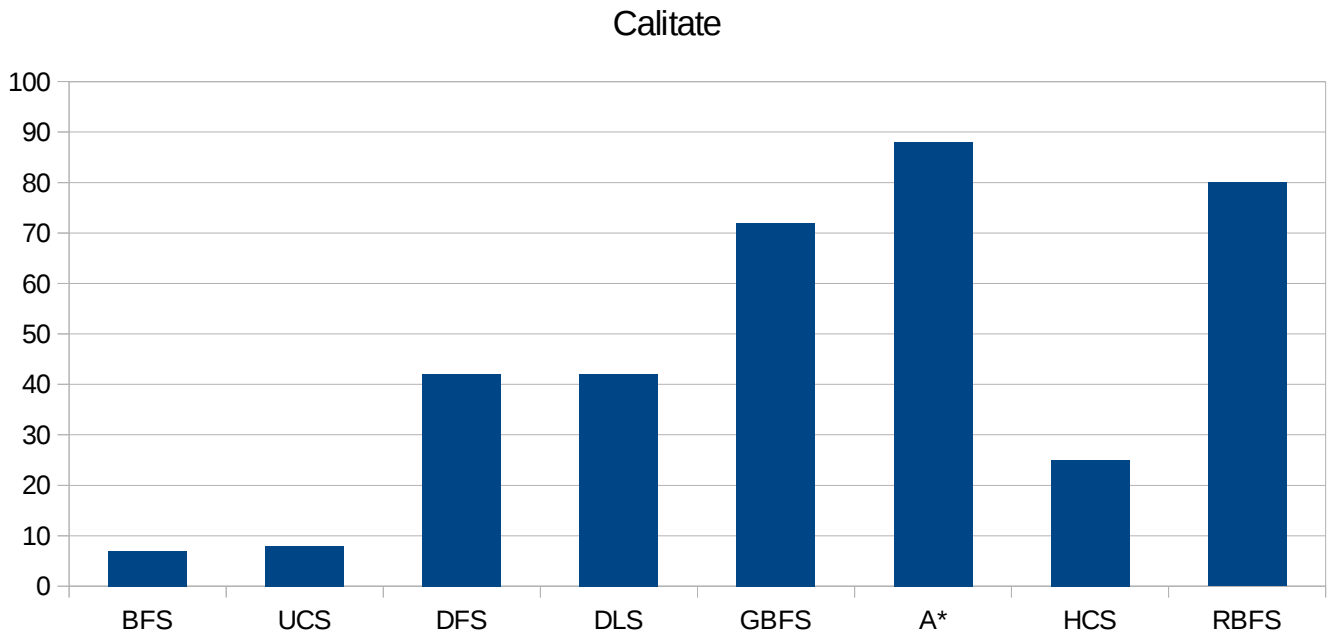

Timp
0.16000
0.14000
0.12000
0.10000
0.08000
0.06000
0.04000
0.02000
0.00000
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS

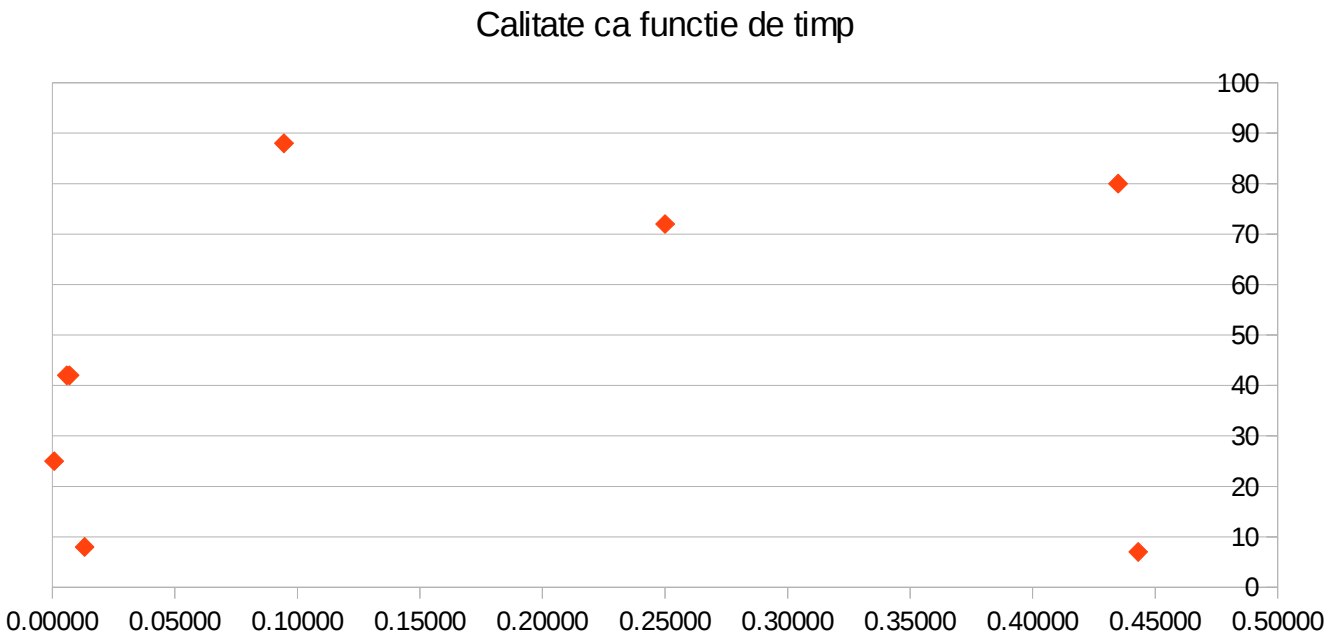

Calitate
70
60
50
40
30
20
10
0
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS


Calitate ca functie de timp
70
60
50
40
30
20
10
0
0.00000
0.02000
0.04000
0.06000
0.08000
0.10000
0.12000
0.14000
0.16000

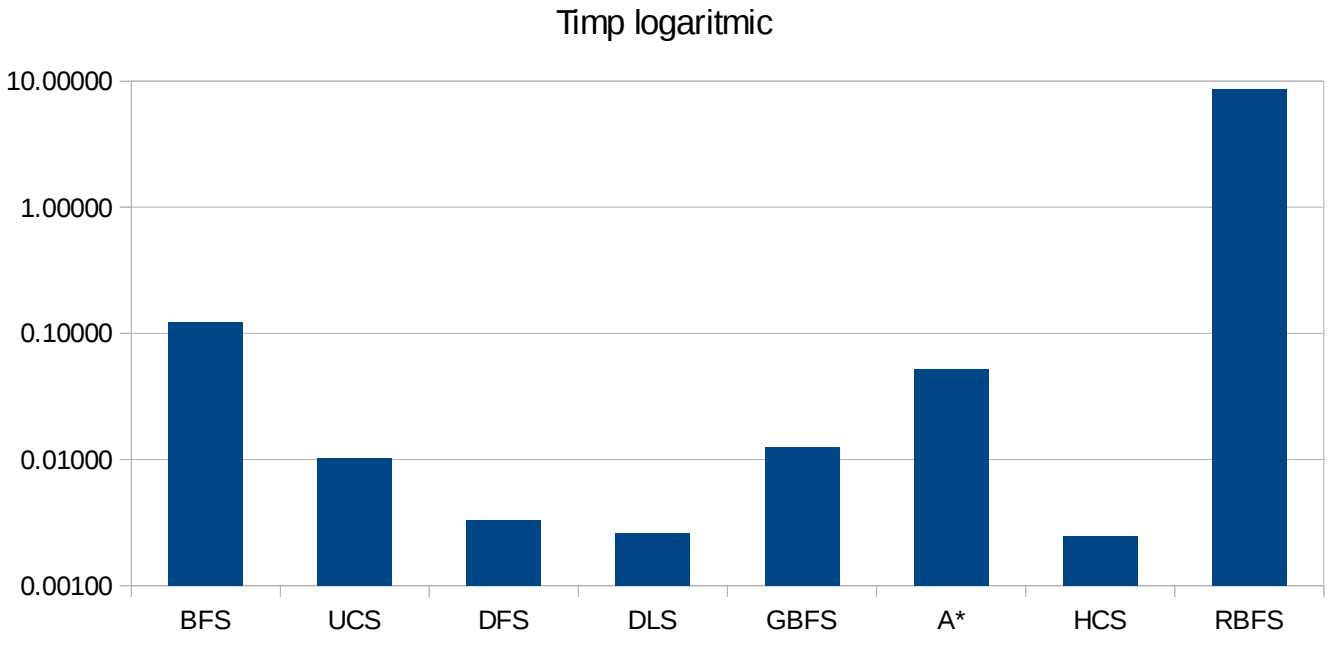
***Test5***

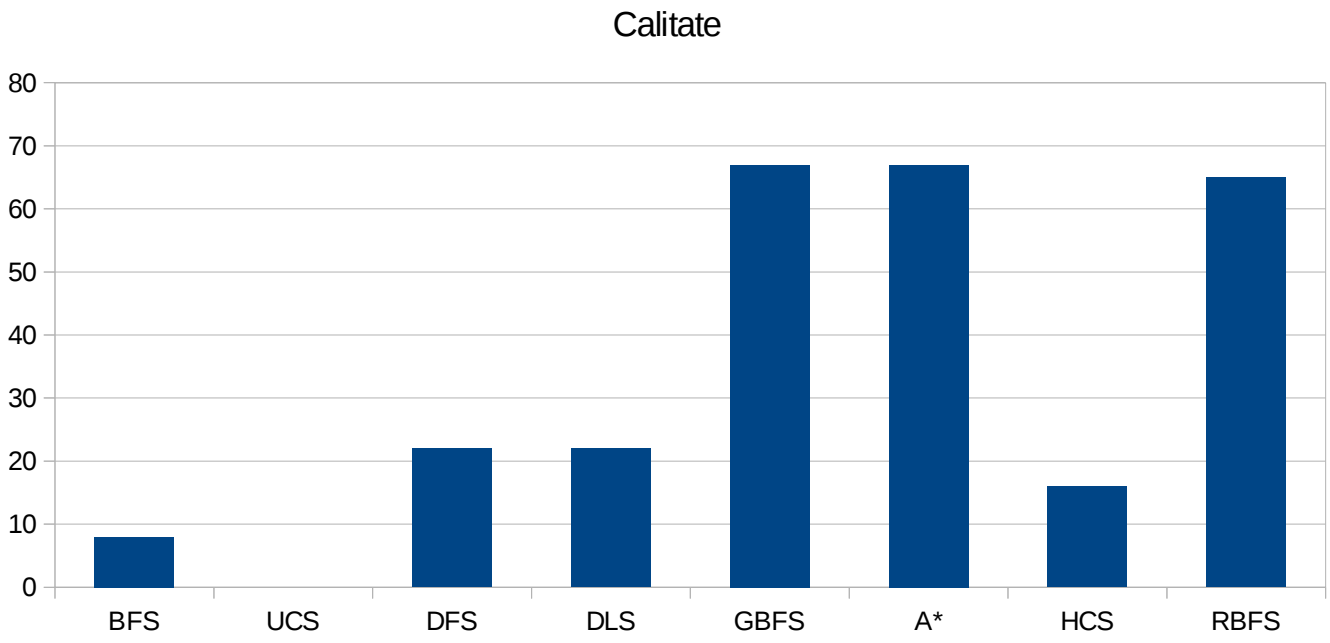

Timp
0.50000
0.45000
0.40000
0.35000
0.30000
0.25000
0.20000
0.15000
0.10000
0.05000
0.00000
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS

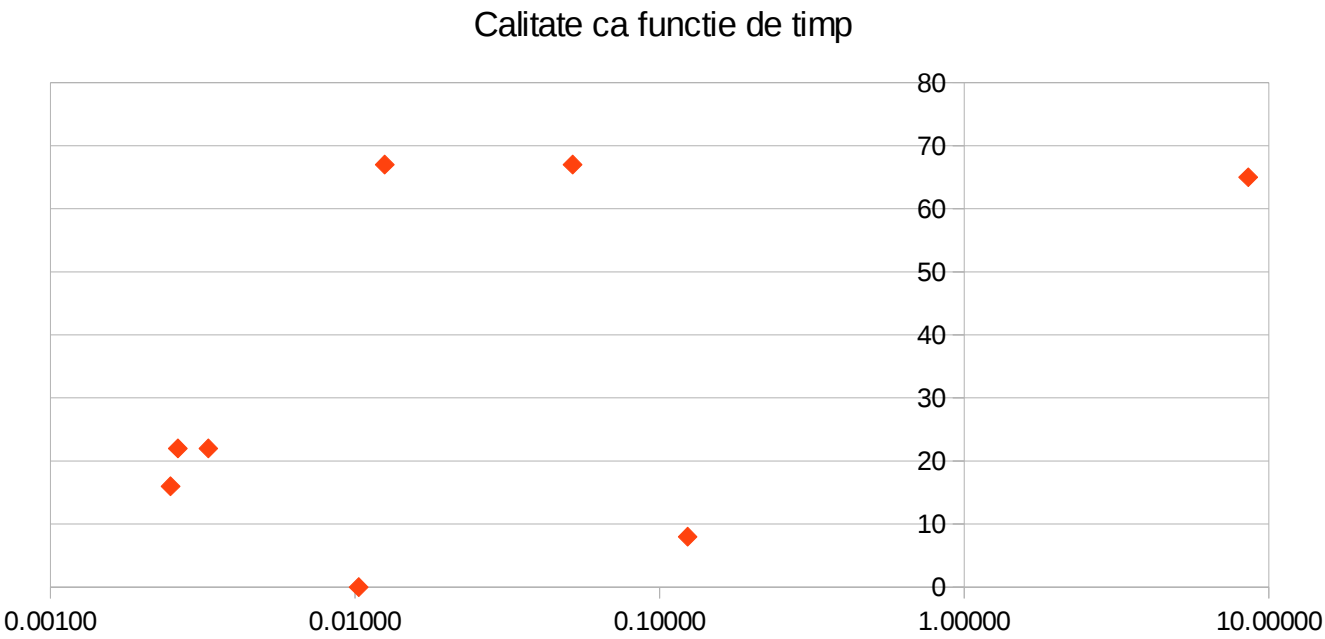

Calitate
100
90
80
70
60
50
40
30
20
10
0
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS


Calitate ca functie de timp
100
90
80
70
60
50
40
30
20
10
0
0.00000
0.05000
0.10000
0.15000
0.20000
0.25000
0.30000
0.35000
0.40000
0.45000
0.50000

***Test6***


Timp logaritmic
10.00000
1.00000
0.10000
0.01000
0.00100
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS


Calitate
80
70
60
50
40
30
20
10
0
BFS
UCS
DFS
DLS
GBFS
A*
HCS
RBFS


Calitate ca functie de timp
80
70
60
50
40
30
20
10
0
0.00100
0.01000
0.10000
1.00000
10.00000

Concluzie:

După cum se observa din graficele prezentate mai sus, A* este cea mai buna alegere pentru o problemă de acest tip, deoarece ne ofera o soluție cu o calitate foarte buna, într-un timp mediu spre mic. Pe lângă A* o alta soluție rezonabila este GBFS, care, de multe ori, ne ofera o soluție comparativa calitativ cu A*, de multe ori, într-un timp puțin mai mic.

RBFS ne ofera de asemenea o soluție care este destul de buna din punct de vedere calitativ, însă pe lângă A* și GBFS acesta ruleaza într-un timp mult mai mare. Acest lucru se întâmpla deoarece este necesar ca unele noduri să fie reexplorate.

HCS nu este o soluție viabila pentru problema noastră, cel puțin pentru cele 2 euristici folosite de mine. Acest lucru se datorează faptului ca, de multe ori, copii unui nod nu au neeaparat o valoare mai buna decât nodul părinte. Acest lucru se datoareaza faptului ca, de multe ori, când se trece în nodul copil se consuma benzina. Date fiind cele de mai sus, HCS se oprește după 1 sau 2 mutari, sau de cele mai multe ori după nici o mutare.

În ceea ce privește algoritmii neinformati, ID durează cel mai mult, urmat de BFS după care UCS, DFS, DLS. Pentru DLS, dacă nu ii este data o adâncime destul de mare astfel încât sa conțină o soluție, atunci el va explora toate starile care pot avea adâncimea curenta, pentru a vedea ca nu se găsește o soluție. În testele mele, am dat DLS – ului o adâncime destul de mare, și de multe ori ajunge sa găsească o soluție egala cu cea a DFS – ului. Pentru ID, după cum am zis și mai sus, nu ruleaza decât pentru testele 1 și 2. Acest lucru se întâmpla deoarece pentru testele mai mari, poate ajunge la adancimi de 20 – 30. În acest caz, sunt foarte multe stari de explorat, și reexplorat pana când se ajunge la o stare finala, iar acest lucru, cu puterea mea de procesoare nu ruleaza într-un timp acceptabil. În ceea ce privește calitatea, putem spune ca tine mai mult de noroc, și anume, care dintre acești algoritmi se „impiedica” de cea mai buna stare finala. În principiu, DFS și DLS ne ofera cel mai bun raport calitate timp, deoarece ruleaza foarte repede (chiar dacă pe unele teste obținem calitatea 0)

**4. Comparație intre cele 2 euristici**

Pentru început o stare finala este atunci când:

- fie am dus toți clientii și nu mai avem client în mașina

- fie am rămas fără benzina

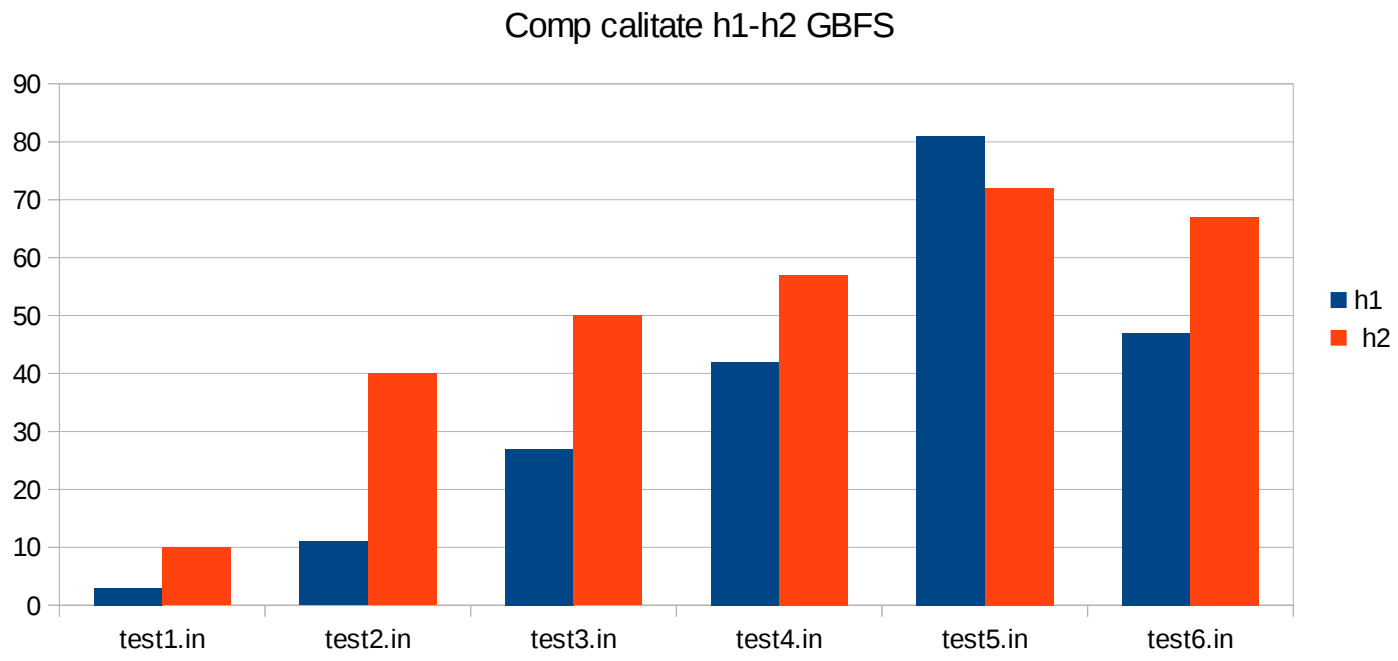
- fie cu benzina ramasa, nu mai putem face încă o cursa completă (distanța pana la client + distanța călătorie client)

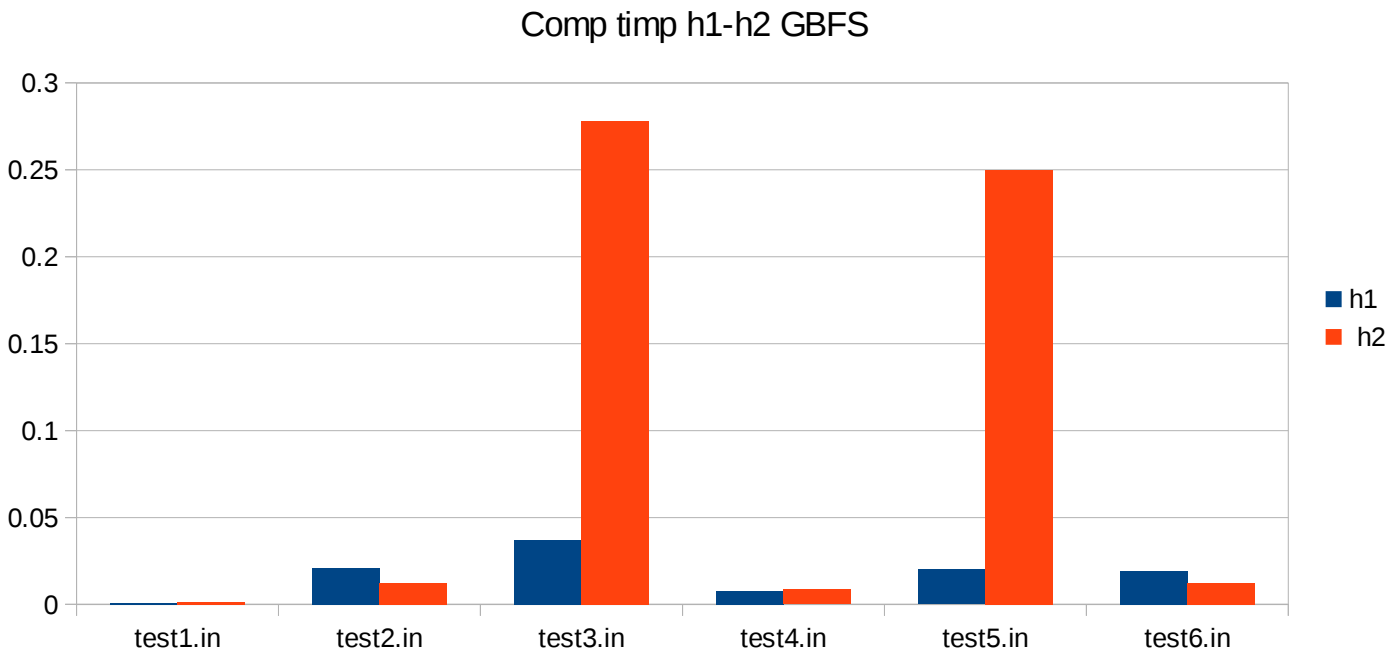
Cele 2 euristici alese de mine sunt urmatoarele:

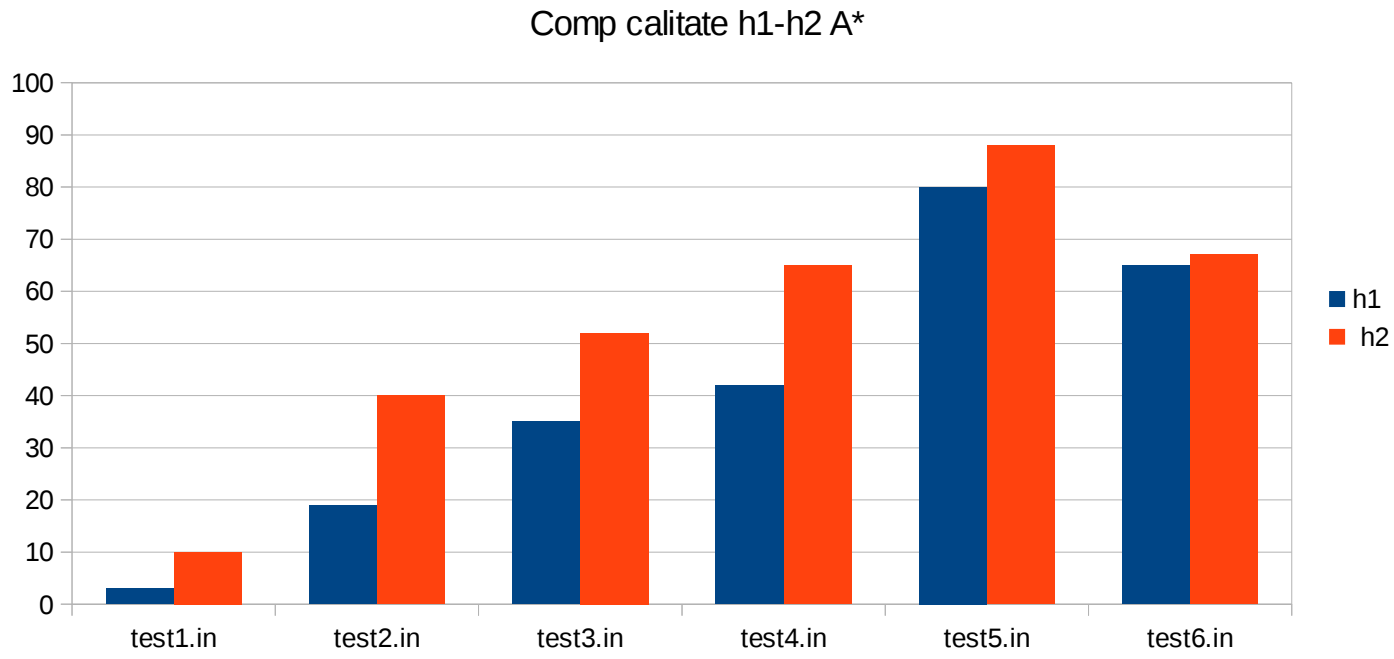
```
def h1(state):
    max_from_clients = sum([c for _, _, _, _, c in clients])

    return max(max_from_clients - state[fuel_idx] - state[venit], 0)

def h2(state):
    max_from_clients = sum([c for _, _, _, _, c in clients])

    if state[client] != -1:
        _, _, dx, dy,   = clients[state[client]]
        client_destination_weigh = abs(state[x_idx] - dx) + abs(state[y_idx] - dy)
    else:
        client_destination_weigh = 0

    return max_from_clients - state[venit] - possible_profit(state) \
        + get_distance_to_best_reachable_client(state) + client_destination_weigh * 2
```

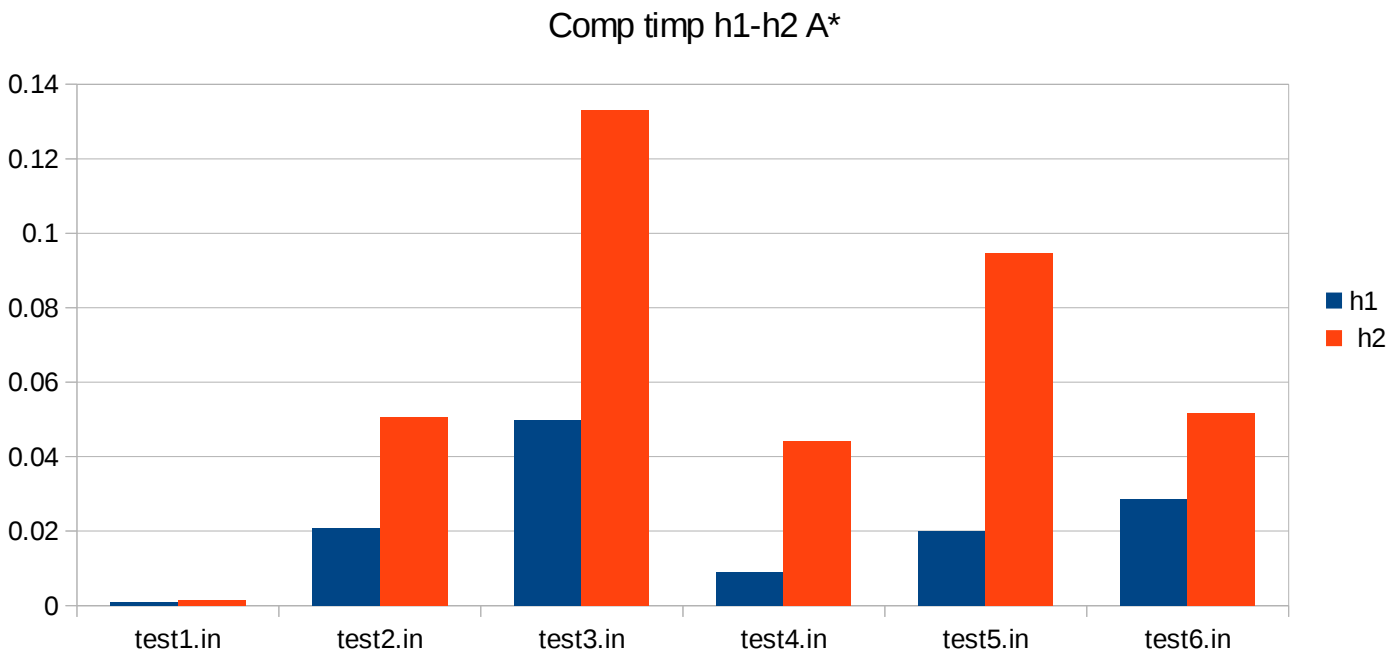
Prima euristica este o euristica mai simpla. Se calculează venitul maxim pe care îl putem obtine de la clienți, din care se scade benzina curenta și venitul curent.

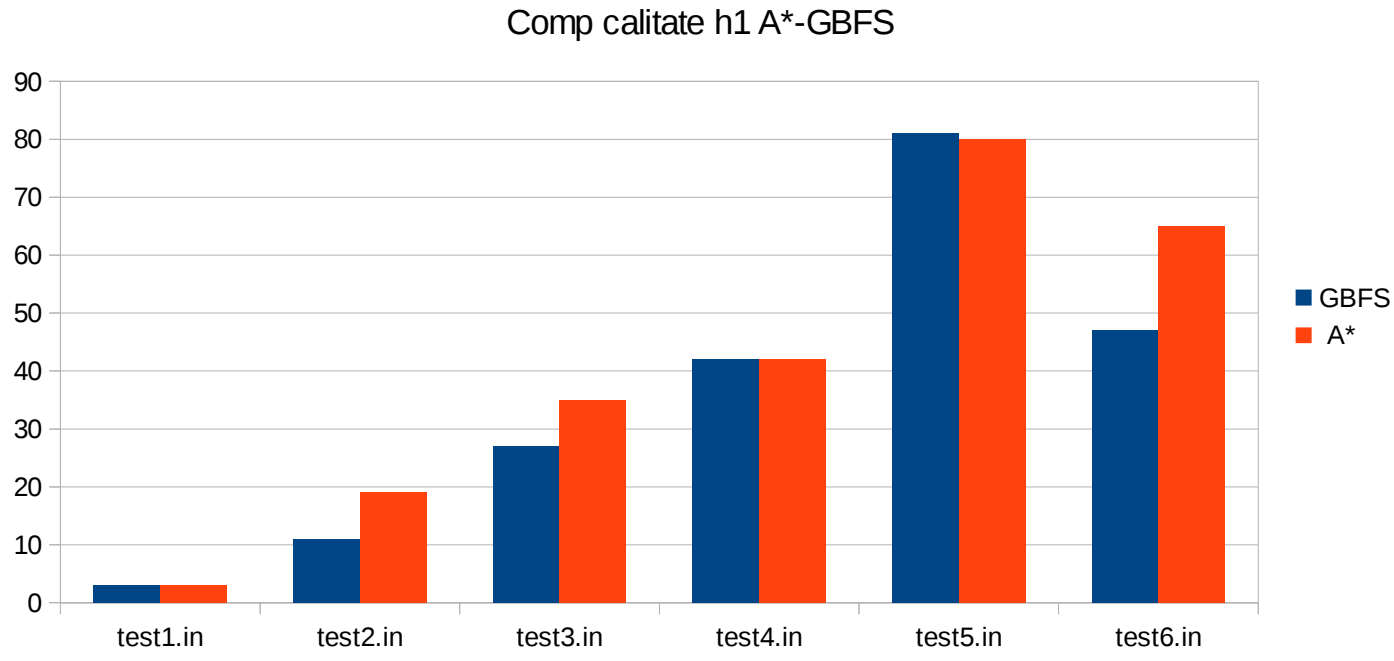
Cea dea doua euristica este ceva mai complexa, ia în calcul venitul curent, o aproximare a venitul posibil pe care îl putem obtine din aceasta stare, cât și distanța către cel mai bun client, sau dacă avem deja un client, distanța către destinația acestuia.

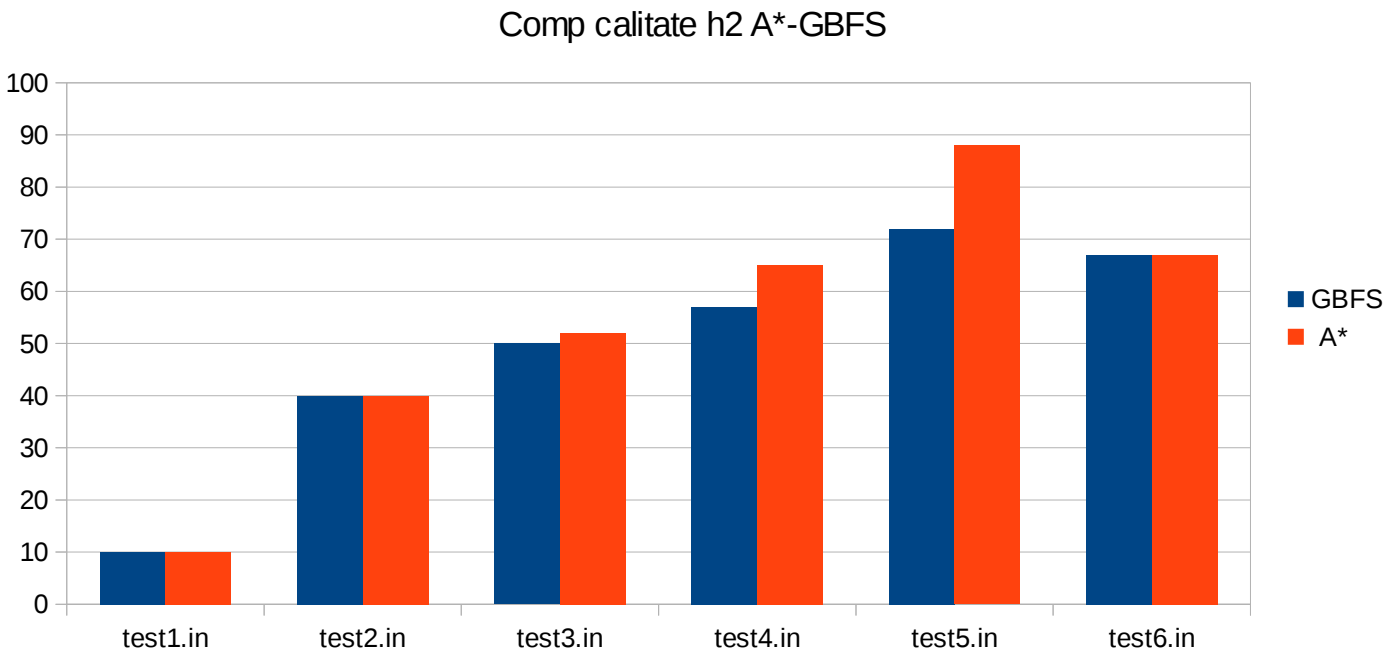



Comp timp h1-h2 GBFS
0.3
0.25
0.2
0.15
0.1
0.05
0
test1.in
test2.in
test3.in
test4.in
test5.in
test6.in
h1
h2


Comp calitate h1-h2 A*
100
90
80
70
60
50
40
30
20
10
0
test1.in
test2.in
test3.in
test4.in
test5.in
test6.in
h1
h2


Comp timp h1-h2 A*
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
test1.in
test2.in
test3.in
test4.in
test5.in
test6.in
h1
h2


Comp calitate h1 A*-GBFS
90
80
70
60
50
40
30
20
10
0
test1.in
test2.in
test3.in
test4.in
test5.in
test6.in
GBFS
A*




Deoarece algoritmul HCS nu a fost deloc potrivit pentru problema data, nu l-am mai inclus în compararea euristicilor, deoarece pentru ambele euristici se comporta asemănător, și anume se oprea după prima sau după primele stari.

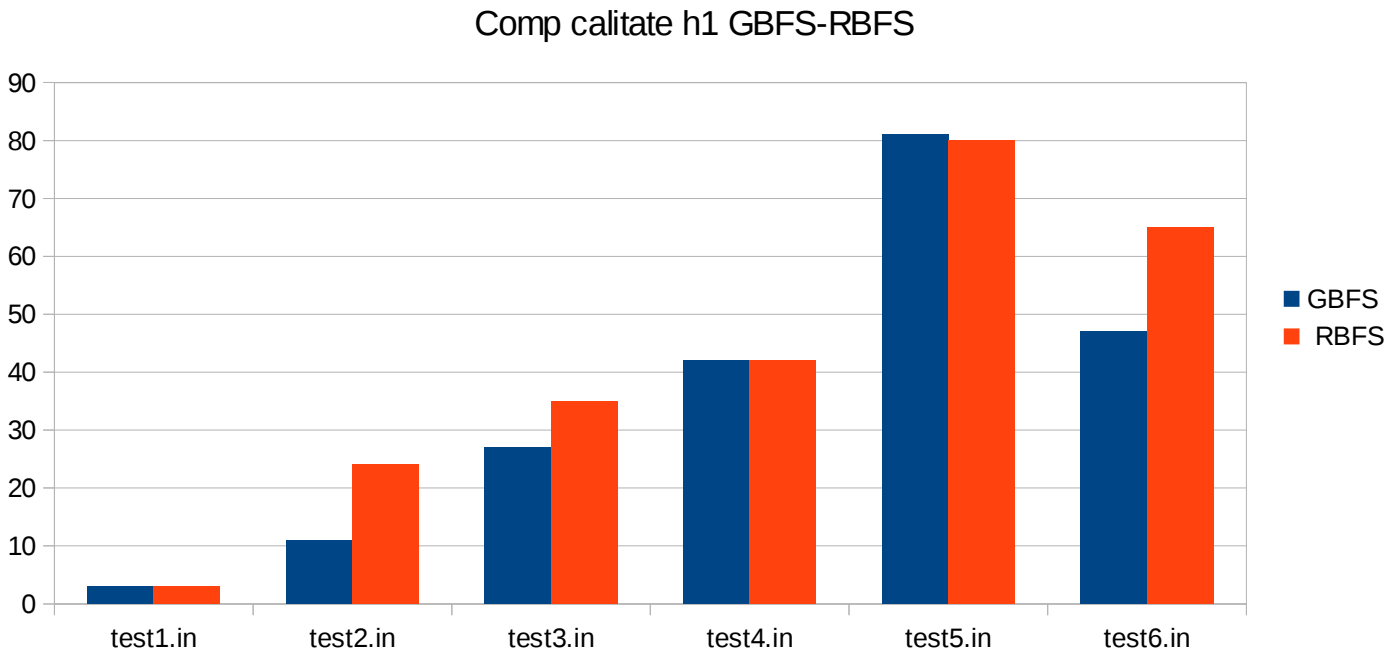
Așa cum se observa din primele 2 grafice, așa cum ma așteptam, euristica h1 obține rezultate puțin mai proaste decât euristica h2 pentru toate testele, mai puțin testul 5 pentru algoritmul GBFS. Acest lucru era de așteptat deoarece euristica h2 este mult mai complexa decât euristica h1, și reprezintă mult mai bine potențialul unei stari.

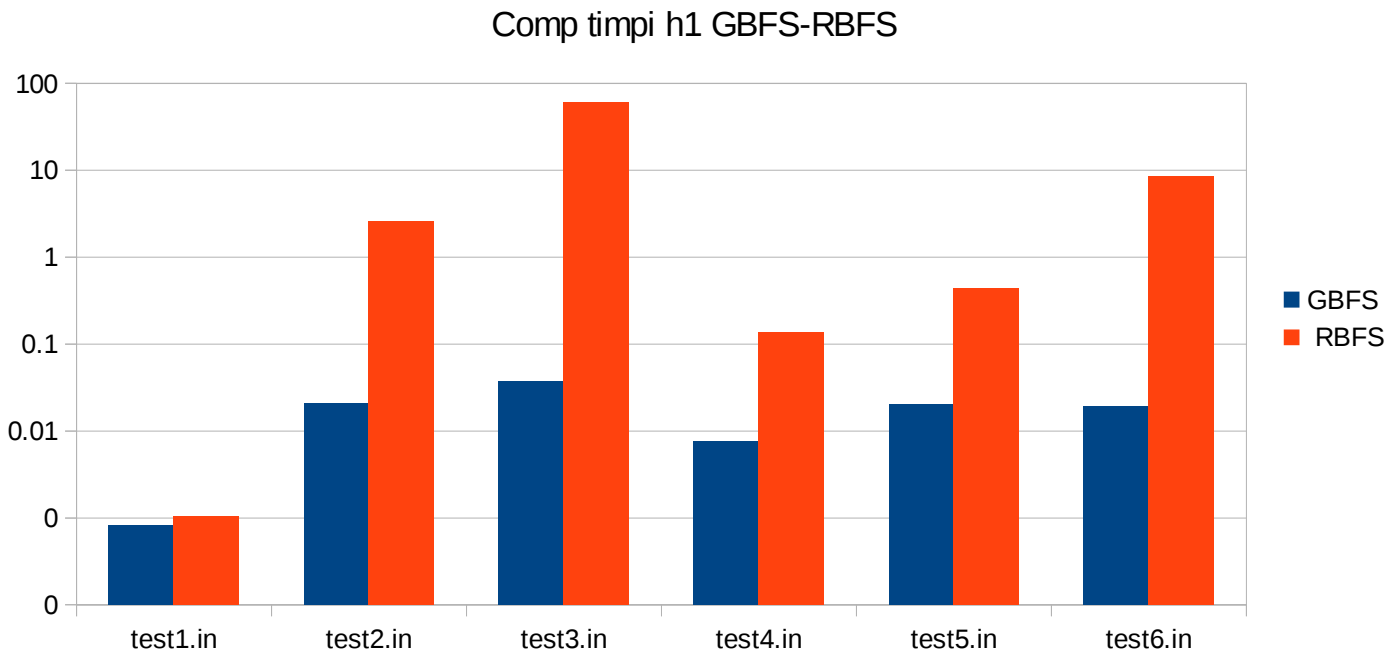
În ceea ce privește comparatia intre algoritmul GBFS și algoritmul A* pentru euristicile folosite, se observa ca atât pentru euristica h1 cât și pentru euristica h2 algoritmii au rezultate apropiate, deși algorimul A* are valorile mai bune în toate cazurile, mai puțin pentru testul 5.

Deși soluția h2 ne ofera rezultate calitative mai bune, se observa ca rularea algoritmilor durează mai mult. Acest lucru se datorează faptului ca sunt necesare mai multe calcule pentru euristica h2.

**5. Comparare GBFS cu RBFS pentru euristica h1**

Am ales sa compar GBFS cu RBFS pentru euristica h1 deoarece pentru euristica h2 obtineam timp mult prea mare de rulare.



După cum se vede din cele 2 grafice, RBFS ofera rezultate mai bune decât GBFS, cu toate acestea, nu se justifica folosirea RBFS deoarece timpii de rulare sunt mult mai mari (de aceea a trebuit sa folosesc scara logaritmica pentru graficul cu timpi). Acești timpi de rulare mai mari se obțin deoarece RBFS re-viziteaza anumite noduri (adică tot arborele care pleacă din acel nod) ceea ce îl face foarte ineficient.

În concluzie, deși RBFS obtine rezultate puțin mai bune, nu se poate justifica folosirea sa în locul GBFS – ului.