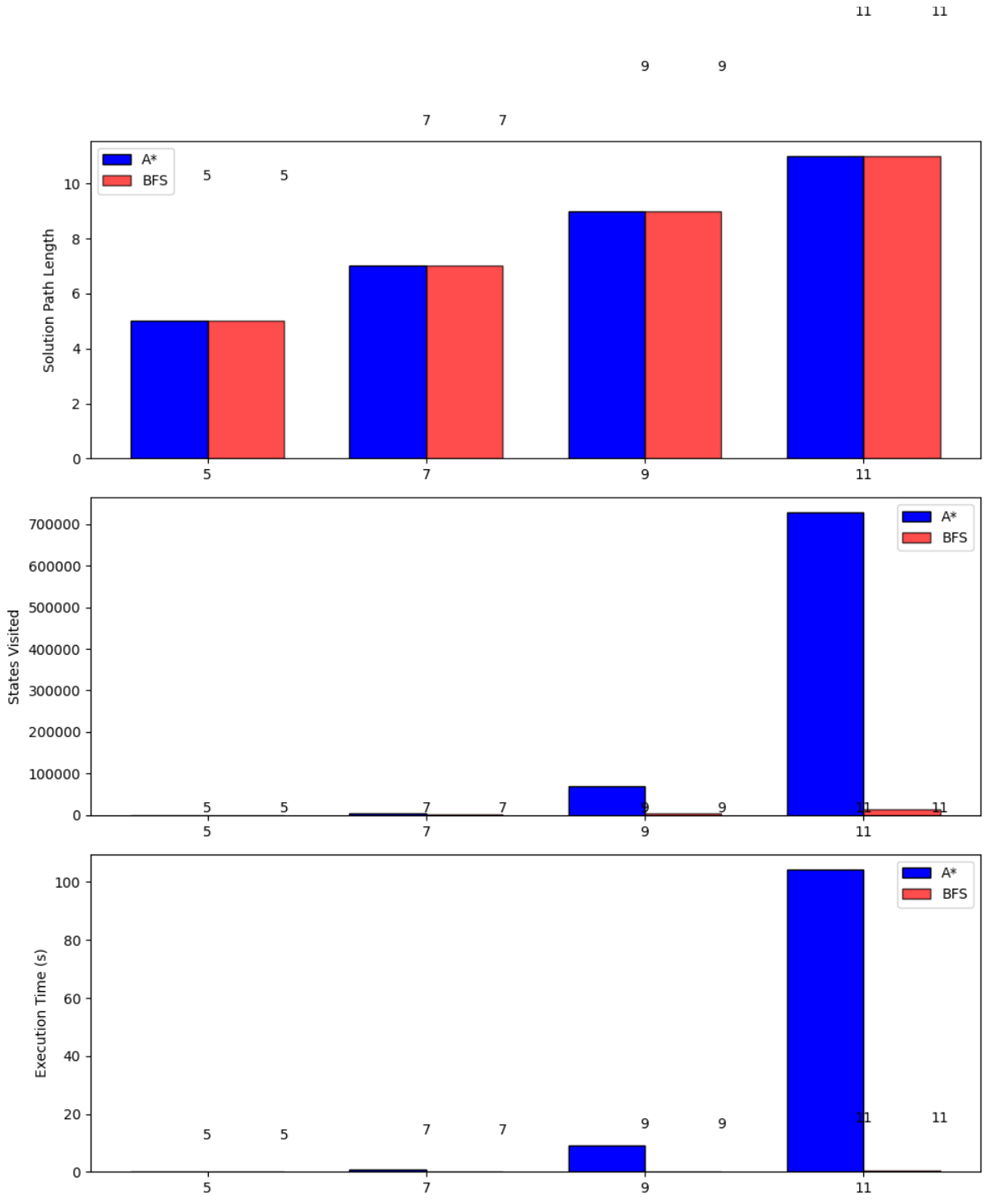
Tema 1 IA - Rezultate și observații

Pantazică Viviana - Ștefania, grupa 341 C3

3.1 Cerința 1: A* și Bidirectional BFS

- Am folosit o euristică h1 admisibilă care calculează numărul de stickere care au culori diferite de starea finală a cubului. Cu cât acest număr este mai mic, cu atât înseamnă că ne apropiem de starea dorită.
  Pentru a garanta că va fi găsită cea mai scurtă cale către soluție, împărțim acest număr la 8; astfel euristica devine și una admisibilă.

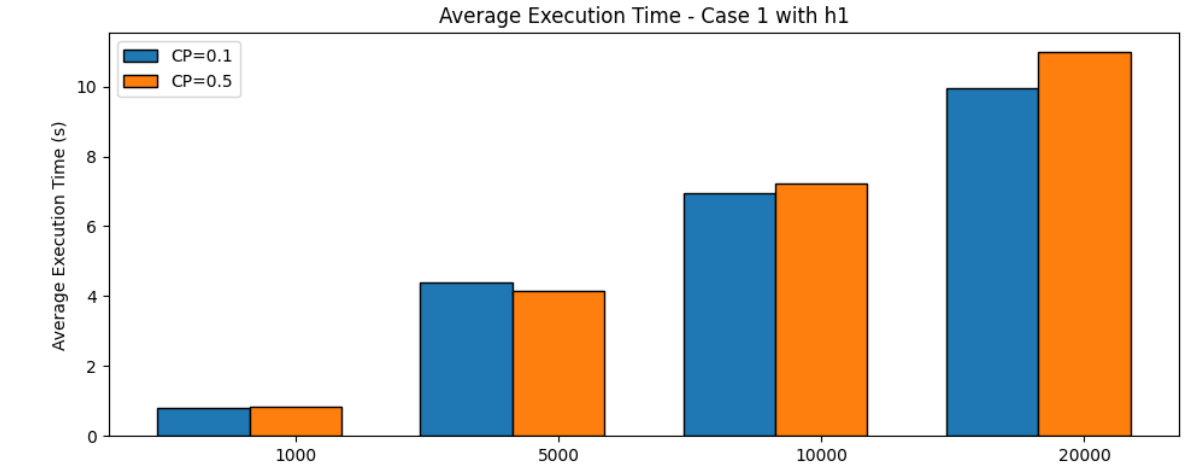
Rezultate A* în comparație cu BFS:

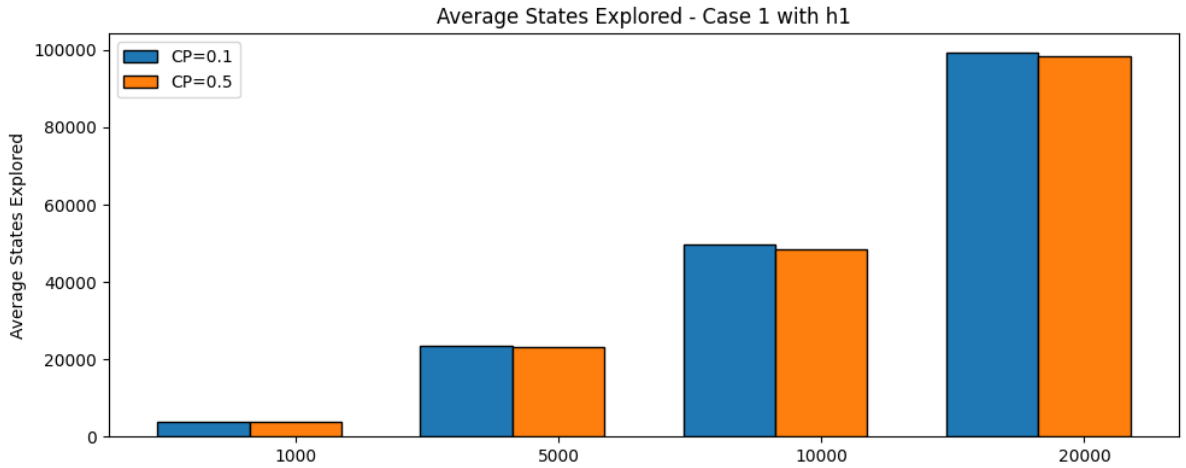


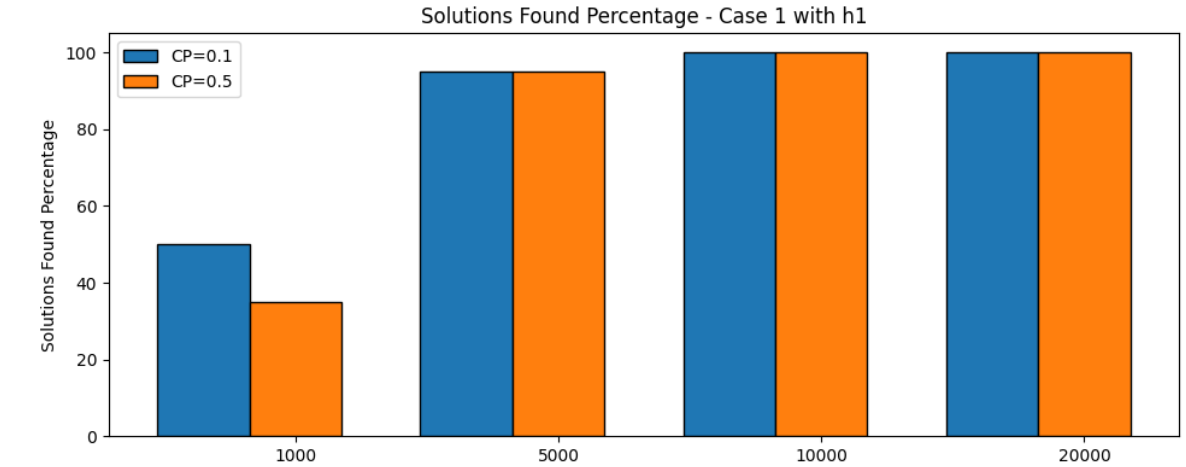
Interpretarea rezultatelor:

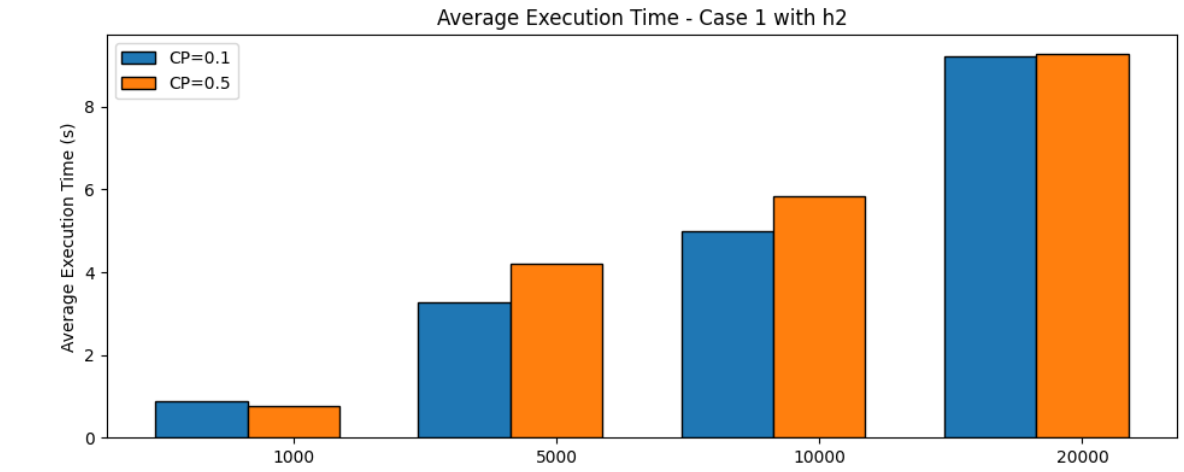
- În cazul A*:
  - solution_path_length (numărul minim de mutări necesare pentru a ajunge la o soluție) crește odată cu mărimea problemei
  - states_visited crește odată cu mărimea problemei, datorită faptului că A* explorează un nr semnificativ de configurații ale cubului, dar este limitat de folosirea euristicii h1 care ghidează căutarea
  - timpul de execuție crește exponențial pentru cazuri mai complexe (case3, case4), din cauza complexității exponențiale a lui A*.
- În cazul BFS:
  - similar cu A*, valorile rezultate cresc odată cu mărimea problemei
  - BFS explorează toate stările posibile la o anumită adâncime înainte de a avansa la adâncimea următoare (de aceea numărul de stări vizitate obținut este mai mare decât în cazul A*)
  - timpul de execuție este influențat de operațiile pe coadă
- Comparație A* vs BFS:
  - A* oferă soluții într-un timp mai scurt decât BFS pentru aceeași dimensiune a problemei
  - A* explorează mai puține stări în comparație cu BFS datorită euristicii, care îi permite să se orienteze către stările cu costuri mai mici
  - Pentru ambii algoritmi, creșterea dimensiunii problemei duce la o creștere exponențială a timpului și a stărilor vizitate

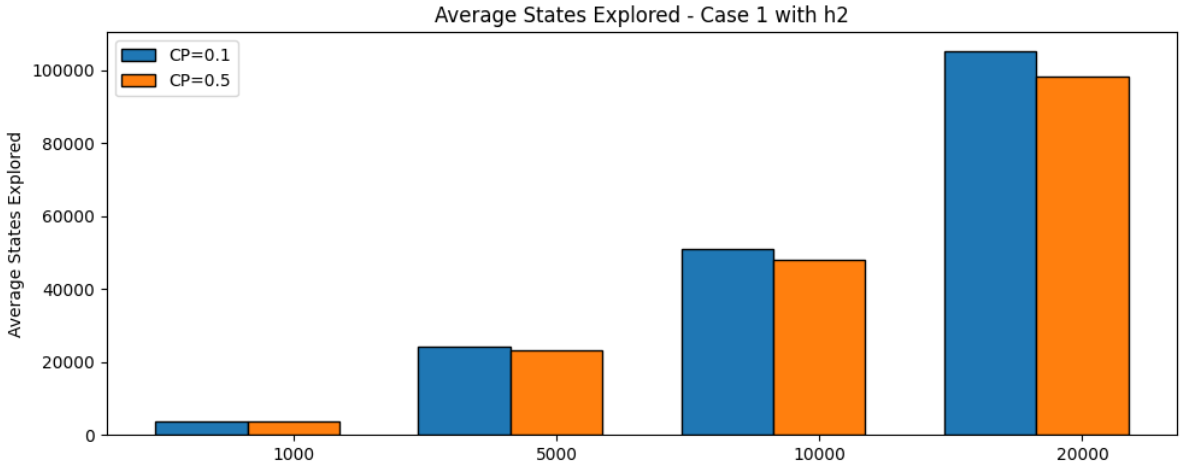
3.2 Cerința 2: Monte Carlo Tree Search

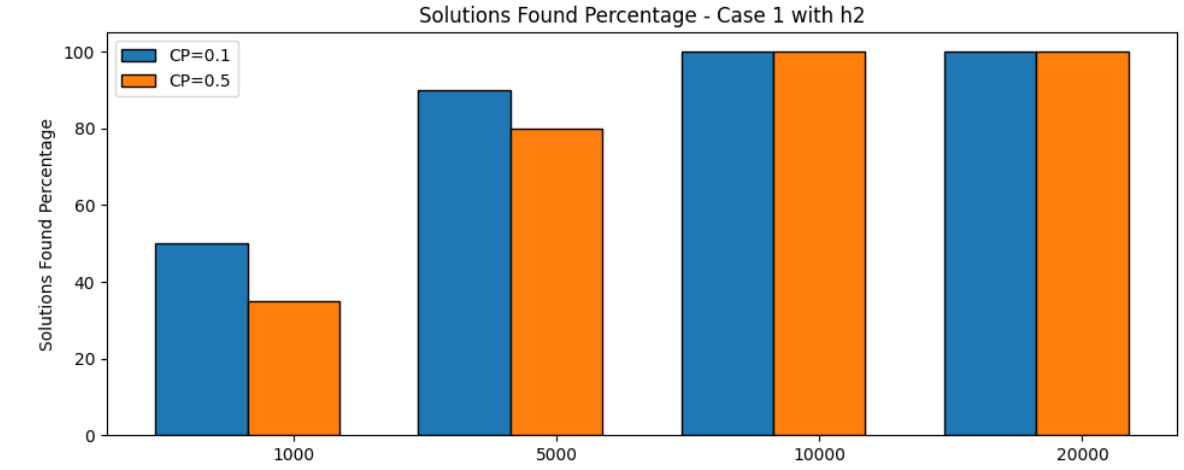


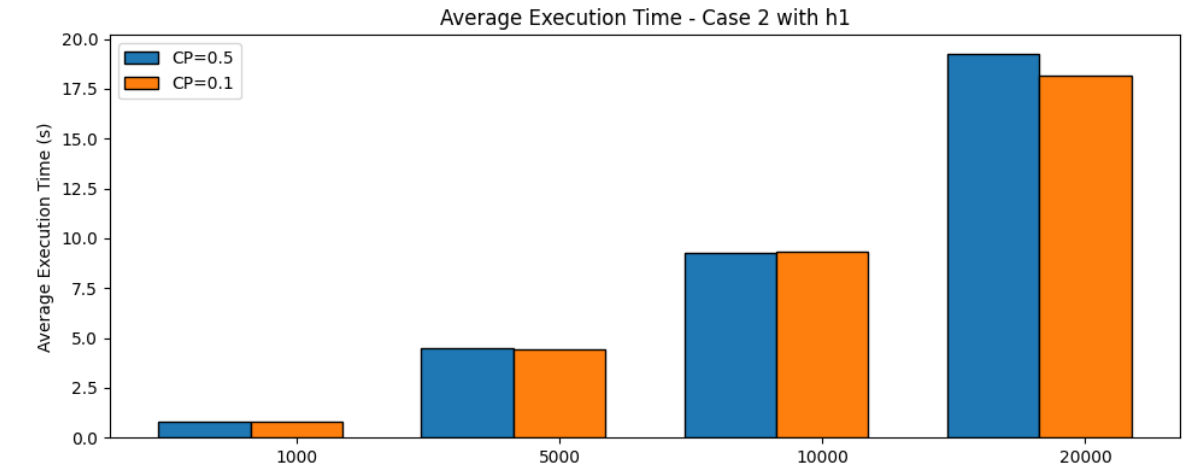


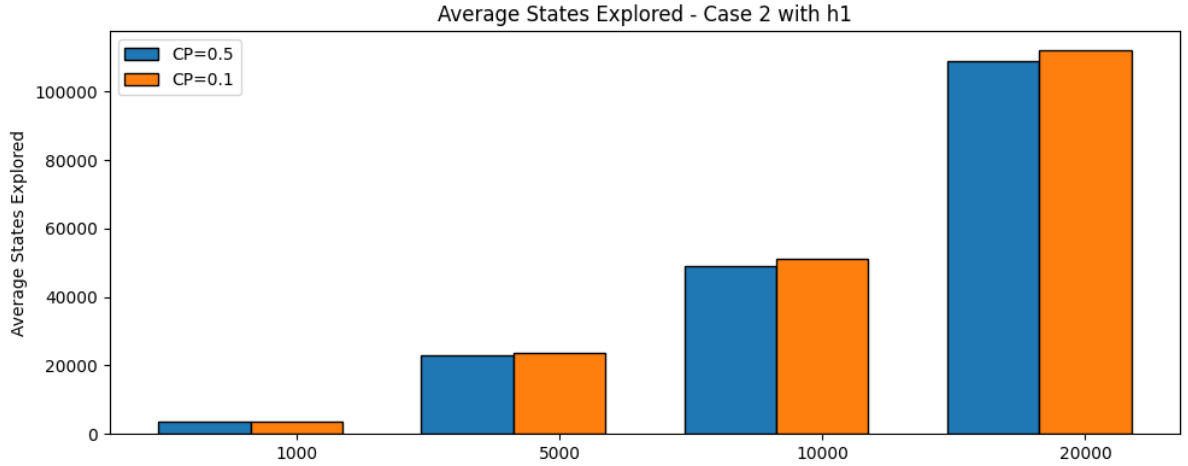


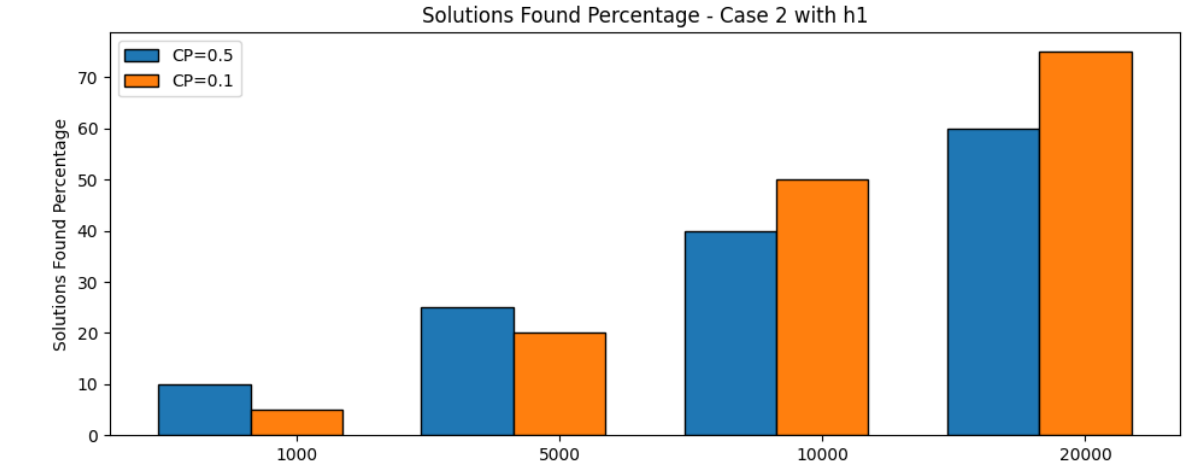

Average Execution Time - Case 1 with h2
CP=0.1
CP=0.5
Average Execution Time (s)
0
2
4
6
8
1000
5000
10000
20000

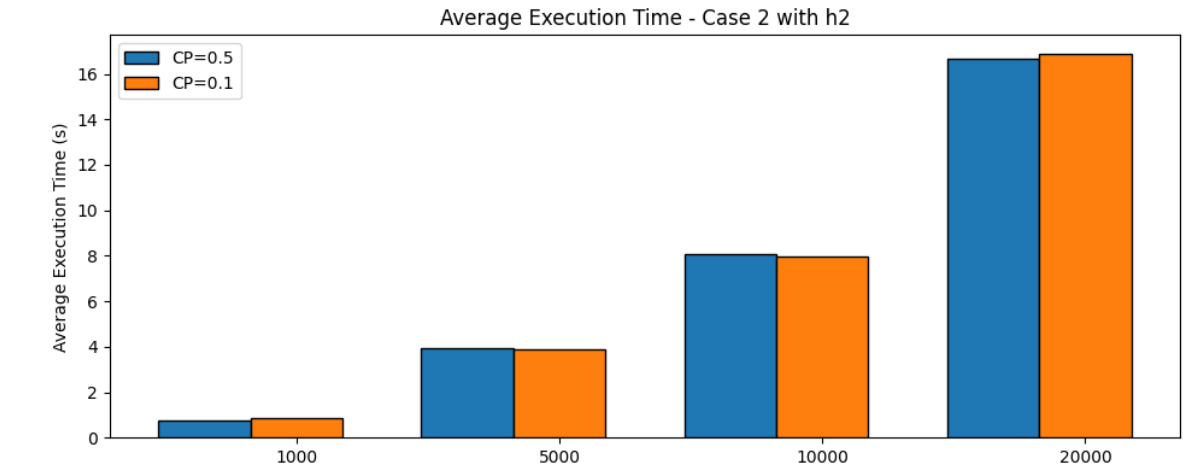

Average States Explored - Case 1 with h2
CP=0.1
CP=0.5
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000

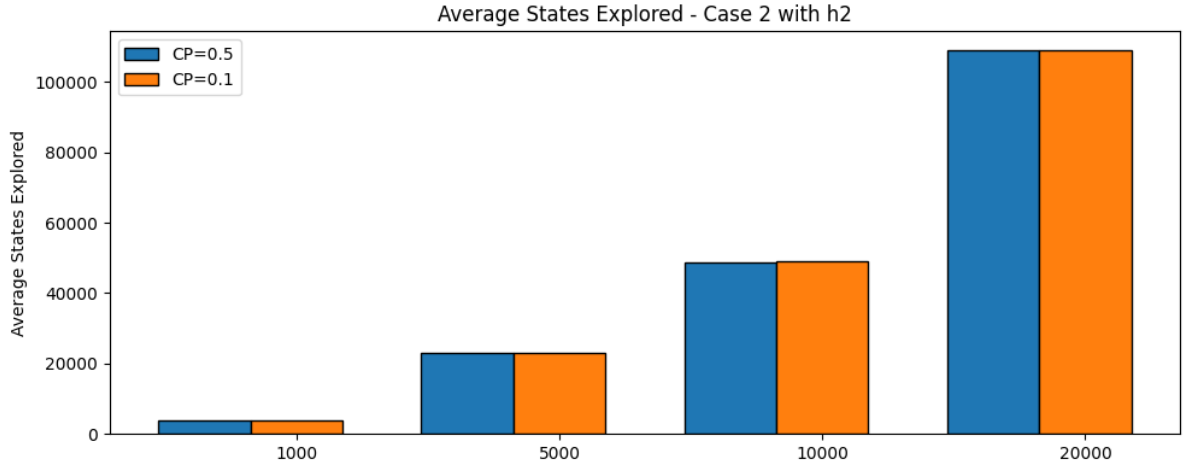

Solutions Found Percentage - Case 1 with h2
CP=0.1
CP=0.5
Solutions Found Percentage
0
20
40
60
80
100
1000
5000
10000
20000

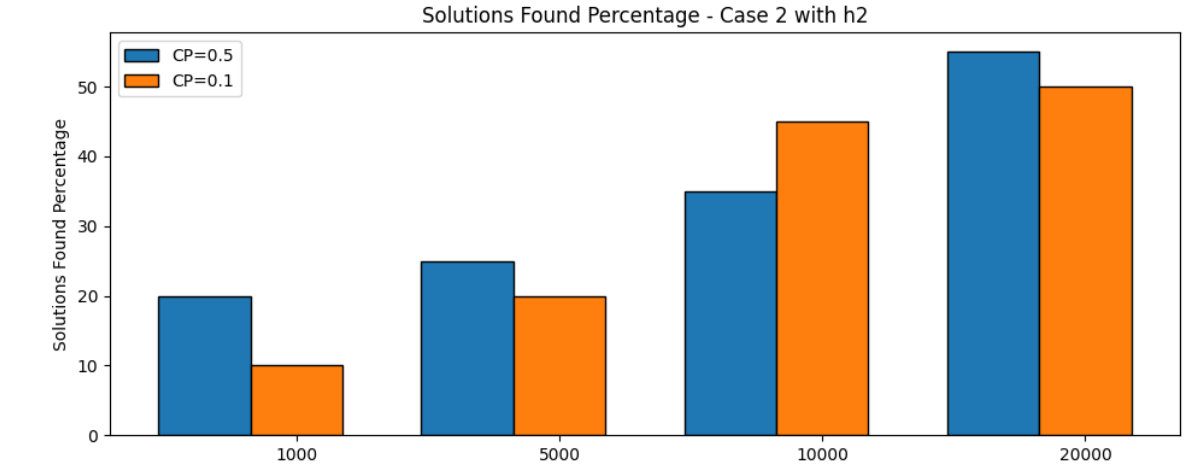

Average Execution Time - Case 2 with h1
CP=0.5
CP=0.1
Average Execution Time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1000
5000
10000
20000

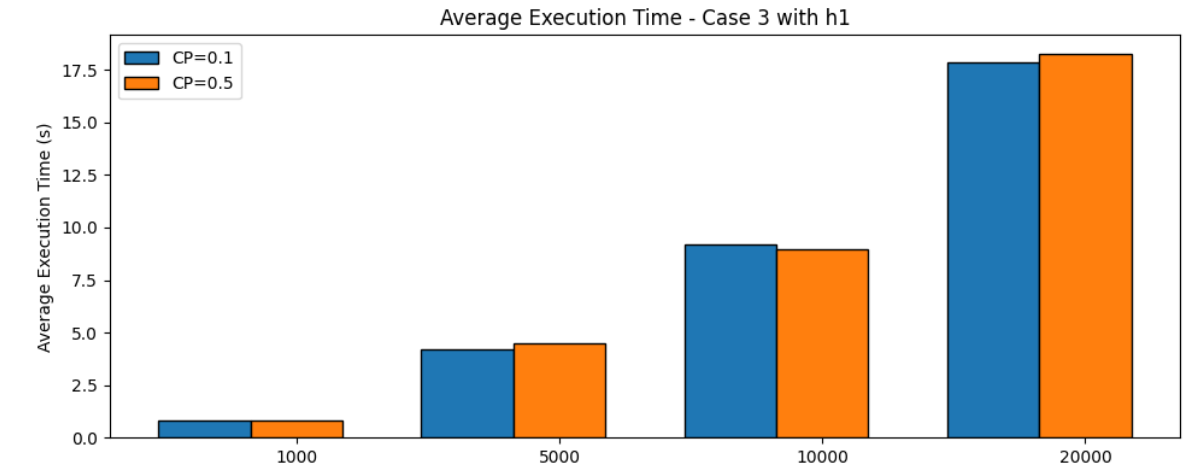

Average States Explored - Case 2 with h1
CP=0.5
CP=0.1
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000

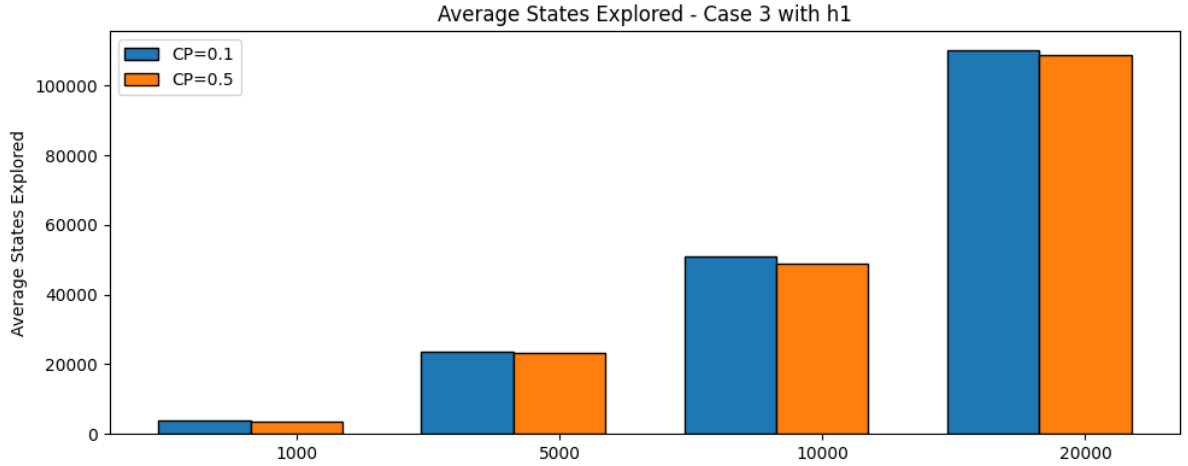

Solutions Found Percentage - Case 2 with h1
CP=0.5
CP=0.1
Solutions Found Percentage
0
10
20
30
40
50
60
70
1000
5000
10000
20000

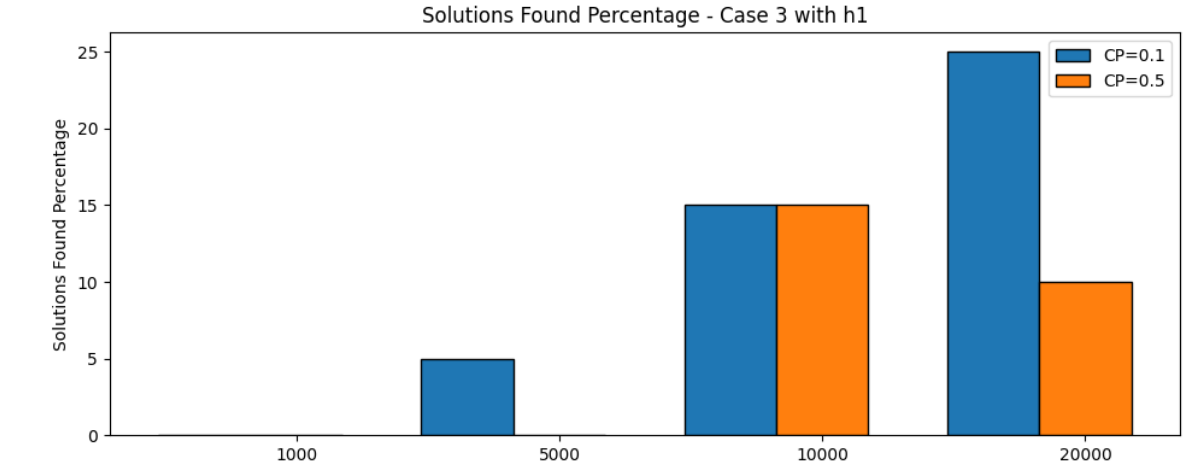

Average Execution Time - Case 2 with h2
CP=0.5
CP=0.1
Average Execution Time (s)
0
2
4
6
8
10
12
14
16
1000
5000
10000
20000

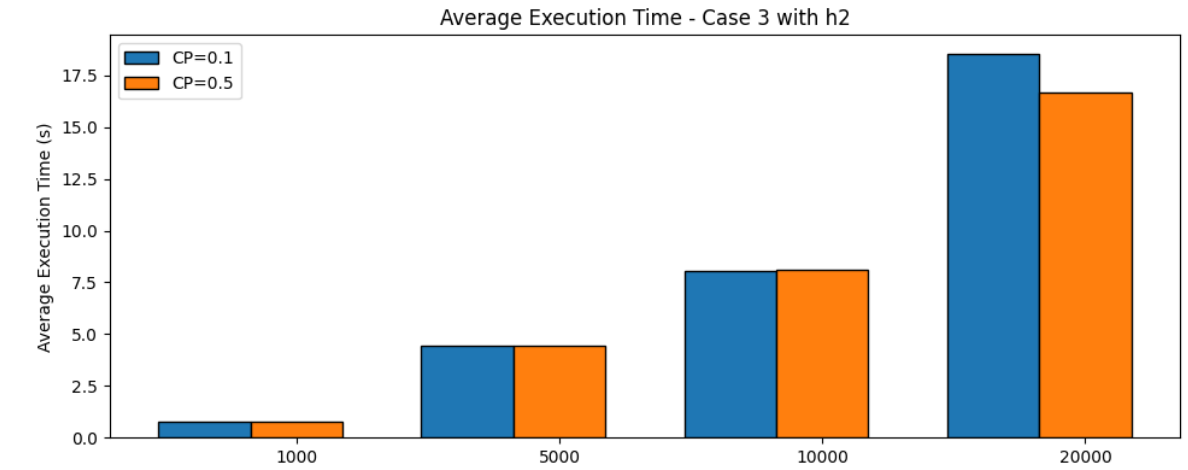

Average States Explored - Case 2 with h2
CP=0.5
CP=0.1
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000

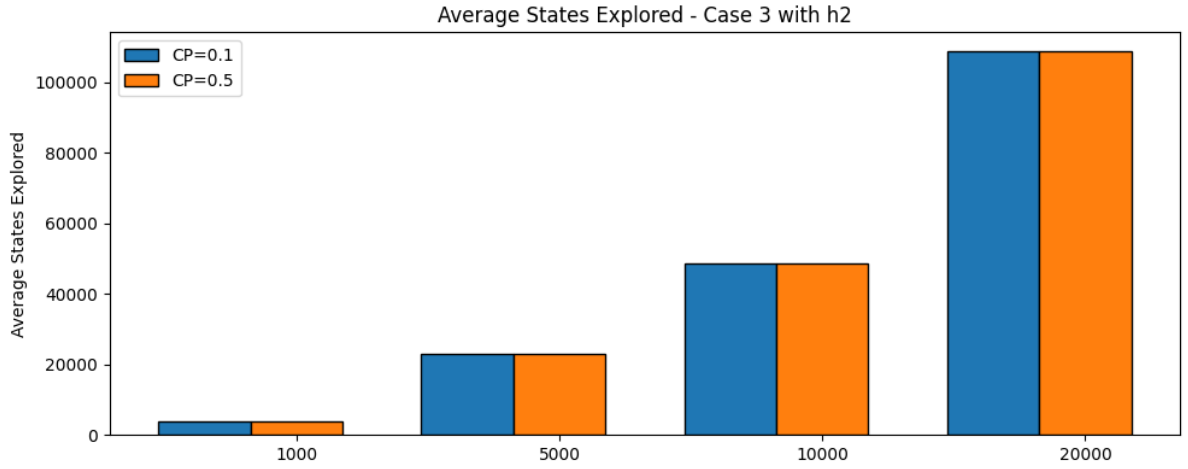

Solutions Found Percentage - Case 2 with h2
CP=0.5
CP=0.1
Solutions Found Percentage
0
10
20
30
40
50
1000
5000
10000
20000

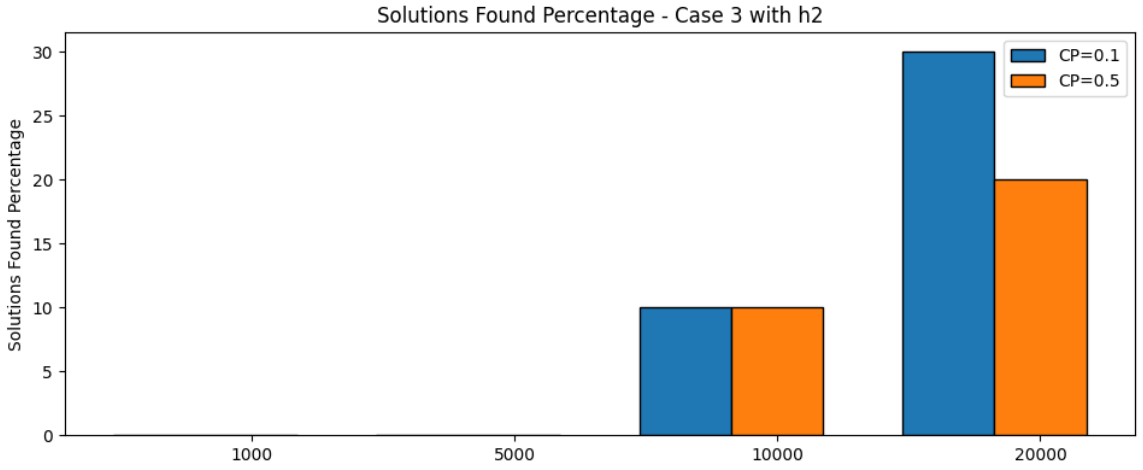

Average Execution Time - Case 3 with h1
CP=0.1
CP=0.5
Average Execution Time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1000
5000
10000
20000

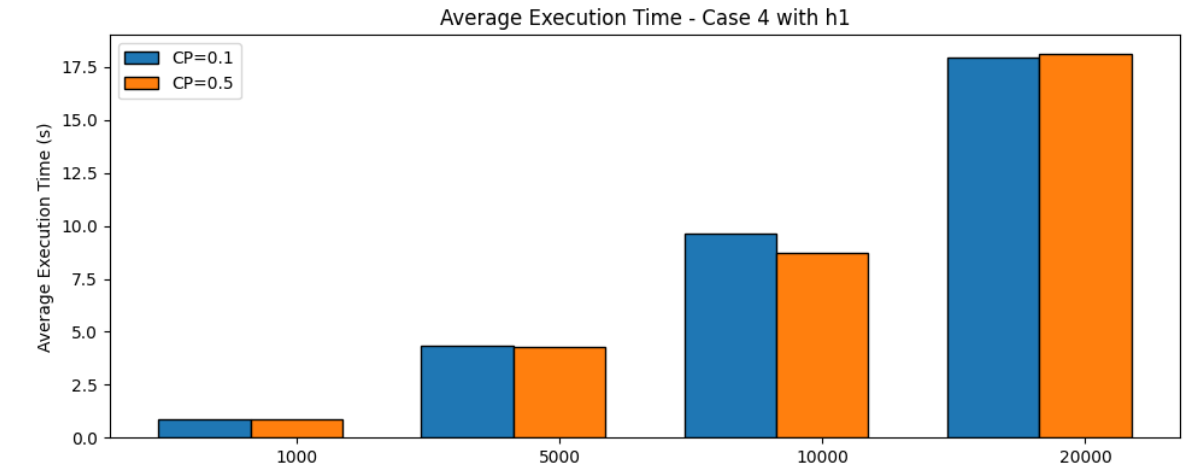

Average States Explored - Case 3 with h1
CP=0.1
CP=0.5
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000

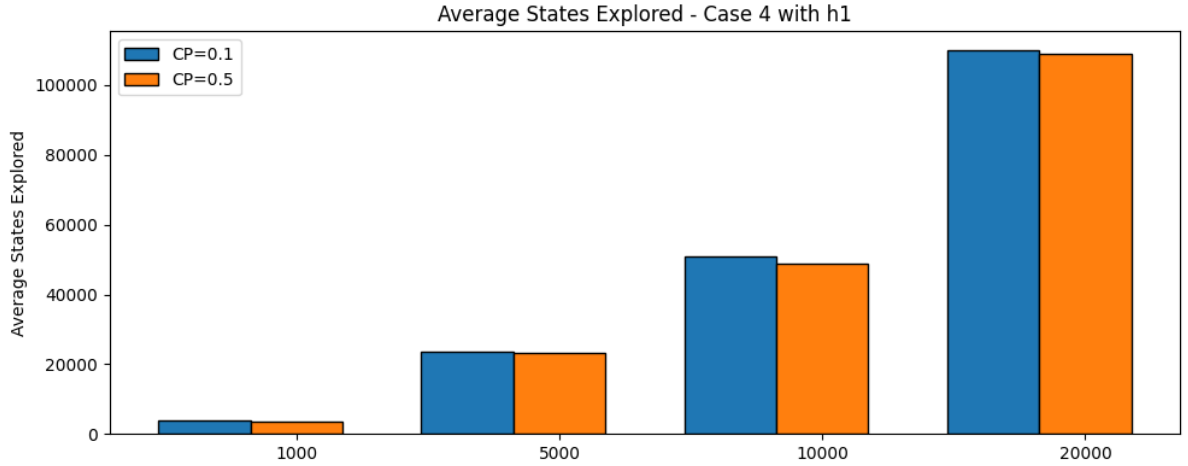

Solutions Found Percentage - Case 3 with h1
CP=0.1
CP=0.5
Solutions Found Percentage
0
5
10
15
20
25
1000
5000
10000
20000

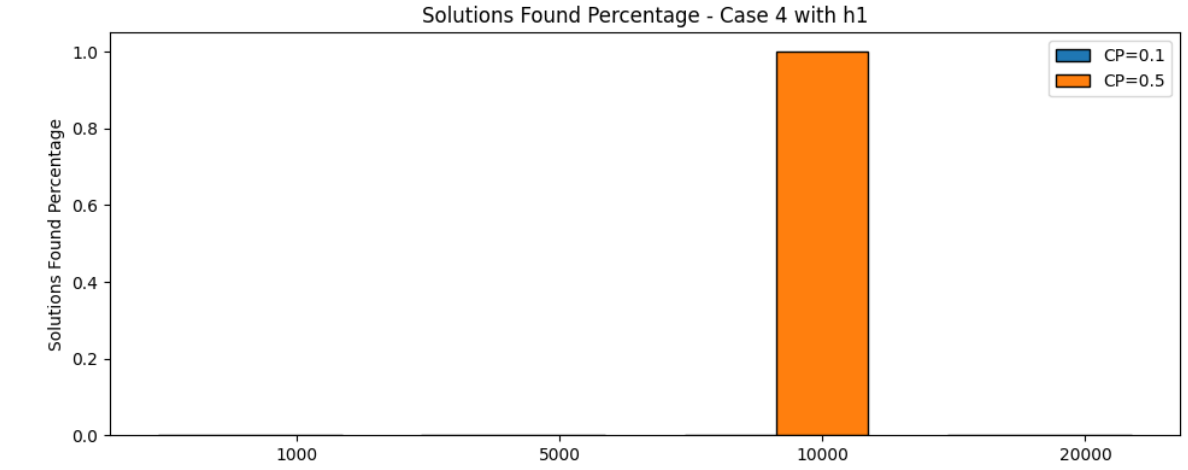

Average Execution Time - Case 3 with h2
CP=0.1
CP=0.5
Average Execution Time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1000
5000
10000
20000

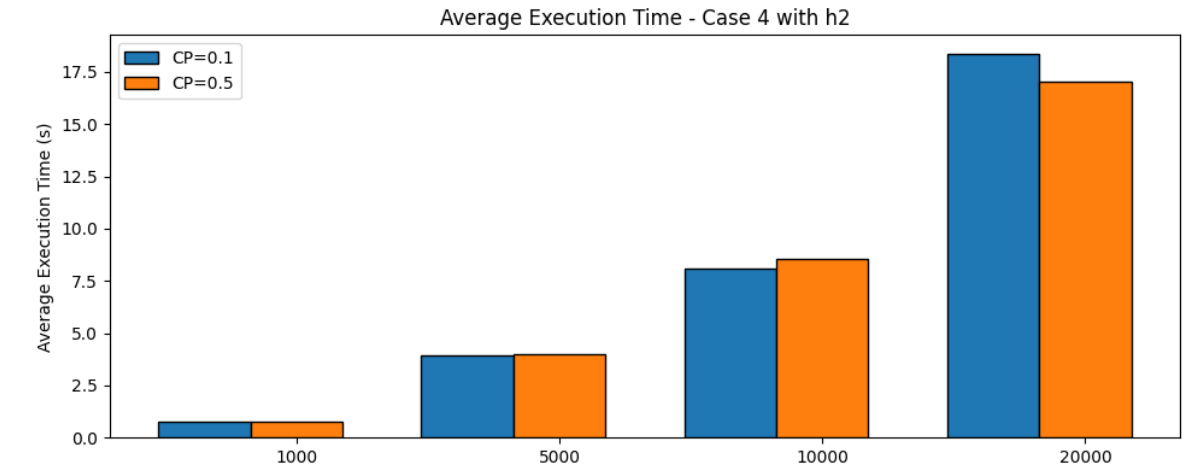

Average States Explored - Case 3 with h2
CP=0.1
CP=0.5
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000

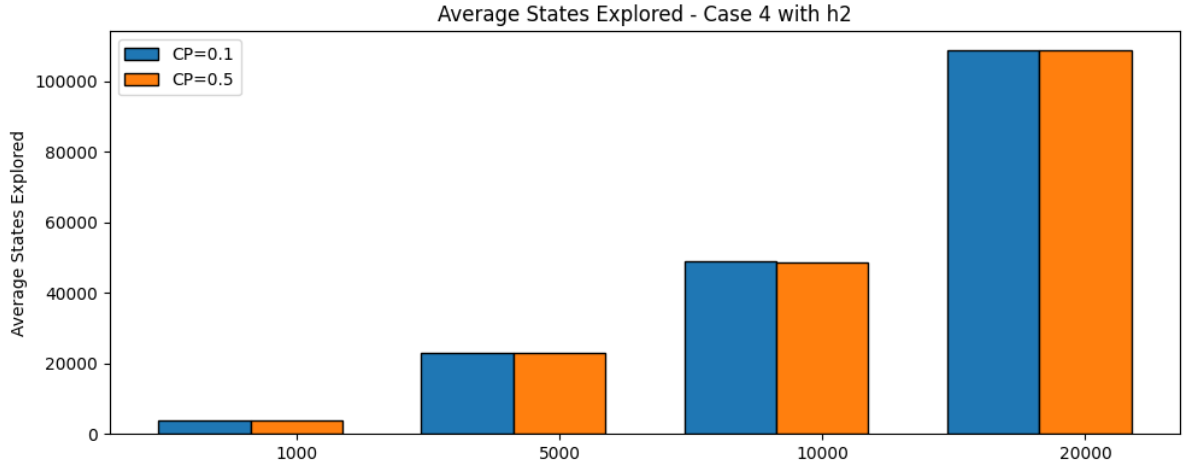

Solutions Found Percentage - Case 3 with h2
CP=0.1
CP=0.5
Solutions Found Percentage
0
5
10
15
20
25
30
1000
5000
10000
20000

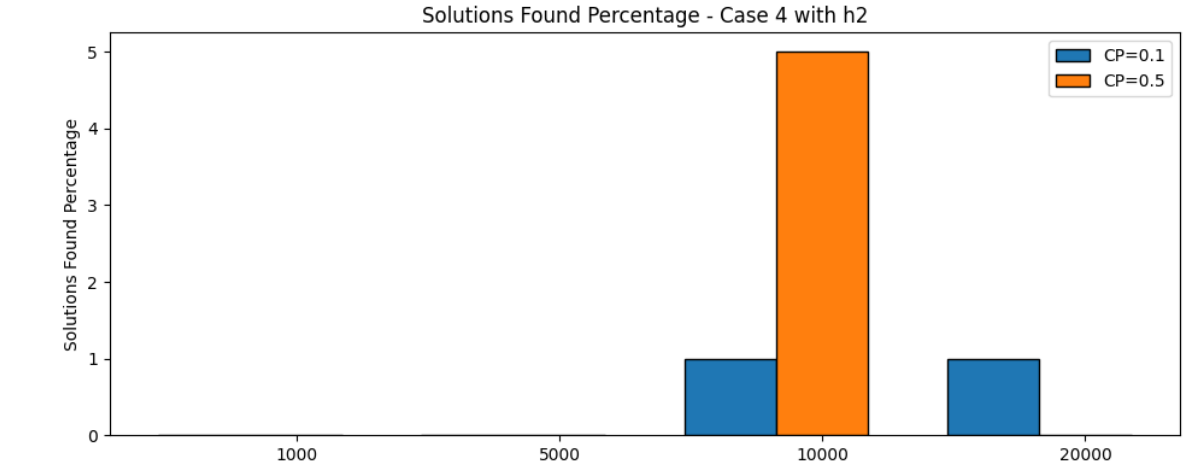

Average Execution Time - Case 4 with h1
CP=0.1
CP=0.5
Average Execution Time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1000
5000
10000
20000


Average States Explored - Case 4 with h1
CP=0.1
CP=0.5
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000


Solutions Found Percentage - Case 4 with h1
CP=0.1
CP=0.5
Solutions Found Percentage
0.0
0.2
0.4
0.6
0.8
1.0
1000
5000
10000
20000


Average Execution Time - Case 4 with h2
CP=0.1
CP=0.5
Average Execution Time (s)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
1000
5000
10000
20000

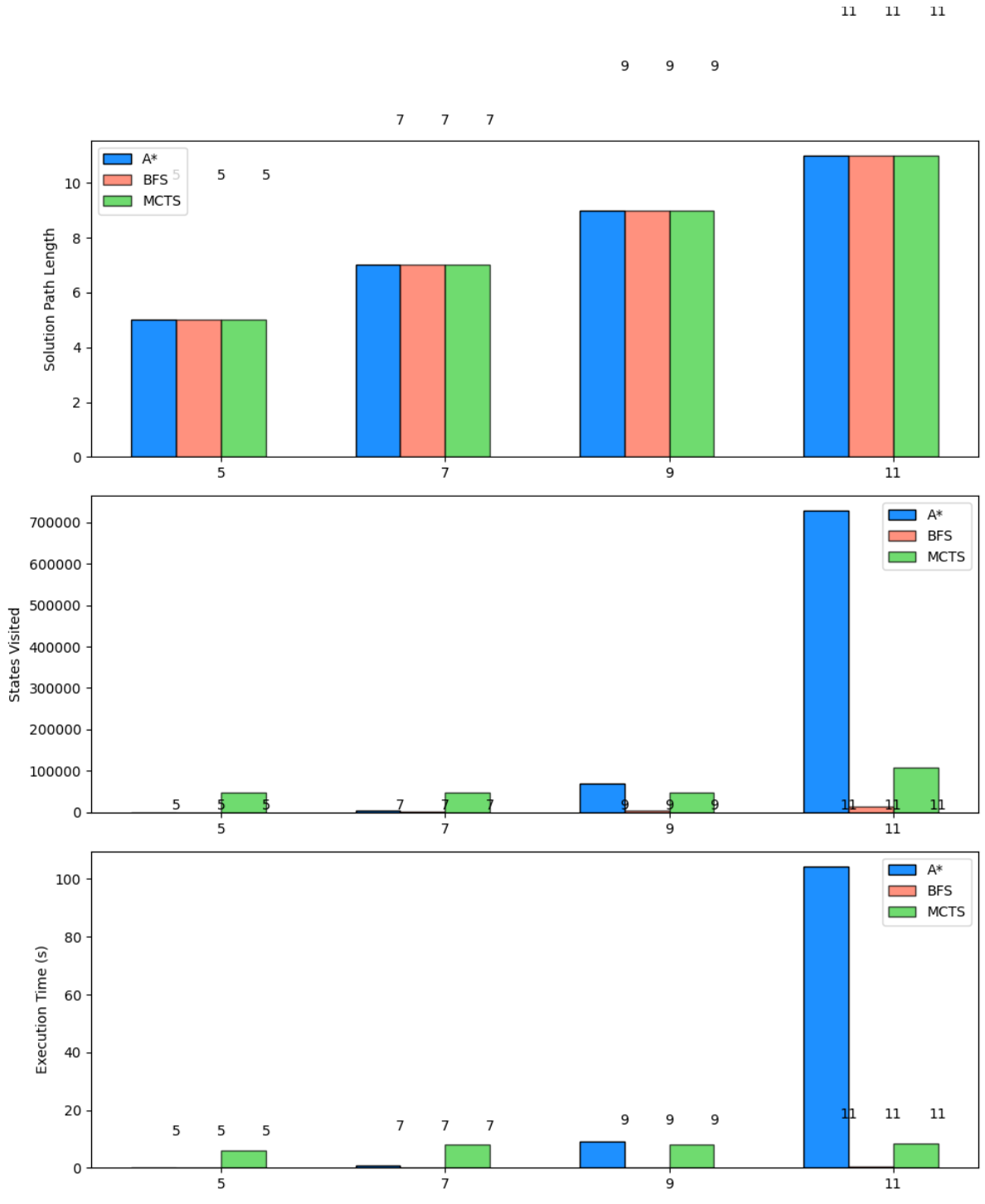

Average States Explored - Case 4 with h2
CP=0.1
CP=0.5
Average States Explored
0
20000
40000
60000
80000
100000
1000
5000
10000
20000


Solutions Found Percentage - Case 4 with h2
CP=0.1
CP=0.5
Solutions Found Percentage
0
1
2
3
4
5
1000
5000
10000
20000

Interpretarea rezultatelor:

- procentajul de soluții găsite crește în general odată cu creșterea bugetului
- timpul de execuție a programului crește odată cu complexitatea cazului
- constanta CP influențează explorarea vs. exploatarea în MCTS. Valori mai mari duc la o explorare mai mare a spațiului de stări, ceea ce poate crește șansele de a găsi soluții noi
- utilizarea unei euristici admisibile (h1) ar trebui să conducă la estimări mai precise, dar nu este garantată găsirea celei mai bune soluții. Euristicile neadmisibile (h2) pot furniza estimări inexacte, dar pot explora un spațiu de stări mai larg
- MCTS eșuează în anumite cazuri să găsească o soluție, procentajul de soluții găsite fiind 0.0.

Pentru compararea celor 3 algoritmi MCTS, A* și BFS am ales varianta de MCTS care a dat rezultate bune în majoritatea cazurilor dpdv al procentajului de soluții găsite: varianta care folosește euristica neadmisibilă (h2), buget 1000 și CP = 0.5. Am obținut:



Concluziile mele ar fi următoarele:

- BFS are un comportament dezirabil în majoritatea cazurilor și este superior celorlalți 2 algoritmi analizați
- A* explorează cele mai multe stări și de aceea și timpul de execuție este mai mare
- MCTS are rezultate destul de consistente în ceea ce privește toți factorii de analiză, dar nu reușește întotdeauna să furnizeze o soluție

3.3 Cerința 3: Pattern database

Am implementat metoda *build_pattern_database(cube, max_lim)* folosind aproape aceeași implementare ca la BFS și ținând cont de cerință (se pornește căutarea de la goal_state și există o limitare egală cu 7). Rezultatul a fost crearea unui catalog care reține o stare și costul corespunzător acesteia (costul a fost calculat ca distanță între starea finală și cea curentă; deci costul e cu atât mai mic cu cât ne apropiem de soluție).

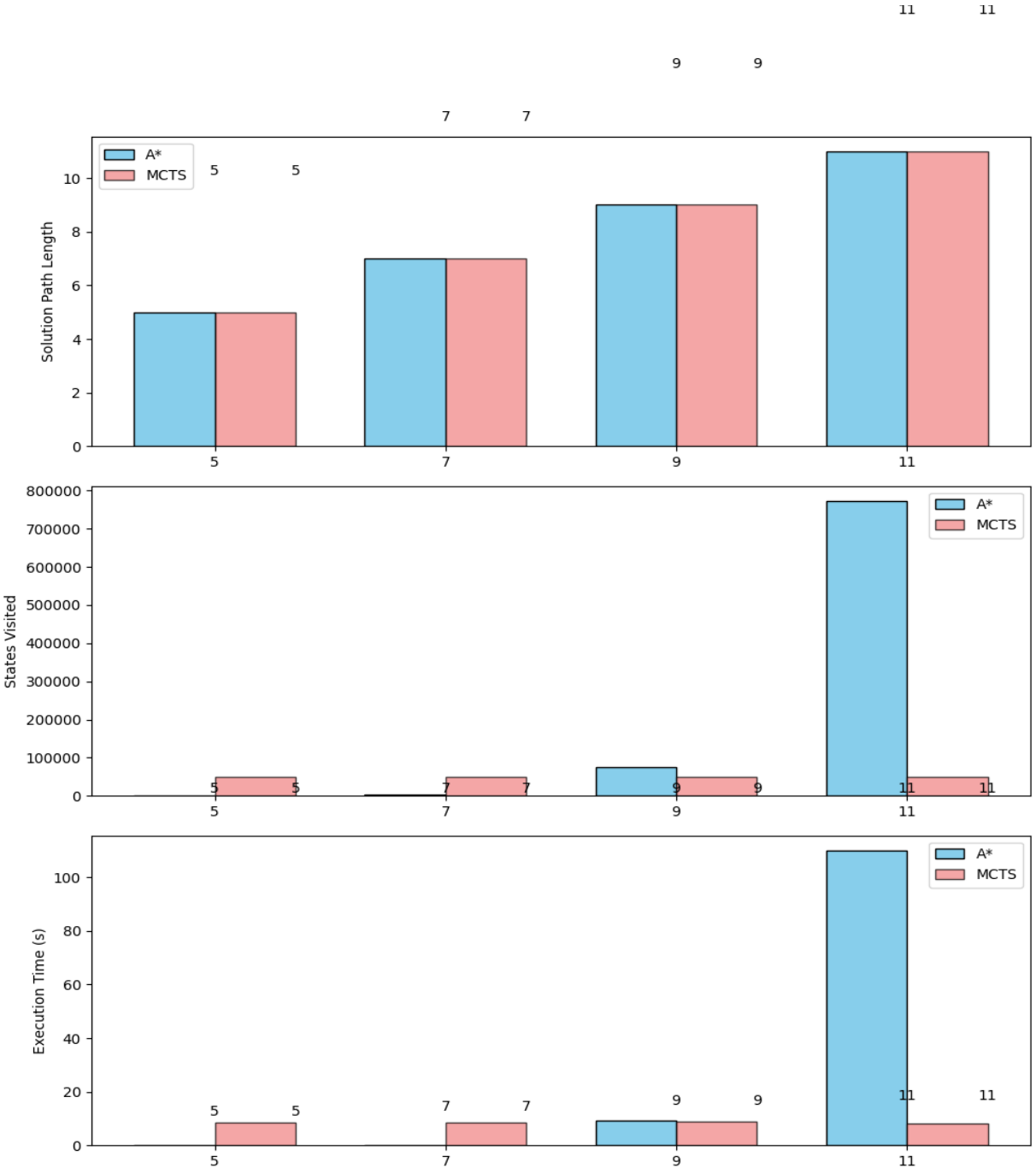
Euristica h3 iterează prin pattern_database și extrage costul reținut în acest catalog pentru starea dată ca parametru. În cazul în care starea nu a fost înregistrată în catalog, se va folosi valoarea rezultată de euristica h1.

Am rescris funcțiile A* cu pattern_database, respectiv MCTS, pentru a folosi euristica h3. Am ales pentru MCTS cazul în care algoritmul a furnizat o soluție de cele mai multe ori: buget 1000, CP=0.5. Rezultatele sunt următoarele:

- A* poate fi mai eficient în găsirea unor soluții mai scurte în spațiul de stări al cubului Pocket
- MCTS pare să viziteze un număr mai mic de stări în comparație cu A* pentru toate nivelurile de complexitate
- MCTS poate fi mai eficient în explorarea spațiului de stări, vizitând un număr mai mic de configurații
- A* are un timp de execuție mai mare în comparație cu MCTS, în special pentru nivelurile de complexitate mai mari (case3, case4)
- MCTS poate oferi soluții într-un timp mai scurt, dar poate să nu ofere întotdeauna soluții.

Pattern database a îmbunătățit pertotal performanțele algoritmilor, deoarece:

- Pattern database stochează informații precalculate despre configurațiile stărilor în apropierea goal state, deci se reduce spațiul de căutare
- Automat, se reduce și timpul de execuție.


A*
MCTS
Solution Path Length
States Visited
Execution Time (s)
5
7
9
11