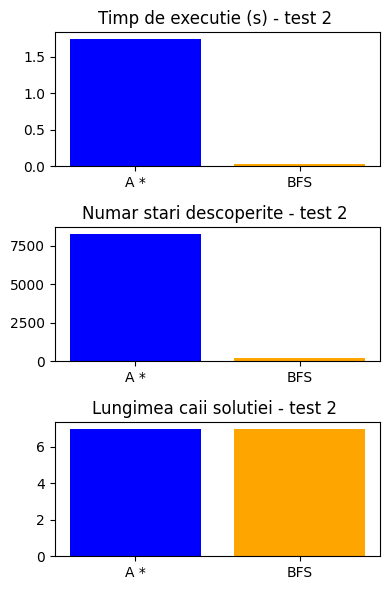
**Tema1 – Pocket Cube**

*Avramescu Cosmin-Alexandru – 344C3*

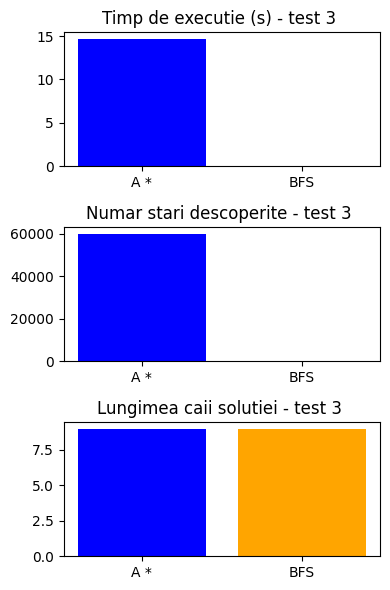
1. **A \* si BFS bidirectional**

**Test 1 vs Test2 vs Test3 vs Test 4**

****O imagine care conține text, captură de ecran, diagramă, linie

Descriere generată automat

**O imagine care conține text, captură de ecran, diagramă, număr

Descriere generată automat**

**Test 1**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** |
| **A \*** | 0.14 | 646 | 5 |
| **BFS bidirectional** | 0.02 | 110 | 6 |

**Test 2**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** |
| **A \*** | 1.78 | 8281 | 7 |
| **BFS bidirectional** | 0.02 | 181 | 7 |

**Test 3**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** |
| **A \*** | 14.68 | 59968 | 9 |
| **BFS bidirectional** | 0.006 | 58 | 9 |

**Test 4**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** |
| **A \*** | 215.52 | 808230 | 11 |
| **BFS bidirectional** | 0.01 | 94 | 9 |

Se poate observa ca pe cel mai simplu caz (test 1), timpul de executie este asemanator, insa pe masura ce crestem gradul de amestecare al cubului rubik, bfs-ul se descurca exponential mai bine deoarece are timpi de executie foarte buni (este rapid) si nici nu expandeaza atat de multe stari precum A \*. Cu toate acestea, singurul dezavantaj al BFS este ca nu gaseste mereu solutia de lungime cea mai scurta. A \*, datorita euristicii admisibile, gaseste mereu solutia cea mai scurta. Discutam de un trade-off si consider ca viteza mult mai mare a BFS este mai importanta, asa ca il consider pe acesta un algoritm mai eficient per total.

1. **Monte Carlo Tree Search**

**Test 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

**Test 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

**Test 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

**Test 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

Acest algoritm are cele mai slabe rezultate din cauza alegerilor aleatorii pe care le face. Este nevoie de bugete foarte mari pentru a reusi sa se gaseasca solutie si chiar si asa cu bugete mari, nu se gasesc solutii pentru cuburile amestecate mai puternic.

1. **Pattern database**

**Test 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

**Test 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

**Test 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

**Test 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Timp executie (s)** | **Numar stari descoperite** | **Lungime cale pana la solutie** | **Constanta C** | **Buget** |
| **1** |  |  |  |  | **1000** |
| **2** |  |  |  |  | **5000** |
| **3** |  |  |  |  | **10000** |
| **4** |  |  |  |  | **20000** |

Se poate observa ca performantele MCTS si A \* se imbunatasesc considerabil datorita pattern database. Din moment ce retinem costul real intr-un catalog pentru toate starile pana la adancime 7 fata de solutie, obtinum un avantaj foarte mare. Avand in vedere ca un cub rubik 2x2x2 se poate rezolva in cel mult 14 mutari, asta inseamna ca noi retinem in dictionar jumatate din stari pentru care estimam exact costul. Prin urmare, A\* si MCTS ruleaza mult mai repede deoarece converg mai repede la solutie datorita estimarii bune a costului.