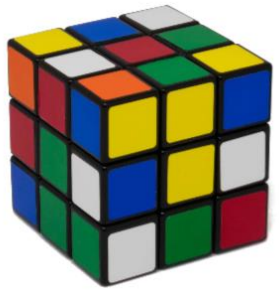
Planning And Reasoning

Project presentation – Rubik's Cube

Simone Scaccia

SAPIENZA
UNIVERSITÀ DI ROMA

Rubik's Cube
Problem definition



- 8 corners, 12 edges
- Number of feasible states

$$8! \times 3^7 \times \frac{12!}{2} \times 2^{11} = 43{,}252{,}003{,}274{,}489{,}856{,}000$$

- Number of goal states: 1
- Median optimal solution length: 18 moves
- Branching factor:
  - 18 with only 90 degree moves
  - 13 with 90, 180, 270 degree moves

Depth	Nodes
1	18
2	243
3	3,240
4	43,254
5	577,368
6	7,706,988
7	102,876,480
8	1,373,243,544
9	18,330,699,168
10	244,686,773,808
11	3,266,193,870,720
12	43,598,688,377,184
13	581,975,750,199,168
14	7,768,485,393,179,328
15	103,697,388,221,736,960
16	1,384,201,395,738,071,424
17	18,476,969,736,848,122,368
18	246,639,261,965,462,754,048

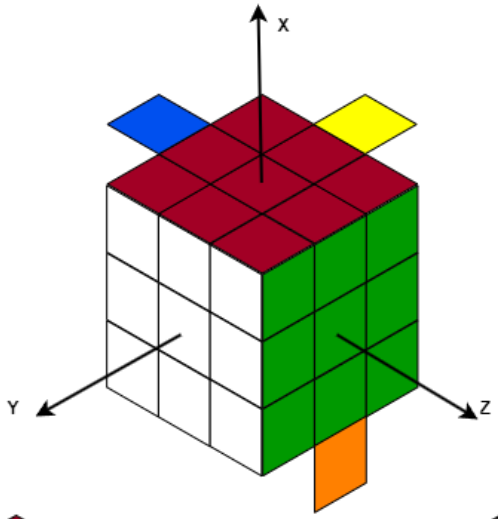
Table 1: Nodes in search tree as a function of depth

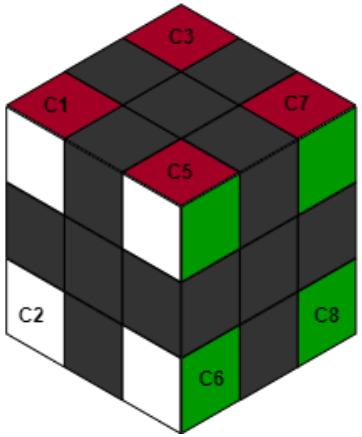
References: https://cdn.aaai.org/AAAI/1997/AAAI97-109.pdf

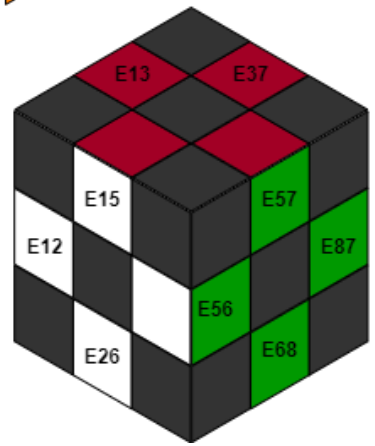
Rubik's Cube

PDDL domain definition: predicates

```
(:predicates
    (cube1 ?x ?y ?z)
    (cube2 ?x ?y ?z)
    (cube3 ?x ?y ?z)
    (cube4 ?x ?y ?z)
    (cube5 ?x ?y ?z)
    (cube6 ?x ?y ?z)
    (cube7 ?x ?y ?z)
    (cube8 ?x ?y ?z)
    (edge12 ?y ?z)
    (edge13 ?x ?z)
    (edge15 ?x ?y)
    (edge26 ?x ?y)
    (edge24 ?x ?z)
    (edge48 ?x ?y)
    (edge34 ?y ?z)
    (edge37 ?x ?y)
    (edge56 ?y ?z)
    (edge57 ?x ?z)
    (edge68 ?x ?z)
    (edge78 ?y ?z)
)
```

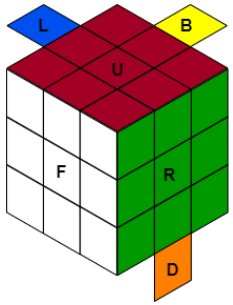






Rubik's Cube

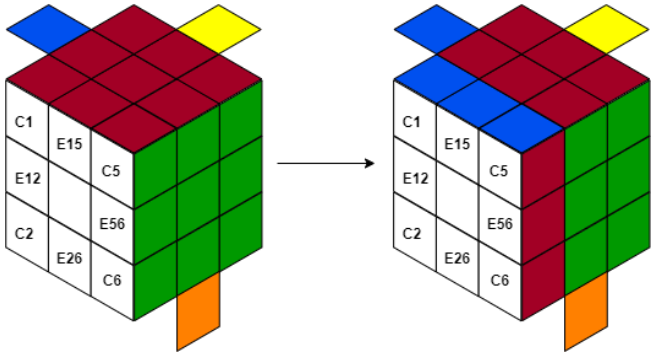
PDDL domain definition: actions



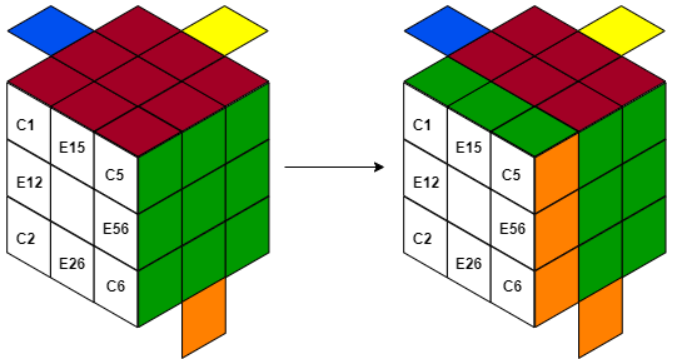

```
(:action F
 :parameters ()
 :precondition (and)
 :effect
    (and
        (forall (?x ?y ?z) (when (cube1 ?x ?y ?z) (and (not (cube1 ?x ?y ?z)) (cube5 ?z ?y ?x) )))
        (forall (?x ?y ?z) (when (cube5 ?x ?y ?z) (and (not (cube5 ?x ?y ?z)) (cube6 ?z ?y ?x) )))
        (forall (?x ?y ?z) (when (cube6 ?x ?y ?z) (and (not (cube6 ?x ?y ?z)) (cube2 ?z ?y ?x) )))
        (forall (?x ?y ?z) (when (cube2 ?x ?y ?z) (and (not (cube2 ?x ?y ?z)) (cube1 ?z ?y ?x) )))

        (forall (?x ?y) (when (edge15 ?x ?y) (and (not (edge15 ?x ?y)) (edge56 ?y ?x))))
        (forall (?y ?z) (when (edge56 ?y ?z) (and (not (edge56 ?y ?z)) (edge26 ?z ?y))))
        (forall (?x ?y) (when (edge26 ?x ?y) (and (not (edge26 ?x ?y)) (edge12 ?y ?x))))
        (forall (?y ?z) (when (edge12 ?y ?z) (and (not (edge12 ?y ?z)) (edge15 ?z ?y))))
    )
)
```

- F action: clock-wise 90° rotation



- Frev action: counter clock-wise 90° rotation



Rubik's Cube

PDDL problem definition

```
(:objects yellow white blue green orange red)
(:init
    (cube1 orange yellow blue)
    (cube2 orange yellow green)
    (cube3 orange white blue)
    (cube4 red yellow green)
    (cube5 yellow blue red)
    (cube6 white green orange)
    (cube7 red green white)
    (cube8 red blue white)
    (edge12 red blue)
    (edge24 blue yellow)
    (edge34 red white)
    (edge13 yellow red)
    (edge15 green red)
    (edge26 orange yellow)
    (edge48 white orange)
    (edge37 white green)
    (edge56 orange blue)
    (edge68 yellow green)
    (edge78 blue white)
    (edge57 green orange)
)
```

- Initial random state: start with an unshuffled Rubik's Cube and apply a sequence of random moves.

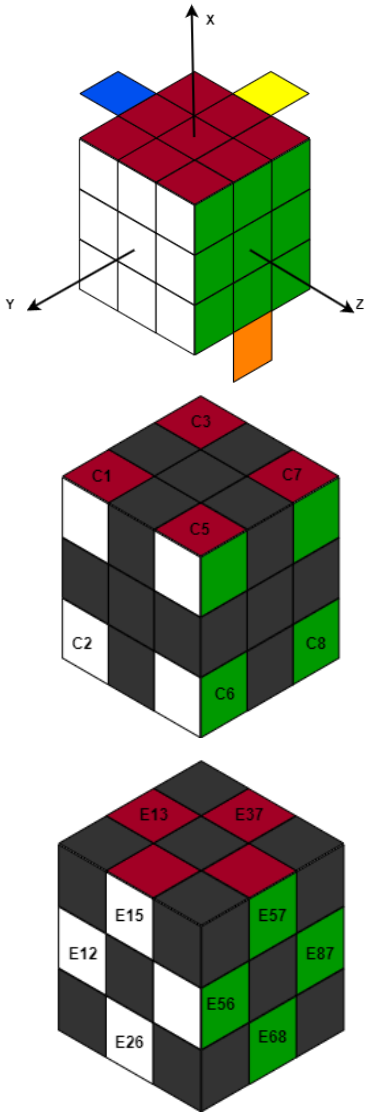
```
(:goal
    (and
        (cube1 red white blue)
        (cube2 orange white blue)
        (cube3 red yellow blue)
        (cube4 orange yellow blue)
        (cube5 red white green)
        (cube6 orange white green)
        (cube7 red yellow green)
        (cube8 orange yellow green)

        (edge12 white blue)
        (edge24 orange blue)
        (edge34 yellow blue)
        (edge13 red blue)

        (edge15 red white)
        (edge26 orange white)
        (edge48 orange yellow)
        (edge37 red yellow)

        (edge56 white green)
        (edge68 orange green)
        (edge78 yellow green)
        (edge57 red green)

    )
)
```



Experiments
Search algorithm

- A*:
  - Worst case space complexity: $1 + b^* + (b^*)^2 + \cdots + (b^*)^d$ nodes generated.
  - b: branching factor
  - d: shortest path. We don't know d in our problem.
- IDA*:
  - Space complexity: $O(\ell b)$
    - b: branching factor
    - l: longest generated path
  - Properties:
    - **Semi-complete** if the heuristic is safe and cost(a) > 0 for all actions
    - **Optimal** if the heuristic is admissible
  - Fast-Downward:
    - https://www.fast-downward.org/Doc/SearchAlgorithm#Iterated_search
    - https://www.fast-downward.org/Doc/SearchAlgorithm#A.2A_search_.28eager.29

Experiments

Fast-Downward heuristics

- Additive heuristic:
  - admissible: no
  - consistent: no
  - safe: yes for tasks without axioms

  Then IDA* is semi-complete and not optimal
- Max heuristic:
  - admissible: yes for tasks without axioms
  - consistent: yes for tasks without axioms
  - safe: yes for tasks without axioms

  Then IDA* is semi-complete and optimal
- FF heuristic:
  - admissible: no
  - consistent: no
  - safe: yes for tasks without axioms

  Then IDA* is semi-complete and not optimal

References: https://www.fast-downward.org/Doc/Evaluator

Experiments

IPC planner: Scorpion 2023

- IPC 2023 optimal track

Coverage	folding	labyrinth	quantum.	recharg.	ricochet.	rubiks	slither.	SUM
ragnarok	8	**8**	13	**14**	**17**	10	**7**	77
scorpion-2023	8	5	**14**	**14**	**17**	10	6	74
odin	8	5	13	**14**	**17**	10	6	73

- Based on Fast-Downward
- GitHub: https://github.com/ipc2023-classical/planner25

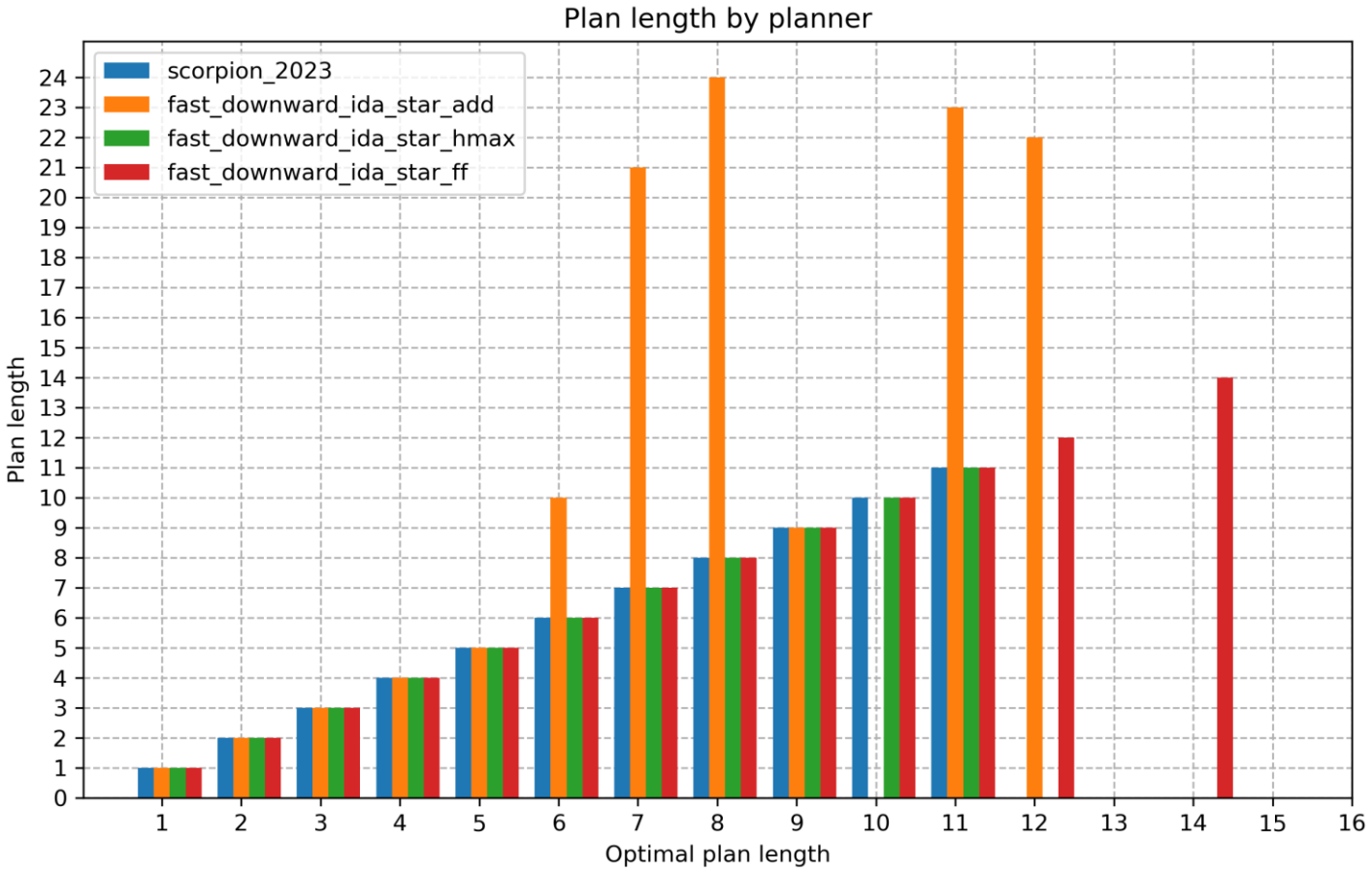
Experiments

Comparison between planners

- Generate problems with increasing optimal solution length
- Run the planners with the following constraints:
  - Time limit: 1 hour
  - Space limit: 13 GB
- Evaluate the performances by comparing the following metrics:
  - Plan cost, if a plan is found
  - Execution time
  - Generated states

Results

Plan length



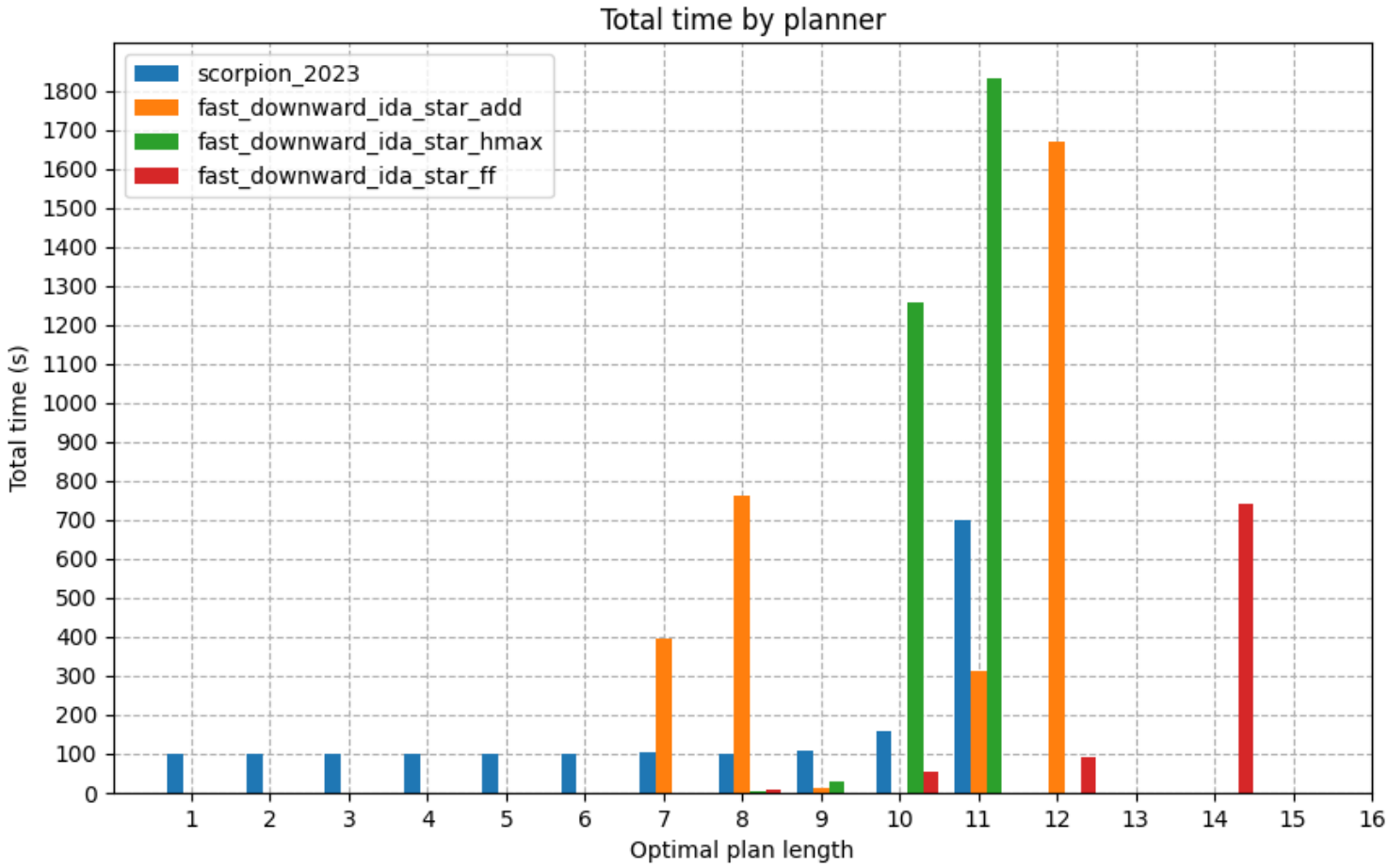
Results

Plan length: comments

- An empty bar indicates that no solution was found due to either a time or space constraint failure, resulting in the planner being aborted.
- It is expected that the additive heuristic may not always find the optimal plan.
- Surprisingly, all the plans found by the FF heuristic are optimal, also on problems in which the two optimal planners don’t follow the constraints.

Results

Execution time



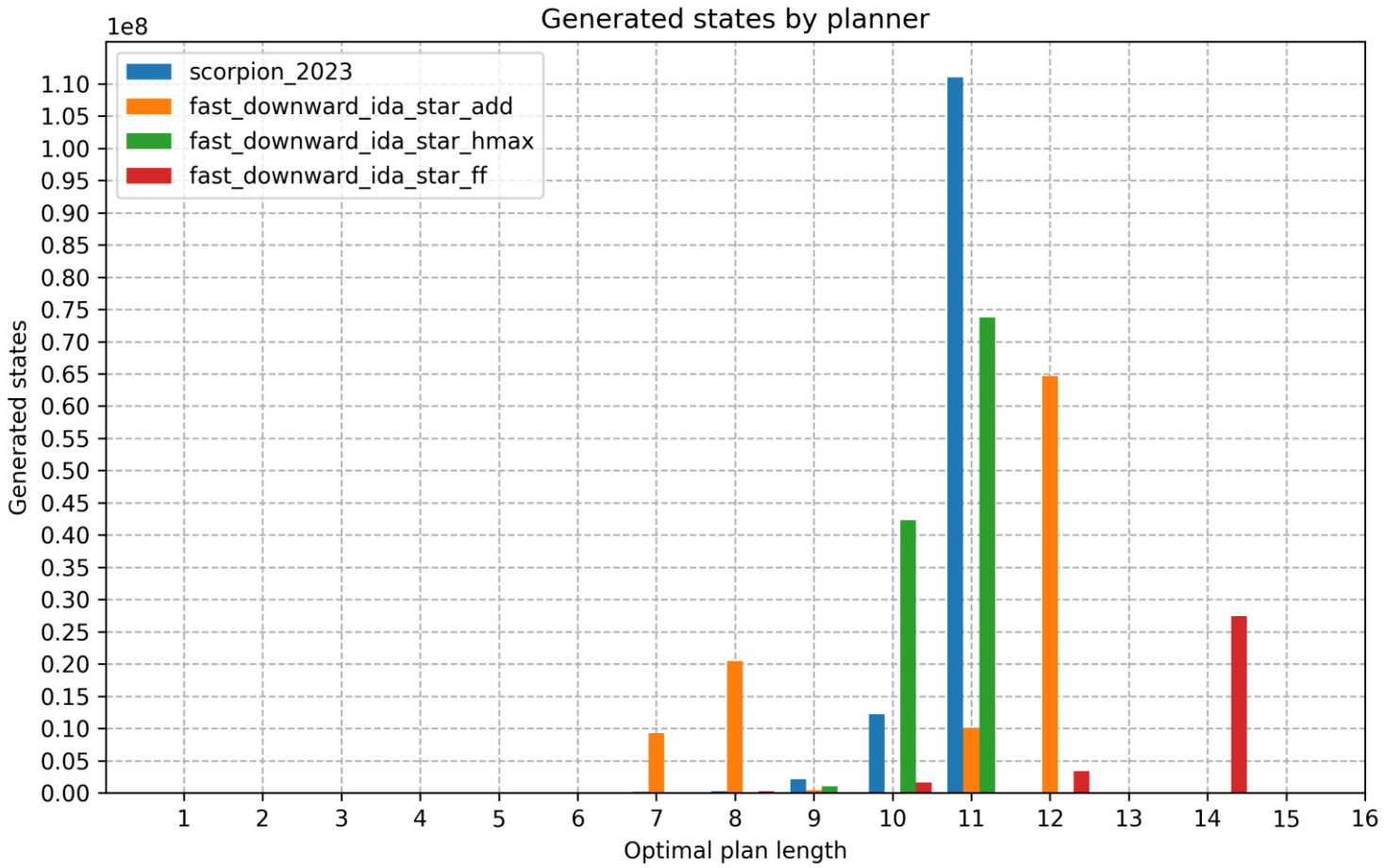
Results

Execution time: comments

- We can observe that the execution time follows an exponential function, consistent with the expectations of IDA*.
- The only difference between the heuristics is the timing of the explosion.

Results

Generated states



Results

Generated states: comments

- As we previously mentioned, the number of generated states in IDA* is directly proportional to the branch factor, which remains constant, and the maximum path length generated. Therefore, given that the number of generated states is exponential in the length of the optimal solution, it follows that the maximum path length generated is also exponential in the length of the optimal solution.

Conclusions

- Our result is not good, considering that the optimal solution length median is 18 moves, and we break the constraint after 11 moves.
- The Scorpion planner performs similarly to the IDA* hmax heuristic, making it a promising alternative for this problem.
- We know that the difficulty of Rubik's Cube problems can vary greatly, so conducting experiments with different problems or IPC planners can lead to different results.
- The code used to generate the results is available at: https://github.com/simonescaccia/Rubiks-Cube-planners-comparison