Solving the Rubik's Cube with a PDDL Planner

Bharath Muppasani, Vishal Pallagani, Kausik Lakkaraju,
Biplav Srivastava, Forest Agostinelli

AI Institute, University of South Carolina, Columbia, South Carolina, USA
{bharath@email., vishalp@email., kausik@email., biplav.s@, foresta@cse.}sc.edu

Abstract

Rubik's Cube (RC) is a popular puzzle that is also computationally hard to solve. In this demonstration, we introduce the first PDDL formulation for the 3-dimensional RC and solve it with an off-the-shelf Fast-Downward planner. We also create a plan executor and visualizer to show how the plan achieves the intended goal. Our system has types of two audiences: (a) planning researchers who can explore a hard problem and improve their planning algorithms, and (b) RC learners who want to learn how to solve the puzzle at their own pace and can now modify an initial plan (e.g., manually, using other algorithms) and see their execution.

1 Introduction

As artificial intelligence (AI) continues to solve problems that humans struggle to solve, there is an emerging need for humans to understand these solutions so that we can trust AI, create new educational opportunities, and even discover new knowledge. Many of these problems are path-finding problems. That is, the problem is to find a sequence of actions (a path) to go from any given state to a goal state. AI has been successfully applied to solve the Rubik's Cube (RC) [Agostinelli *et al.*, 2019; Lakkaraju *et al.*, 2022; Joyner, 2008; Agostinelli *et al.*, 2021] but these methods used opaque learning techniques which are hard for RC learners to benefit from.

While no PDDL encoding of a 3x3x3 RC problem is known to the authors, there is previous work[1] for a 2x2x2 RC setting and is solved with the Fast-Forward planner. Authors in [Büchner *et al.*, 2022] modeled the RC problem in finite domain representation, which enables the common general purpose solvers to be used on the RC problem. Our contributions are: (a) introducing the first PDDL formulation for a 3-size RC; planning researchers can use it to evaluate their planning algorithms, (b) enabling RC learners to use off-the-shelf planners to find custom and optimized way to solve any given RC configuration. Moreover, learning based RC solvers, which have been shown to scale to large instances, can use it as a labeled data generator for training. A demonstration can be seen at the url - here.

[1] https://wu-kan.cn/2019/11/21/Planning-and-Uncertainty/

2 Background

Rubik's Cube (RC)

The Rubik's Cube is a 3-D combination puzzle with colored faces made up of 26 smaller colored pieces linked to a central spindle, with the goal of rotating the blocks until each face of the cube is a single color. To solve the puzzle, one can perform certain actions that correspond to the different faces of the cube. The major actions of a Rubik's cube are Up(U), Down(D), Right(R), Left(L), Front(B), and Back(B), which define a rotation of 90 degrees in a clockwise direction of the respective face per action. The inverse of these actions corresponds to a 90-degree rotation in the anti-clockwise direction (suffix 'rev'). One can solve the RC from a scrambled state to the original configuration by performing a set of above mentioned actions.

While sticking to planning terminology of actions, problems and plans, we want to clarify terminology prevalent in RC literature that we will also refer to. A sequence of actions are called *macro-actions* and a collection of macro-actions are called *algorithms* in RC parlance. A solver may employ a strategy for sequencing macro-actions to solve the cube. We use the Fast-Downward AI Planner [Helmert, 2006] to solve the RC problem. Fast Downward is a domain-independent classical planning system based on heuristic search.

3 System Description

We now discuss how an RC can be modelled in PDDL and how the generated plan is linked with a visualizer for providing better understanding to people coming from a non-planning background as well.

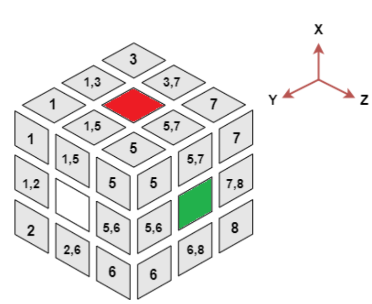

Figure 1: Rubik's cube description to define the domain encoding.

Listing 1: Action L of Rubik's Cube modeled in PDDL

```
(:action L
:effect (and
;for corner cubelets
(forall(?x ?y ?z)(when (cube1 ?x ?y ?z)
  (and (cube2 ?y ?x ?z))))
(forall(?x ?y ?z)(when (cube3 ?x ?y ?z)
  (and (cube1 ?y ?x ?z))))
(forall(?x ?y ?z)(when (cube4 ?x ?y ?z)
  (and (cube3 ?y ?x ?z))))
(forall(?x ?y ?z)(when (cube2 ?x ?y ?z)
  (and (cube4 ?y ?x ?z))))

;for edge cubelets
(forall(?x ?z)(when (edge13 ?x ?z)
  (and (edge12 ?x ?z))))
(forall(?y ?z)(when (edge34 ?y ?z)
  (and (edge13 ?y ?z))))
(forall(?x ?z)(when (edge24 ?x ?z)
  (and (edge34 ?x ?z))))
(forall(?y ?z)(when (edge12 ?y ?z)
  (and (edge24 ?y ?z))))))
```

RC representation in PDDL: In the PDDL domain, the Rubik's cube problem environment has been defined by assuming the cube pieces are in a fixed position, and are named accordingly, as defined in Figure 1. Each action in the RC environment is defined as the change of colors on these fixed cube pieces. The 3D axis of the cube is considered as three separate parameters X, Y, and Z that specify the position of the color on the cube's pieces. The three-color cubelet is specified as a predicate with three parameters: X, Y, and Z, which indicate the piece's colors on three separate axes. The two-color edge piece between the cubelets is specified as a predicate with two parameters denoting the piece's colors on the two axes. The predicate names define the fixed position of the cubelets and edge pieces that are defined with respect to the different faces of the cube. The representation considered for the cube positions is shown in the Figure 1. One of the actions, action 'L', of RC designed in PDDL of Rubik's cube from the description provided have been shown in Listing 1.

The PDDL for RC is modelled by considering the moves in the RC domain as change of colors on the cube pieces. When the move L is applied to the RC, for example, the left layer is rotated clockwise with respect to the left face. This may be regarded as a clockwise translation of colors from the left layer's cubelet and edge pieces. When the action L is performed on Figure 1, the colors on the pieces: cube1, cube2, cube4, cube3, are circularly shifted towards bottom. The same applies for the edge pieces. The change of color axis on these pieces is also handled accordingly.

Generating and Visualizing the Plan: Our system's PDDL encoded RC solver, in combination with a Visualizer, generates plan actions to solve the problem using the Fast-Downward (FD) AI planner. We have used publicly available

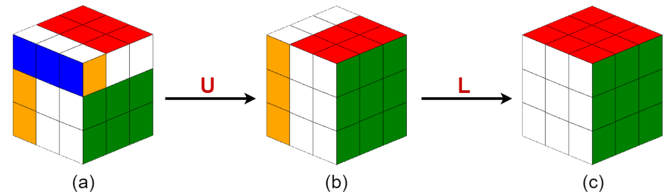

Figure 2: Shuffled state of the cube. Solution found with FD planner: U, L. Cost: 2.

three-rubiks-cube npm package [2] for 3D RC visualization. A random scrambled state of RC represented in the Visualizer is shown in the Figure 2. AI planners are controllable in generating the desired plans. We can specify the search algorithm and the heuristics to the Fast-Downward planner. Each different search algorithm generates different plans to reach the goal state. We employed A* search algorithm in combination with different heuristics which supports conditional-effects in our system and gives plans in minutes.

The system architecture is shown in Figure 3. The users need to upload the domain file and the problem file of RC and select a heuristic of their choice from the dropdown menu. The uploaded domain file and problem file along with the selected heuristic are sent to the API endpoint. The RC visualizer is scrambled to match the initial state from the uploaded problem file. On the back end, the Fast-Downward Planner with A* search along with the selected heuristic evaluates the uploaded problem and generates a solution. This solution is provided to the visualizer to solve the RC. The user may visually follow the actions in the plan file generated to observe the RC being solved step by step. Additional information on the parameters of search time, total time, evaluated states, expanded states, and generated states are also displayed in the front end.

4 Heuristic Evaluation

In our RC solver system, we have incorporated the Fast-Downward AI planner, which offers a range of heuristic implementations for efficient planning. As our RC PDDL employs conditional effects, we have selected a subset of heuristics that support conditional effects in domain modeling. A short description of these heuristics is provided below:

Blind heuristic is based on the idea that the search algorithm doesn't have any knowledge about the problem domain. It estimates the distance to the goal based on the number of actions needed to reach it, without considering their effects.

Max heuristic estimates the maximum cost of achieving any one of the goals, without considering their interdependencies.

Goal Count heuristic calculates the number of unsatisfied goals in the current state and estimates the cost of satisfying all of them. It works best when there is a small number of goals.

Landmark Cost Partitioning heuristic is based on the idea of breaking down the planning problem into smaller subproblems, then solving them separately. It uses a technique

[2] https://github.com/lab89/three-rubiks-cube

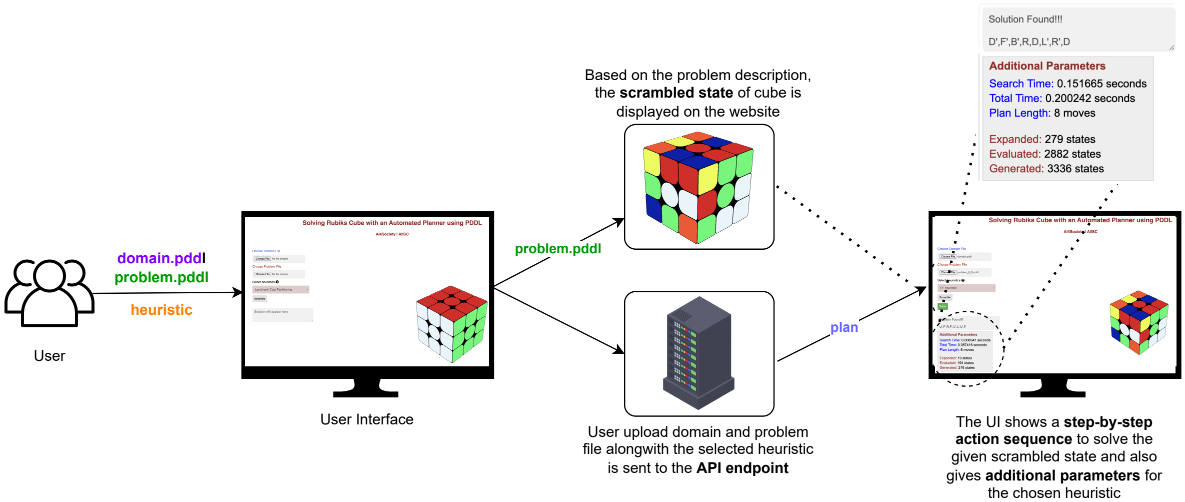

Figure 3: System Architecture.

called cost partitioning to distribute the estimated cost of achieving the goal among these sub-problems. [Karpas and Domshlak, 2009]

FF heuristic is based on a simple idea: to achieve a goal, find the shortest sequence of actions that lead to it. FF (Fast Forward) planner does a forward search from the initial state to the goal state. [Hoffmann, 2001]

Causal Graph heuristic constructs a graph representing the causal relationships between actions and the effects they have on the state. It estimates the cost of achieving the goal based on this graph. [Helmert, 2004]

Context-Enhanced Additive heuristic combines several sub-heuristics that take into account different aspects of the problem domain. It estimates the cost of achieving the goal by adding up the costs of these sub-heuristics and weighting them based on their relevance to the current state. [Helmert and Geffner, 2008]

In Table 1, we show the performance of these heuristics on two RC problems (P1 & P2) which are 7 & 8 moves away from the goal state respectively. Here we compare the heuristics across time and plan cost metrics. Apart from these metrics, we also display additional parameters, like the number of states generated before the solution is found, on our website (as shown in Figure 3). It can be seen that the context-enhanced additive heuristic is worst performing across these metrics for both the problems (P1 & P2).

5 Conclusion

In this work, we have demonstrated the capability of a planner to solve a complex puzzle, i.e., Rubik's Cube. For realizing this, we have created the first PDDL domain for RC. In order to make the generated plan understandable by people outside the planning community as well, we have integrated the generated plan with a visualizer showing step-by-step moves to achieve a fully solved RC. In the future, we would like to perform a comparative study of the performance of various planners and different encodings (PDDL vs. SAS+) to solve a given RC configuration. Additionally, an empirical study on the performance of abstraction heuristics on the RC modeled in PDDL, which showed some promising results on SAS+ encoding of RC [Büchner *et al.*, 2022], would be interesting. Integration of a suitable plan validator would be needed so that human-edited plans can be verified before execution (currently, VAL [Howey *et al.*, 2004] does not handle conditional effects). This would help us to assist a learner to solve RC under various constraints such as time or moves.

Heuristic \Problem	**P1**		**P2**	
	time (s)	**cost**	**time (s)**	**cost**
LM Cost Partitioning	0.3614	7	1.5284	8
FF	0.0777	7	0.1607	8
Goal count	4.8279	7	75.6992	8
Blind	28.1153	7	274.05	8
Max	0.4576	7	3.9793	8
Causal Graph	498.73	**21**	52.3757	**18**
Context-enhanced additive	**741.146**	**21**	**526.99**	**18**

Table 1: Comparison of different heuristics on two RC problems.

References

[Agostinelli *et al.*, 2019] Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Mach. Intell.*, 1(8):356–363, 2019.

[Agostinelli *et al.*, 2021] Forest Agostinelli, Mihir Mavalankar, Vedant Khandelwal, Hengtao Tang, Dezhi Wu, Barnett Berry, Biplav Srivastava, Amit Sheth, and Matthew Irvin. Designing children's new learning partner: Collaborative artificial intelligence for learning to solve the Rubik's cube. In *Interaction Design and Children*, pages 610–614, 2021.

[Büchner *et al.*, 2022] Clemens Büchner, Patrick Ferber, Jendrik Seipp, and Malte Helmert. A comparison of abstraction heuristics for rubik's cube. In *ICAPS 2022 Workshop on Heuristics and Search for Domain-independent Planning*, 2022.

[Helmert and Geffner, 2008] Malte Helmert and Héctor Geffner. Unifying the causal graph and additive heuristics. In *ICAPS*, pages 140–147, 2008.

[Helmert, 2004] Malte Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, volume 16, pages 161–170, 2004.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246, jul 2006.

[Hoffmann, 2001] Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57–57, 2001.

[Howey *et al.*, 2004] Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301. IEEE, 2004.

[Joyner, 2008] David Joyner. *Adventures in group theory: Rubik's Cube, Merlin's machine, and other mathematical toys*. JHU Press, 2008.

[Karpas and Domshlak, 2009] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *IJCAI*, pages 1728–1733. Pasadena, CA, 2009.

[Lakkaraju *et al.*, 2022] Kaushik Lakkaraju, Thahimum Hassan, Vedant Khandelwal, Prathamjeet Singh, Cassidy Bradley, Ronak Shah, Forest Agostinelli, Biplav Srivastava, and Dezhi Wu. ALLURE: A multi-modal guided environment for helping children learn to solve a Rubik's cube with automatic solving & interactive explanation. In *AAAI*, 2022.