Comparing Rubik's Cube Solvability in Domain-Independent Planners Using Standard Planning Representations for Insights and Synergy with Upcoming Learning Methods

Bharath Muppasani, Vishal Pallagani, Biplav Srivastava, Forest Agostinelli

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

Abstract

The Rubik's Cube (RC) is a renowned puzzle that has spurred AI researchers to investigate efficient representations and solution techniques. This paper introduces the first RC representation in the widely-used PDDL language, enhancing its accessibility to PDDL planners, competitions, and knowledge tools, and improving human readability. We then bridge across existing approaches and compare performance using the IPC-2023 benchmark dataset for RC. In a benchmark experiment, DeepCubeA[1] optimally solves all tasks; Scorpion, using SAS+ and pattern database heuristics, optimally solves 75% of tasks; and FastDownward, with PDDL representation and the FF heuristic, solves 60% of tasks, with 83.33% being optimal. Notably, the introduced PDDL domain for RC was notorious at the 2023 International Planning Competition (IPC) for posing a significant challenge for emerging planners. While the maximum any planner could solve optimally was only 55% of the tasks, our results show improvement over them. Our findings shed light on the interplay between representational choices and plan optimality, guiding future strategies that amalgamate general-purpose solving methods, heuristics, and both standard and custom representations.

Notations	Description
RC	Rubik's Cube
PDDL	Planning Domain Definition Language (McDermott 2000)
Custom	RC representation in DeepCubeA (Agostinelli et al. 2019)
h^{Blind}	FastDownward with Blind[3]
h^{Max}	FastDownward with Max-cost[3]
h^{GC}	FastDownward with Goal count[3]
h^{CG}	FastDownward with Causal Graph[3]
h^{CEA}	FastDownward with Context-enhanced Additive[3]
$h^{\text{LM-Cost}}$	FastDownward with Landmark Cost Partitioning[3]
h^{FF}	FastDownward with FF[3]
$h^{\text{M\&S}}$	Scorpion with Merge & Shrink[4]
$h^{\text{PDB-Man}}$	Scorpion with Max Manual PDB[4]
$h^{\text{PDB-Sys}}$	Scorpion with Max Systematic PDB[4]

Table 1: Notations and their descriptions.

1 Introduction

The Rubik's Cube is a 3D puzzle game that has been widely popular since its invention in 1974. It has been a subject of interest for researchers in Artificial Intelligence (AI) due to its computational complexity and potential for developing efficient problem-solving algorithms. RC has motivated researchers to explore alternative representations that simplify the problem while preserving its complexity. Efficient algorithms have been developed to solve RC in the least number of moves, and they have been used in various applications, including robot manipulation, game theory, and machine learning. Therefore, in this paper, we aim to explore the different representations and algorithms to solve RC and evaluate their performance and effectiveness in solving this challenging puzzle.

Various solution approaches have been proposed RC including Reinforcement Learning (RL) and search. For instance, DeepCubeA (Agostinelli et al. 2019) uses RL to learn policies for solving RC, where the cube state is represented by an array of numerical features. Although DeepCubeA is a domain-independent puzzle solver, it employs a custom representation for RC. On the other hand, Büchner et al. (2022) utilized SAS+ representation to model the RC problem in a finite domain representation, which enables standard general-purpose solvers like Scorpion to be used on the RC problem. Despite the success of these approaches, no prior work has explored the use of Planning Domain Definition Language (PDDL) to encode a 3x3x3 RC problem, while a previous study[2] has encoded a 2x2x2 RC problem using PDDL and solved it with a Fast-Forward planner.

In this paper, we introduce a novel approach for representing RC in PDDL. We encode the initial state and goal state using a set of predicates, each of which specifies the color of a sticker on a particular cube piece or edge piece. We then define the actions that can be taken to manipulate the cube pieces and edges. Our PDDL representation en-

[1]DeepCubeA trained with 12 RC actions

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

ables us to model RC as a classical planning problem, which can be solved using off-the-shelf planning tools. To the best of our knowledge, this is the first attempt to represent RC formally using PDDL. We also evaluate the effectiveness of our approach by comparing it with other state-of-the-art representations in terms of the efficiency and effectiveness of problem-solving. Our major contributions are:

- We develop the first PDDL formulation for the 3x3x3 Rubik's Cube, which is a novel and significant contribution to the existing literature. This PDDL formulation will enable the use of standard PDDL planners for solving Rubik's Cube problems, which was not previously possible.
- We bridge across hither-to incomparable RC solving approaches, compare their performance and draw insights from results to facilitate new research.
- We perform a comparative analysis of two standard representations, SAS+ and PDDL, and a custom representation in DeepCubeA, an RL approach for solving RC on a set of common benchmark RC problems. This comparative analysis is important as it provides insights into the strengths and weaknesses of these different approaches, and helps to identify which method may be most appropriate for a given problem setting.
- We contextualize the IPC results using the encoding.

Our results indicate several key insights: SAS+ emerges as the best representation among those considered for solving the Rubik's Cube. While deep learning methods exhibit strong performance, their applicability is limited to the training data. By combining advancements from both automated planning and deep learning fields, significant improvements in solving the Rubik's Cube can be achieved, provided common baselines are established.

The paper is organized as follows: We overview Rubik's Cube-solving ecosystem, including the RC problem, domain-independent planners and heuristics, and learning-based RC solvers. Then, we compare three representations for RC: DeepCubeA, SAS+, and PDDL. Next, we outline the experiments conducted, including the heuristics considered and the experimental setup, followed by the result analysis. We compare RC solvers and heuristics for the number of problems solved and plan optimality. Furthermore, the introduced PDDL domain for RC was notorious at IPC 2023 for posing a significant challenge to emerging planners. The detailed results, showcasing the performance of various planners on RC domain, are provided in the supplementary material.

2 The RC Solving Ecosystem

In this section, we describe the RC problem, planners, and heuristics that are used for our study. Table 1 summarizes the notations used.

[3]https://www.fast-downward.org/Doc/Evaluator

[4]https://jendrikseipp.github.io/scorpion/Evaluator/

RC Problem

The Rubik's Cube is a 3-D combination puzzle consisting of 26 smaller colored pieces anchored to a central spindle. The objective is to manipulate the cube until each face displays a uniform color. The primary rotations correspond to the cube's faces: Up (U), Down (D), Right (R), Left (L), Front (F), and Back (B), each representing a 90-degree clockwise turn of the respective face. Their inverses (denoted with a 'rev' suffix) indicate a counter-clockwise rotation. The cube is initially rotated by a random sequence of rotations in the puzzle's initial configuration. The goal is to find a series of rotations that results in the solved state, where each face has a single color. One can solve the RC from a scrambled state to the original solved configuration by performing a set of the above-mentioned actions.

Domain-Independent Planners and Heuristics

Classical Planning Formalism Consider F to be a set of propositional variables or *fluents*. A *state* $s \subseteq F$ is a subset of fluents that are true, while all fluents in $F \setminus s$ are implicitly assumed to be false. A subset of fluents $F' \subseteq F$ holds in a state s if and only if $F' \subseteq s$. A classical planning instance is a tuple $P = \langle F, A, I, G \rangle$, where F is a set of fluents, A is a set of actions, $I \subseteq F$ an initial state, and $G \subseteq F$ a goal condition. Each action $a \in A$ has precondition pre(a) $\subseteq F$, add effect add(a) $\subseteq F$, and delete effect del(a) $\subseteq F$, each a subset of fluents. Action a is applicable in state $s \subseteq F$ if and only if pre(a) holds in s, and applying a in s results in a new state $s \oplus a = (s \setminus \text{del}(a)) \cup \text{add}(a)$. A *plan* for P is a sequence of actions $\prod = \langle a_1, ..., a_n \rangle$ such that a_1 is applicable in I and, for each $2 \leq i \leq n, a_i$ is applicable in $I \oplus a_1 \oplus ... \oplus a_{i-1}$. The plan $\prod$ solves P if G holds after applying $a_1, ..., a_n$, i.e. $G \subseteq I \oplus a_1 \oplus ... \oplus a_n$.

Abstractions Let $\mathcal{T} = \langle S, \mathcal{L}, T, s_I, S_* \rangle$ be a transition system. An *abstraction* $\alpha : S \to S^\alpha$ maps the states of $\mathcal{T}$ to a set of *abstract states* S^α. The induced transition system is $\mathcal{T}^\alpha = \langle S_\alpha, \mathcal{L}, T^\alpha, \alpha(s_I), \{\alpha(s) | s \in S_*\} \rangle$ where $T^\alpha = \{\langle \alpha(s), o, \alpha(s') \rangle | \langle s, o, s' \rangle \in T\}$. By construction, every path in $\mathcal{T}$ is a path in $\mathcal{T}^\alpha$. Consequently, the length of the shortest path between state $\alpha(s)$ and $\alpha(s')$ in $\mathcal{T}^\alpha$ is a lower bound on the length of the shortest path between state s and s' in $\mathcal{T}$. Thus, the abstract goal distance for a given state is an admissible estimate of the true goal distance (Büchner et al. 2022). In the later section of the paper, we mention the abstraction heuristics used for our work.

PDDL Planning has evolved through various representations like STRIPS (Fikes and Nilsson 1971), ADL (Pednault 1994), and SAS+ (Bäckström 1995). However, the Planning Domain Description Language (PDDL) (McDermott et al. 1998; Fox and Long 2003) has become the standard notation. Planners often use PDDL for problem specification, although they might convert to other formats for efficiency (Helmert 2009). PDDL requires a domain description file detailing general information and a problem description file defining the initial and goal states. A planner uses both files to produce a plan, verifiable by tools like VAL (Howey and Long 2003).

International Planning Competition

The International Planning Competition (IPC) is a key event for evaluating planning systems. Through its annual benchmarks, IPC highlights the latest challenges and innovations in planning. The RC 12 action PDDL domain in IPC-2023 Classical track (Taitler et al. 2024) stood out as one of the most challenging, emphasizing the intricacy of the RC problem[5]. Detailed results on RC from IPC 2023 can be found in the supplementary material.

Learning-based RC Solver

There exist specialized solvers for solving the Rubik's Cube, which can be classified as either domain-dependent or domain-independent. DeepCubeA is an example of a domain-independent solver that employs domain-dependent custom representation encoding for RC, as proposed in (McAleer et al. 2018; Agostinelli et al. 2019). The solver trains a deep neural network to be a heuristic function using approximate value iteration and uses the learned heuristic function to guide weighted A* search. However, although DeepCubeA was able to find an optimal path in the majority of verifiable test cases, DeepCubeA does not have optimality guarantees.

3 Comparision of RC representations

In this section, we describe and provide a comparative analysis of different RC representations comprising of RL and Planning formal languages.

DeepCubeA

The DeepCubeA algorithm adopts a unidimensional array as a representation of the Rubik's Cube (RC) state. Specifically, this array encompasses 54 elements, each of which corresponds to a unique sticker color present on a cube piece of the RC. While this array-based modeling offers computational advantages, it is limited by its inability to fully encapsulate the spatial orientation of Rubik's Cube. Furthermore, the usage of a hard-coded representation and implicit assumptions concerning the position of cubelets poses a challenge to novice users seeking to comprehend the array-based representation.

SAS+

In Büchner et al. 2022, Rubik's Cube is modeled with 18 actions in SAS+ representation as a factored effect task, with each face labeled as F, B, L, R, U, or D. The orientation of each cube piece is represented as a triple of values, and for corner cube pieces, the orientation is a permutation of {1, 2, 3}, while for edge cube pieces, it is a permutation of {1, 2, #} (where # represents a blank symbol). The rotation of the cube in 3D space is captured as the permutation of the respective triple for each cube piece. In our experimentation, we have considered the SAS+ model with 12 actions, to be consistent with the IPC-2023. Additionally, we provide results and insights using the 18 action model in the supplementary material. The SAS+ model has 20 variables, each having 24 distinct values, leading to 480 fact pairs, and uses 16 bytes to represent a single RC state. Every variable accounts for all possible positions and orientations of a cube piece.

[5]None of the planners in IPC-23 were able to solve more than 11 problem instances optimaly

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 (not(cube5 ?x ?y ?z)) (cube2 ?y ?x ?z
      ))))
(forall(?x ?y ?z)(when (cube3 ?x ?y ?z)
  (and (not(cube3 ?x ?y ?z)) (cube1 ?y ?x ?z
      ))))
(forall(?x ?y ?z)(when (cube4 ?x ?y ?z)
  (and (not(cube4 ?x ?y ?z)) (cube3 ?y ?x ?z
      ))))
(forall(?x ?y ?z)(when (cube2 ?x ?y ?z)
  (and (not(cube2 ?x ?y ?z)) (cube4 ?y ?x ?z
      ))))
;for edge cubelets
(forall(?x ?z)(when (edge13 ?x ?z)
  (and (not(edge13 ?x ?z)) (edge12 ?x ?z))))
(forall(?y ?z)(when (edge34 ?y ?z)
  (and (not(edge34 ?y ?z)) (edge13 ?y ?z))))
(forall(?x ?z)(when (edge24 ?x ?z)
  (and (not(edge24 ?x ?z)) (edge34 ?x ?z))))
(forall(?y ?z)(when (edge12 ?y ?z)
  (and (not(edge12 ?y ?z)) (edge24 ?y ?z))))
  ))
```

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. These fixed cube pieces are modeled as predicates in the RC domain and the colors they possess in the three-dimensional space as parameters of these predicates. With the help of conditional effects, 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 colors on the cube's pieces. One of these axes can be connected to each face of the cube. According to the representa-

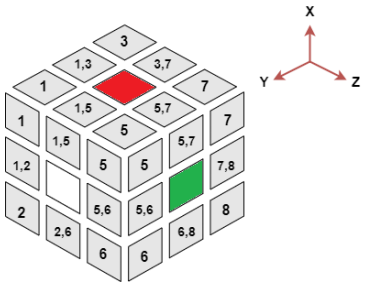


Figure 1: Rubik's cube description to define the domain encoding.

Planner with Heuristic	PDDL					SAS+				
	# Solved	**Cost**	**# Nodes**	**Secs**	**Mem (MB)**	**# Solved**	**Cost**	**# Nodes**	**Secs**	**Mem (MB)**
h^{Blind}	8 (100%)	4.5	4.52E+06	22.92	338.58	8 (100%)	4.5	1.37E+07	49.26	572.81
h^{CEA}	8 (75%)	7.38	1.08E+06	80.80	105.48	9 (55.55%)	10.11	3.84E+06	72.27	186.04
h^{CG}	9 (77.77%)	7.89	1.13E+07	165.50	1103.55	10 (50%)	11.7	1.87E+07	131.01	882.00
h^{Max}	10 (100%)	5.5	3.95E+06	155.12	324.82	11 (100%)	6.0	1.04E+07	136.51	543.48
$h^{LM-Cost}$	10 (100%)	5.5	1.33E+06	9.42	124.77	11 (100%)	6.0	1.13E+07	74.03	528.25
h^{GC}	11 (100%)	6.09	4.71E+06	25.84	405.09	11 (100%)	6.09	4.71E+06	11.61	245.69
h^{FF}	**12** (83.33%)	7.17	2.45E+05	11.09	38.44	12 (100%)	6.5	1.25E+06	24.46	72.40
${}^{+}h^{M\&S}$	11 (100%)	6.0	6.18E+06	66.39	512.81	11 (100%)	6.0	8.19E+06	18.03	392.18
${}^{+}h^{PDB-Man}$	9 (100%)	5.0	3.81E+06	20.76	278.12	**15** (100%)	8.0	7.57E+06	20.69	518.24
${}^{+}h^{PDB-Sys}$	9 (100%)	5.0	3.43E+06	127.24	270.31	14 (100%)	7.5	2.88E+06	80.46	2366.62
Ragnarok*	10 (100%)	-	-	-	-	-	-	-	-	-

	# Solved	**Cost**	**# Nodes**	**Secs**	**Mem (MB)**
DeepCubeA	20 (100%)	10.5	9.67+E05	18.13	-

Table 2: Comparison of planner configurations including the total number of problems solved, along with the percentage of optimal plans, average number of nodes generated, average search time in seconds, average memory usage in MB, and average plan cost for the solved instances across different Rubik's Cube models evaluated on the IPC-2023 dataset. (* - IPC-2023 Classical optimal-track winner); + - Admissible heuristics

tion shown in Figure 1, the respective faces on each axis are: $F_X = \langle U, D \rangle$; $F_Y = \langle F, B \rangle$; $F_Z = \langle R, L \rangle$. These different faces of the cube can be identified by the color of the middle cube piece. We considered White, Red, and Green colors as the colors on the front(F), up(U) and right(R) faces respectively (similarly, the counter colors on the counter faces).

The following conventions regarding the RC cube pieces are considered to model the RC domain actions in the PDDL:

1. The corner cube pieces of the RC are modeled as a three-color cubelet and are specified as a predicate with three parameters: *x*, *y*, and *z*, which indicate the piece's colors on three separate axes. There are 8 corner pieces in RC.
2. The edge cube pieces, which are in between corner cube pieces, are modeled as two-color cubelet and is specified as a predicate with two parameters denoting the piece's colors on the two axes. There are 12 edge pieces in RC.
3. We do not consider the rotations performed on the middle layer, as this can be resolved into rotation of right and left faces in the opposite direction. As a result, the middle cube piece of a face is unaltered.

The predicate names define the fixed position of the cubelets that are defined with respect to the different faces of the cube. The representation considered for the cube positions is shown in Figure 1. One of the actions, action 'L', of RC designed in PDDL from the description provided is shown in Listing 1. In this, we refer to corner cube pieces as *cubeP* and edge cube pieces as *edgePQ* where *P* and *Q* are the numbers for the cube pieces as stated in Figure 1. When the move L is applied to the RC, for example, the left face is rotated clockwise. This may be regarded as a 90-degree clockwise translation of colors from the left-face corner and edge cube pieces. Considering the RC representation shown in Figure 1, the colors on the pieces: cube1, cube2, cube4, and cube3, are circularly shifted towards the right. The same applies to the edge pieces. As the left face falls in the Z-plane, only the X-axis and Y-axis colors on the cube pieces are affected.

Definitions and Notations

- C: Set of colors, e.g., $C = \{$*White, Red, Green, ...*$\}$.
- F: Set of faces, with $F_X = \{U, D\}$, $F_Y = \{F, B\}$, and $F_Z = \{R, L\}$.
- A: Set of axes, $A = \{X, Y, Z\}$.
- *cube*(x, y, z): Corner cube piece with colors $x, y, z \in C$ on axes X, Y, Z.
- *edge*(x, y): Edge cube piece with colors $x, y \in C$ on two axes.

Actions and Effects Actions correspond to 90-degree rotations of Rubik's Cube faces. For face $f \in F$, action *f*() (and *frev*()) denotes clockwise (and counter-clockwise) rotations. When rotating face f aligned with axis A, colors on the other axes translate:

- $f \in F_X$: *cube*(x, y, z) to *cube*(x, z, y) and *edge*(x, y) swaps with *edge*(x, z).
- $f \in F_Y$: *cube*(x, y, z) to *cube*(z, y, x) and *edge*(x, y) swaps with *edge*(y, z).
- $f \in F_Z$: *cube*(x, y, z) to *cube*(y, x, z) and *edge*(x, z) swaps with *edge*(y, z).

State and Transition A state s represents the Rubik's Cube configuration as $s = \{$*cube*(x_i, y_i, z_i), *edge*$(x_i, y_i), ...\}$ for $x_i, y_i, z_i \in C$. The transition function $T : S \times A \rightarrow S'$ defines state changes post-action.

During the execution of a problem, FastDownward (Helmert 2006) first translates the domain into a SAS formulation. The resulting SAS version of the RC PDDL domain model has 480 variables each being binary-valued, resulting in 960 fact pairs, and requires 60 bytes to represent a single state of RC. Each variable represents a single configuration of a cube piece.

4 Experiments

In the following section, we will discuss the heuristics considered in our evaluation and the experimental setup, which includes the datasets, problem representations, and details about the planner.

Heuristics Considered

The heuristics considered for our evaluation are presented in 1. Due to space constraints in the main paper, a comprehensive description of each heuristic's workings is provided in the supplementary material. While all heuristics were employed in their default configurations, exceptions were made for the abstract heuristics $h^{\text{M\&S}}$, $h^{\text{PDB-Man}}$, and $h^{\text{PDB-Sys}}$.

Merge and Shrink Heuristic Generates lower bounds in factored state spaces using state merging and shrinking, balancing abstraction size and heuristic accuracy (Helmert et al. 2014). Specific strategies include bisimulation (Nissim, Hoffmann, and Helmert 2011), strongly connected components (Sievers, Wehrle, and Helmert 2016), and exact label reduction (Sievers, Wehrle, and Helmert 2014).

Pattern Database Heuristic The key step in using Pattern Database (PDBs) heuristics is selecting appropriate patterns for the problem at hand. Korf (1997) specified two sets of patterns for solving the Rubik's Cube. We evaluate two settings of PDBs:

Max Manual PDB: Inspired by Korf's patterns, Büchner et al. (2022) have considered 2 patterns for the corner cube pieces and 3 patterns for the edge cube pieces resulting in 4 variables for each pattern. We have considered these patterns for the evaluation of PDDL and SAS+ models.

Max Systematic PDB: This configuration systematically generates all interesting patterns up to a certain size (Pommerening, Röger, and Helmert 2013). A pattern size of 3 has been considered for this evaluation in the interest of memory constraints.

Experimental Setup

We evaluated the performance of our RC PDDL model by utilizing the benchmark problem test set from IPC-23, provided for the Rubik's Cube domain. This test set comprises of 20 problems, each with varying levels of difficulty. To further assess the SAS+ model, we converted the problem test set into its SAS+ version. Moreover, for the DeepCubeA model evaluation, we transformed the problem test set into a custom representation specific to DeepCubeA.

To evaluate the RC PDDL model and SAS+ model, we have used Fast-Downward (Helmert 2006) and Scorpion planner (Seipp, Keller, and Helmert 2020), which is an extension of the Fast-Downward planner. Scorpion planner contains the implementation for PDBs that support conditional effects modeled in the domain file. We perform A* searches (Russell and Norvig 2005) with each heuristic mentioned above on the test set and the two RC representations. We bound the A* search with an overall time limit of 30 minutes and a memory limit of 8GB as per IPC standards. This constraint is the same for the abstraction heuristics as well, despite the fact that these heuristics require significant time for preprocessing and generating abstractions prior to the start of the search.

5 Result Analysis

We conducted an empirical evaluation of the performance of PDDL and SAS+ models, on the IPC test dataset. We also evaluated the test datasets using DeepCubeA (Agostinelli et al. 2019), a state-of-the-art domain-independent RC solver that leverages a combination of deep reinforcement learning and search algorithms. Our results show that DeepCubeA was able to solve all the problems optimally. Table 2 presents the experimental results, including the total number of problems solved and the percentage of optimal plans generated for each configuration tested. The optimal plans we used for comparison were generated during the IPC-23 competition, and are available on the IPC GitHub repository[6]. Figure 2 illustrates the number of states expanded in the A* search algorithm where the x-axis represents the problem number and the y-axis represents the number of evaluated states. Similarly, Figure 3 presents a comparison of the runtime and memory usage for all the considered heuristics where the x-axis represents the problem number and the y-axis represents the memory usage in KB, in Figure 3(a), and total planner time in seconds, in Figure 3(b).

PDDL vs SAS+

When assessing the efficacy of planning-based solvers in terms of their heuristics and representations, our findings indicate that no planner configuration was able to solve problems with optimal plan lengths exceeding 15 steps in the IPC dataset. In terms of the number of problems solved, the h^{FF} is the best performing using PDDL representation. While PDBs performed much better in the SAS+ representation than in the PDDL representation. This can also be inferred from the states expansion trend of the abstraction heuristics shown in Figure 2.

h^{Blind} Predictably, h^{Blind} underperformed compared to other heuristics in both PDDL and SAS+ representations due to its lack of informative guidance. This is evident from the extensive states it evaluated in Figure 2 and its rapid memory saturation depicted in Figure 3. Even for simpler problem instances, its resource consumption was significantly higher than other heuristics.

h^{CG} and h^{CEA} RC, a puzzle-solving domain devoid of modeled preconditions, poses challenges for heuristics like h^{CG} and h^{CEA} in both representations. The domain's complexity and vast branching factor hinder the construction of

[6]https://github.com/ipc2023-classical/ipc2023-dataset/tree/main/opt/rubiks-cube

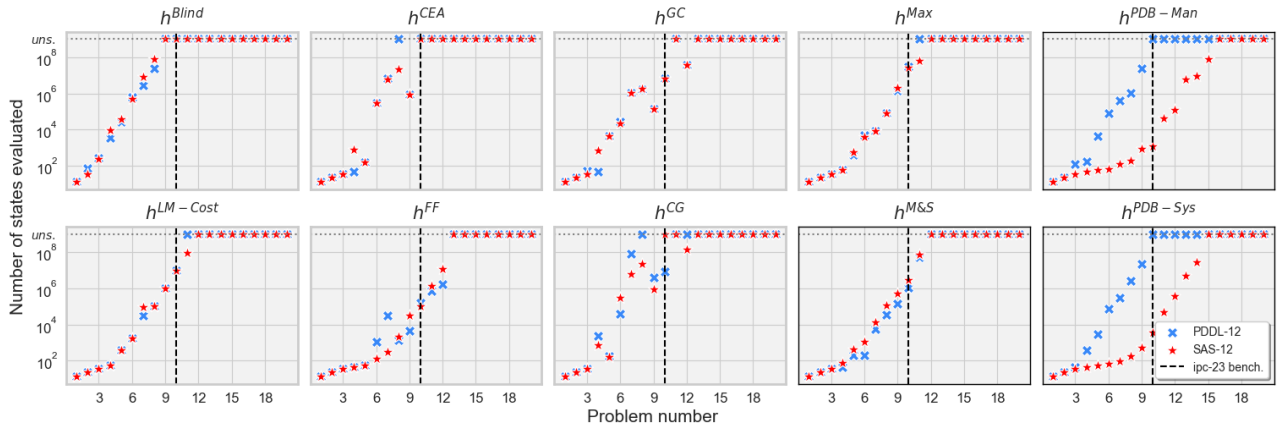
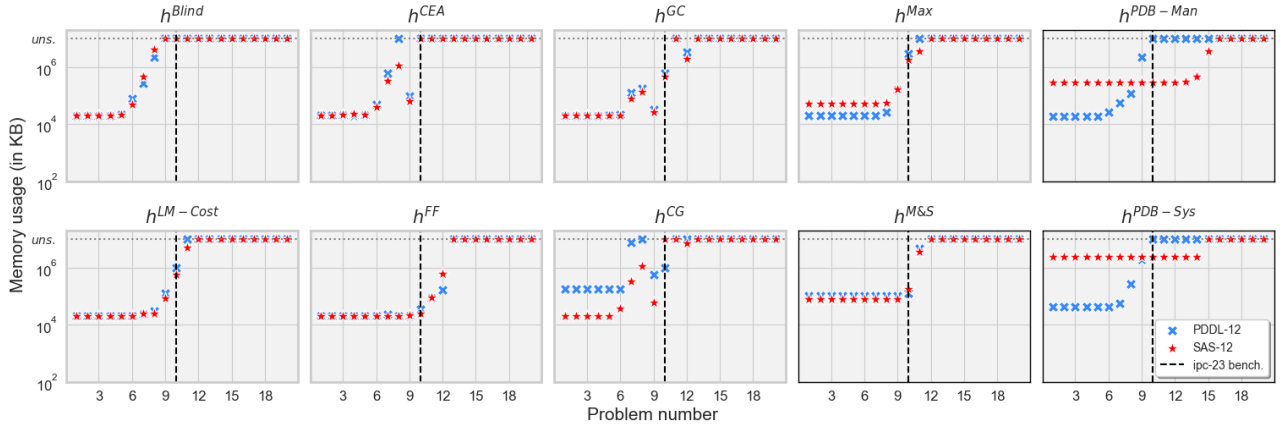
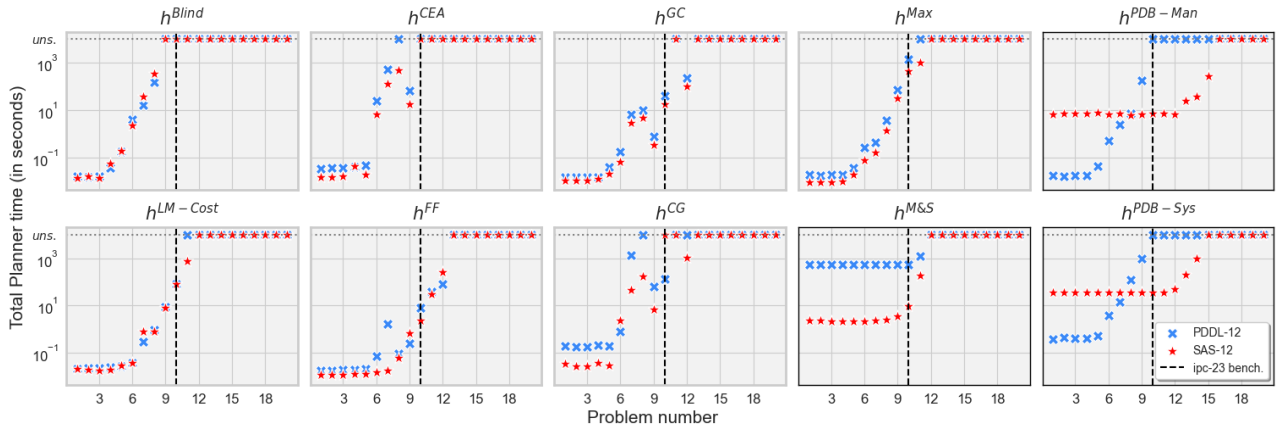


Figure 2: State expansion comparison for the IPC-2023 dataset. The x-axis denotes problem numbers, and the y-axis shows the states evaluated. A top horizontal dotted line indicates unsolved problems, while a vertical dashed line marks the problems solved by the IPC-2023 Classical optimal track winner.



(a) Memory usage comparison. x-axis represents the problem number, while y-axis represents the memory used in KB.



(b) Run time comparison. x-axis represents the problem number, while y-axis represents the total planner time in seconds. Overall the runtime is observed to be higher for PDDL than SAS+ representation.

Figure 3: Comparison of memory usage and runtime for different heuristics and models for IPC-2023 dataset. Plots in bold outline are for the abstract heuristics. A top horizontal dotted line indicates unsolved problems, while a vertical dashed line marks the problems solved by the IPC-2023 Classical optimal track winner.

a precise causal graph for h^{CG}. Similarly, the absence of contextual data limits the efficacy of h^{CEA}. This is underscored by the irregular state evaluation trend observed in Figure 2. Furthermore, these heuristics yield a lower percentage of optimal plans compared to other methods.

$h^{\textbf{Max}}$ demonstrated consistent performance across both PDDL and SAS+ representations, solving 10 and 11 problems respectively with 100% optimality. Its approach of recursively determining the maximum cost for each goal provides a balanced search guidance, as reflected in its results in Figure 2.

$h^{\textbf{LM-Cost}}$ Similarly, $h^{\text{LM-Cost}}$ showcased a consistent performance in both representations, solving 10 and 11 problems with 100% optimality. Distributing the costs of operators among the landmarks they achieve, $h^{\text{LM-Cost}}$ ensures admissibility and provides a more informed estimate of the cost to reach the goal (Karpas and Domshlak 2009). Its performance trend can be observed in Figure 2.

$h^{\textbf{GC}}$ which estimates the number of unsatisfied goals, performed optimally in both representations, solving 11 problems, which is greater than the best planner in optimal-track from IPC-2023 classical track.

$h^{\textbf{FF}}$ derived from the FF planning system (Hoffmann 2001), emerged as one of the top performers, especially in the PDDL representation where it solved 12 problems with 83.33% optimality. This heuristic showcased a notable difference in optimality between the PDDL and SAS+ representations. For the SAS+ representation, while it also solved 12 problems, it achieved 100% optimality. This disparity underscores the influence of the problem representation on the heuristic's ability to find optimal solutions. The performance and resource utilization trends for both representations can be observed in Figure 2 and Figure 3.

It has been observed that abstract heuristics are sensitive to problem representation and exhibit poorer performance in PDDL compared to SAS+ representation due to their expressivity.

$h^{\textbf{M\&S}}$ exhibited similar performance for both PDDL and SAS+ in terms of problem-solving. However, differences emerged in resource consumption. Using the PDDL representation, although the number of states evaluated is less when compared to the SAS+ representation, the $h^{\text{M\&S}}$ showed increased memory usage and longer preprocessing times, as illustrated in Figure 3. This is largely due to the state size: 60 bytes for PDDL compared to 16 bytes for SAS+. The M&S heuristic, which creates abstractions by merging state variables and shrinking state spaces, is more resource-intensive with PDDL's larger states, leading to quicker memory saturation and extended preprocessing.

Pattern Database Heuristic The choice of representation has profound implications for the PDB heuristic:

- **Pattern Size Limitation**: The h^{PDB} imposes constraints on the pattern size, which directly impacts memory usage and the computational time for building projections. Given that the SAS+ representation has a single variable for each cube piece, effective patterns can be selected using a small pattern size. While this is not the case with PDDL as there exist 24 variables for each cube piece.
- **Projection Complexity for SAS+ vs. PDDL**: In SAS+, a small pattern size can lead to many states in projections due to the multiplicity of values a single variable can assume. For instance, representing corner cubies on the Upper Face in SAS+ requires 4 variables, resulting in 331776 states. In contrast, PDDL, using binary-valued variables, needs 96 variables for the same representation. This not only increases computational time but also surpasses the h^{PDB} pattern size constraints, making PDDL less efficient to express useful patterns. A pattern size of 4 variables in PDDL results in only 16 states in projections. This difference can be observed from the memory usage and runtime plots in Figure 3.

Standard vs. Custom Representation

Our experiments, summarized in Table 2, contrast the performance of standard (PDDL and SAS+) and custom (DeepCubeA) representations for the Rubik's Cube problem. While FastDownward with the FF heuristic excelled for PDDL and Scorpion with Max Manual PDB for SAS+, both were outperformed by the tailored approach of DeepCubeA, which solved all tasks optimally. This underscores the trade-off: standard representations offer broad applicability but may be surpassed in efficiency by specialized solutions. The choice hinges on the problem's complexity and the desired solution's precision.

IPC Results Discussion:

In the International Planning Competition (IPC) 2023, the PDDL 12-action version was introduced for Rubik's Cube PDDL domain in the Classical Track. The results from IPC-2023 (Taitler et al. 2024) offered a revealing look into the capabilities of various planners when faced with the Rubik's Cube domain. Planners like Ragnarok (Drexler et al. 2023), Scorpion-2023 (Seipp 2023), Odin (Drexler, Seipp, and Speck), and Dofri (Höft, Speck, and Seipp 2013) showcased a similar performance by solving 10 problems out of 20. However, it was the Hapori-Stonesoup-opt (Ferber et al.) that scored highest by solving 11 out of 20 problems. When we compare these performances with the results of our work, the effects of problem representation on solving complex tasks become evident. For instance, using the Scorpion planner with Max Manual PDB in the SAS+ representation tackled 15 problems, all optimally. These varied performances underscored the challenge posed by Rubik's Cube domain and emphasized the importance of the chosen representation.

The supplementary material provides an extended analysis of the PDDL and SAS+ representations, comparing 12 and 18 action models alongside the DeepCubeA model, using datasets from (Büchner et al. 2022). It offers a detailed exploration of the distinctions between PDDL and SAS+, highlighting their advantages and shortcomings. Additionally, a comprehensive overview of the IPC-2023 classical track results, including the agile and satisficing sub-tracks, is available for a broader perspective.

Planner with Heuristic	PDDL
h^{Blind}	24/30
h^{CEA}	23/30
h^{CG}	30/30
h^{Max}	28/30
$h^{LM-Cost}$	28/30
h^{GC}	27/30
h^{FF}	30/30
$h^{M\&S}$	30/30
$h^{PDB-Sys}$	26/30
DeepCubeA	8/30*

Table 3: Comparative analysis of PDDL and DeepCubeA on the Sokoban domain using IPC-2008 dataset. (*noting that only 8 problems had a grid size close to 10x10, while the remainder exceeded DeepCubeA's grid size limit)

Beyond Rubik's Cube

Building on our comprehensive exploration of the RC, we shift our focus to explore how our insights can be transferred to other challenging puzzle domains. We considered the Sokoban puzzle domain, a strategic game that challenges players to push boxes to designated locations within a confined space. Our analysis, detailed in Table 3, reveals a significant contrast in performance: while Planner with Heuristics combinations excels with PDDL representation, effectively solving a substantial number of Sokoban puzzles, DeepCubeA's success rate is considerably lower. For this analysis, we considered the Sokoban puzzle dataset introduced in IPC-2008, comprising of 30 problems of varying complexity.

Despite DeepCubeA's efficiency in solving the Rubik's Cube, its performance in the Sokoban domain, as detailed in Table 3, reveals a limitation in domain adaptability. With a success rate of 8 out of 30 Sokoban puzzles, this outcome highlights a critical aspect of learning-based approaches: the restrictions in state representation, the challenges in generalizing across diverse problem types, and the need for retraining a new model for each new domain. DeepCubeA supports a grid size of up to `10x10` for Sokoban problem representation[7]. Recent works have explored using Transformer based models to solve Sokoban puzzles, with higher efficiency and fewer search steps than traditional methods (Lehnert et al. 2024). This approach mitigates the limitations of problem representation encountered in RL based models like DeepCubeA. However, it introduces a dependency on high-quality labeled data for fine-tuning, which can be both costly and challenging to obtain.

The comparative analysis with PDDL planners in the Sokoban domain further emphasizes the need for continued research into hybrid models that can leverage the strengths of both learning-based and planning-based approaches for broader domain applicability.

[7]For the Sokoban model available in the DeepCubeA GitHub repository. Nevertheless, one can re-train the model with the required parameters.

6 Discussion and Conclusion

In our research, we compared planning-based and learning-based methods for solving the 3x3x3 RC. We introduced the first PDDL representation for RC and assessed the efficacy of various heuristics across SAS+ and PDDL representations. Our findings suggest that the SAS+ representation is approximately 75% more memory efficient than PDDL, making it the preferred choice among standard representations. However, the best planner configuration achieved only 75% problem-solving with 100% optimality. On the other hand, the learning-based DeepCubeA approach solved all problems optimally using its default 12-action set. Yet, our efforts to train it for an 18-action set were thwarted by out-of-memory/ training errors running over a week. This highlights the challenge with learning-based approaches and adaptability of standard representations like PDDL and SAS+. Moreover, DeepCubeA's solutions lack an explanation about how the solution was found or why it may work.

The adoption of PDDL as a representation has its distinct advantages. It makes complex RC problems more tractable and offers a platform for a clearer explanation of the generated plans. The introduction of ontologies for automated planning, as highlighted by Muppasani et al. (2023), further augments this advantage. Ontologies, with their structured representation of knowledge, can provide a semantic framework that aids in the generation of explanations. This not only enhances the understandability of the plans but also facilitates the integration of domain-specific knowledge, making the explanations more contextually relevant and comprehensive.

Future of Rubik's Cube with AI Planning

Though PDBs that can quickly find optimal solutions to the Rubik's cube have been constructed, such as the PDBs used to find its longest shortest path (Rokicki et al. 2014), these PDBs use domain-specific knowledge based on group theory. As a result, they are limited in their ability to generalize to other domains. On the other hand, this study shows that solving the Rubik's cube with domain-independent approaches while maintaining optimality guarantees is still an open problem. Results show that though learned heuristics are not guaranteed to find a shortest path, they often do so, in practice. Results also show that though traditional planning heuristics have optimality guarantees, they may suffer from slow solving times. Future work can now build on both learning and traditional planning approaches given the custom DeepCubeA representation of the Rubik's cube and our standard representation of the Rubik's cube in PDDL. Such a combination could potentially offer solutions to the RC that are both efficient and accompanied by a comprehensive explanation. Recent advancements introduce a method for training heuristic functions to estimate distances to a set of goal states, instead of having a fixed predetermined goal state, without needing retraining for new goals (Agostinelli, Panta, and Khandelwal 2024). This technique leverages deep reinforcement learning to specify goals more expressively, using first-order logic and answer-set programming.

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Agostinelli, F.; Panta, R.; and Khandelwal, V. 2024. Specifying goals to deep neural networks with answer set programming. In *34th International Conference on Automated Planning and Scheduling*.

Büchner, C.; Ferber, P.; Seipp, J.; and Helmert, M. 2022. A Comparison of Abstraction Heuristics for Rubik's Cube. In *ICAPS 2022 Workshop on Heuristics and Search for Domain-independent Planning*.

Bäckström, C. 1995. Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1): 17–34. Planning and Scheduling.

Drexler, D.; Gnad, D.; Höft, P.; Seipp, J.; Speck, D.; and Ståhlberg, S. 2023. Ragnarok. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Drexler, D.; Seipp, J.; and Speck, D. ???? Odin: A Planner Based on Saturated Transition Cost Partitioning.

Ferber, P.; Katz, M.; Seipp, J.; Sievers, S.; Borrajo, D.; Cenamor, I.; de la Rosa, T.; Fernandez-Rebollo, F.; López, C. L.; Nunez, S.; et al. ???? Hapori Stone Soup.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI'71, 608–620. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artif. Intell.*, 173(5–6): 503–535.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3): 1–63.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–57.

Höft, P.; Speck, D.; and Seipp, J. 2013. Dofri: Planner Abstract. In *Proc. IJCAI*, volume 2357, 2364.

Howey, R.; and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *ICAPS 2003 workshop on "The Competition: Impact, Organization, Evaluation, Benchmarks", Trento, Italy*.

Karpas, E.; and Domshlak, C. 2009. Cost-Optimal Planning with Landmarks. In *IJCAI*, 1728–1733. Pasadena, CA.

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI/IAAI*, 700–705.

Lehnert, L.; Sukhbaatar, S.; Mcvay, P.; Rabbat, M.; and Tian, Y. 2024. Beyond A*: Better Planning with Transformers via Search Dynamics Bootstrapping. *arXiv preprint arXiv:2402.14083*.

McAleer, S.; Agostinelli, F.; Shmakov, A.; and Baldi, P. 2018. Solving the Rubik's Cube with Approximate Policy Iteration. In *International Conference on Learning Representations*.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2): 35–35.

McDermott, D.; Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language. Technical report, Technical Report.

Muppasani, B.; Pallagani, V.; Srivastava, B.; Mutharaju, R.; Huhns, M. N.; and Narayanan, V. 2023. A Planning Ontology to Represent and Exploit Planning Knowledge for Performance Efficiency. *arXiv preprint arXiv:2307.13549*.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Pednault, E. P. D. 1994. ADL and the State-Transition Model of Action. *Journal of Logic and Computation*, 4(5): 467–512.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning.

Rokicki, T.; Kociemba, H.; Davidson, M.; and Dethridge, J. 2014. The diameter of the rubik's cube group is twenty. *siam REVIEW*, 56(4): 645–670.

Russell, S.; and Norvig, P. 2005. AI a modern approach. *Learning*, 2(3): 4.

Seipp, J. 2023. Scorpion 2023. *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated cost partitioning for optimal classical planning. *Journal of Artificial Intelligence Research*, 67: 129–167.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; and Segovia-Aguas, J. 2024. The 2023 International Planning Competition.