J Vis
https://doi.org/10.1007/s12650-026-01111-x

REGULAR PAPER

Bence Dániel Erős · **Roland Kunkli**

sCUBEdiver: a visual analytics tool for discovering optimal 2 × 2 × 2 Rubik's Cube solution sequences

Received: 13 June 2022 / Revised: 22 December 2024 / Accepted: 21 January 2026

Abstract In this paper, we present a visual analytics tool and its user evaluation about the optimal solvability of the 2 × 2 × 2 Rubik's Cube. Information visualization becomes more valuable for games of mental skills because it cannot provide just a single state representation, but the player can overview the whole timeline of a game process with outcomes and compare different strategies interactively. Therefore, we can create new supporting tools for the players for their favorite game activity. Using our dashboard, players can better perceive the distance between their current state and the solution of the 2 × 2 × 2 Rubik's Cube, and they can compare human-centered and computer-generated solution strategies from an information visualization perspective. Our user evaluation shows that this visualization approach can be attractive for the users even when the presented problem has an advanced mathematical background based on abstract group theory.

Keywords 2 × 2 × 2 Rubik's Cube · God's number · Solving algorithm · Visual analytics · Game data visualization

1 Introduction

In this paper, we introduce an information visualization system for the optimal solvability of the 2 × 2 × 2 Rubik's Cube. The Rubik's Cube, this challenging mathematical problem, has become a real popular game and a symbol of knowledge and skill development over the past decades (Anzelak et al. 2006; Valerie et al. 2020; Meinz et al. 2023). One reason for this popularity is that players eagerly try to find any puzzle solution in the shortest possible time. Based on this achievement, people can compare their performances, resulting in competitions (Fang et al. 2021). From a broader perspective, every game activity can get players' attention once people get the chance for skill development, and they can feel a competitive spirit while playing the game. Most of the games have their tournaments, a space for the players to demonstrate their problem-solving abilities.

Next to skill development and competitions, research suggests that a successful game needs a massive bibliography also. This bunch of information allows people to get instructions while they develop their performance (Griffiths 2000). While the deep study of human skill development and understanding people's motivation in playing and competitions remains mainly for psychological research (Deci et al. 1981;

B. D. Erős (✉)
Doctoral School of Informatics, University of Debrecen, Kassai út 26, Debrecen 4028, Hungary
E-mail: eros.bence@inf.unideb.hu

B. D. Erős · R. Kunkli
Faculty of Informatics, University of Debrecen, Kassai út 26, Debrecen 4028, Hungary
E-mail: kunkli.roland@inf.unideb.hu

Published online: 03 March 2026

Harackiewicz and Elliot 1993), there are questions for computer science about this above-mentioned bibliography part. Decades earlier, this bibliography meant books and magazines, but today, Internet technologies have enlarged the opportunities for sharing knowledge: in addition to the sources of information that usually display text and still images, players can also find other options that include moving pictures or allowing interaction. Research about innovative teaching practices by Martinez (2011) reports young people who use information technology to improve their performance regarding their favorite game activities. Based on this report, for example, there is a correlation between the level of expert knowledge of football, basketball, or even baseball statistics and the usage of sources like sport websites, TV broadcasts, and video game sport league simulations. Another case study from the same research describes students who could successfully learn to solve the Rubik's Cube using contents from the YouTube online video sharing platform. In these videos, someone demonstrates a solution algorithm step by step and shares tips for dealing with special cases of the puzzle.

Information visualization and infographics can help the gaming community members achieve new exploration, explanation, and communication approaches. People usually cannot meet with visualizations as standalone sources but as part of TV broadcasts, websites, print media products, or analytical software systems (Perin et al. 2018). Regarding these approaches, we can discover a trend where information visualization gains attraction because it cannot provide a single state representation only, but the player can overview the whole evolution of a game process with successful or even unsuccessful outcomes (Lu et al. 2014; Wallner and Kriglstein 2014). Other visualization techniques provide a comparison of different solution strategies by displaying not just one but multiple trajectories of steps in the abstract problem space (Hinterreiter et al. 2021). Therefore, the players who are keen to wonder about solution techniques that look more efficient than the strategies of other players can draw up a question: *Can we use information visualization to find the optimal solution strategy for a game activity?*

Our motivation is to implement an application for investigating this question in more detail as we present the optimal solution of the $2 \times 2 \times 2$ Rubik's Cube. With a system like this, we can enrich the technology-based bibliography of this puzzle and study toward a better understanding of the supportive effect of computer graphics for human problem-solving during epistemic activities. With this exploration, we aim to expand our understanding of the relationship between technology and creativity.

For this purpose, we review previous research on the concept of *God's number* as an upper bound, which defines the maximum number of moves required to solve the Rubik's Cube from any scrambled position. We also present datasets that model the twists of the puzzle, and we assess our experiments to calculate the optimal solution. In addition, we have implemented two solution algorithms for comparison, the $2 \times 2 \times 2$ *Beginner's method* and the *Ortega method*. With the comparison procedures, we would like to assist players in discovering the most effective solution that goes beyond their pre-learned strategies and traditional algorithms while they can unlock the full potential of creative thinking. The new contributions of this article are:

1. A visual analytics system that can help the users to discover optimal solution sequences for the $2 \times 2 \times 2$ Rubik's Cube.
2. Visual tools that facilitate the quantitative analysis of the high-dimensional and the 2D game state spaces, enabling the evaluation of dimensional-reduction effects on the puzzle's state space.
3. A user evaluation reflecting the understandability of our visualizations and the tool's general usefulness.

The paper is organized as follows: in Sect. 2, we explain the most important building blocks and properties of the Rubik's Cube and the mathematical background of the puzzle. Section 3 is about related work. Later, the overview of our system is shown in Sect. 4, and Sect. 5 introduces the methodology and the main steps for creating our dataset. In Sect. 6, we present the panels of our dashboard and detail the interaction possibilities throughout the analysis process with our system. The user evaluation is discussed in Sect. 7. Section 8 concludes this paper with a summary of the results, and it mentions our plans for future work.

2 The design and the mathematical background of the Rubik's Cube

2.1 Design and characteristics

The standard form of the Rubik's Cube is a $3 \times 3 \times 3$ array of little cubes. This simple form and the mechanism of this toy have inspired of today's various spatial puzzles. We refer the reader to papers by Sun and Zheng (2015) and Zeng et al. (2018) for more details of this diversity because here, we focus only on the design of the $2 \times 2 \times 2$ version of the standard form, called the *Pocket Cube*.

The Pocket Cube contains eight so-called *cubies*—following the system of concepts by Frey and Singmaster (1982)—as a $2 \times 2 \times 2$ array of little cubes. Each face of the puzzle is divided into a 2×2 array of *facelets*, resulting in a number of 24 stickers on the surface of the toy. Regarding the cubies, all of them are *corner cubies*. They have three colored facelets placed at the eight corners of the puzzle. A $2 \times 2 \times 1$ group of the cubies results in a layer. The Pocket Cube has a top layer and a bottom layer. During a rotation sequence, we can move all of the twenty-four facelets.

2.2 The Singmaster notation

To solve any given shuffle, the player has to perform rotations to restore the facelets to the solution state. This easily understandable goal of the puzzle becomes difficult because of the size of its state space, where we can generate $4.3 \cdot 10^{19}$ different shuffles in the original form. Even the $2 \times 2 \times 2$ Rubik's Cube has 3 674 160 possible states despite its reduced dimensions. To navigate between states, the players can rotate any face of the puzzle, where they can perform a rotation about the spatial axes both in a clockwise or a counterclockwise direction. In this work, we define a move based on the *quarter-turn metrics*, where a twist means a rotation by 90°, in opposition with *half-turn metrics* and its rotations by 180°. To refer to any move, we can use the name convention created by Singmaster (1981). This widely used notation system uses the initial letters from the name of the faces, where *U* stands for Up face, *R* for Right, *F* for Front, and similarly, *D* for Down, *L* for Left, and *B* for Back. Once the face is rotating in the counterclockwise direction, the uppercase letter appears in a lowercase form. Some work uses an apostrophe or another mark as an exponent with the same uppercase form in this case, such as *R* instead of *r*. The reader can find such examples in works by Kunkle and Cooperman (2009) and Agostinelli et al. (2019). In this work, we use the lowercase form.

2.3 Reducing state-space complexity of the Pocket Cube

Players who would like to fix the orientation of the $2 \times 2 \times 2$ Rubik's Cube in space usually select one of the corner cubies and regard its orientation as a reference point. Fixing the orientation of the Pocket Cube significantly reduces the complexity of the state space. Without a fixed orientation, the state space would expand to 88 179 840 states because the cube itself can be oriented in 24 different ways in space for each of the 3 674 160 valid permutations. These 24 orientations are essentially equivalent, adding unnecessary complexity without providing additional unique information. By fixing the orientation, we avoid counting these redundant states, thereby reducing the state space to 3 674 160 unique states.

2.4 Related mathematical concepts and God's number

Regarding the rotations from a mathematical perspective, players can define these operators as a permutation g: $Z_n \rightarrow Z_n$ by a $2 \times n$ array. Here, n represents the number of facelets that change positions after performing a rotation. For a twist on the Pocket Cube, n is always 12. The *cycle notation* allows creating a compact notation for these permutations (Joyner 2008). Based on the indexing of Fig. 1, we present the mathematical meaning of the rotations as permutations in Table 1.

The mathematical background of the puzzle is related to *group theory*, where researchers searched for the diameter of the Rubik's Cube group for a long time. This value, commonly known as *God's number*, is the maximum number of rotations required to find the solution state starting from any initial shuffle. For the $3 \times 3 \times 3$ Rubik's Cube, Rokicki et al. (2014) published that God's number is 20, regarding the half-turn metrics-associated *Cayley graph*. (The Cayley graph is a graphical interpretation of a group (Joyner 2008), and in this context, God's number refers to the diameter of this graph). At the time of this result, the quarter-turn metric had only an anticipated number (Rokicki 2014), but soon after, the correct value was published

Table 1 The letters of the Singmaster notation and its cyclic permutations

Counterclockwise quarter-turn rotations	Clockwise quarter-turn rotations
u ↔ (0 2 3 1) (4 6 8 10) (5 7 9 11)	U ↔ (0 1 3 2) (4 10 8 6) (5 11 9 7)
r ↔ (1 7 21 18) (3 15 23 10) (8 16 17 9)	R ↔ (1 18 21 7) (3 10 23 15) (8 9 17 16)
f ↔ (2 13 21 8) (3 5 20 16) (6 14 15 7)	F ↔ (2 8 21 13) (3 16 20 5) (6 7 15 14)

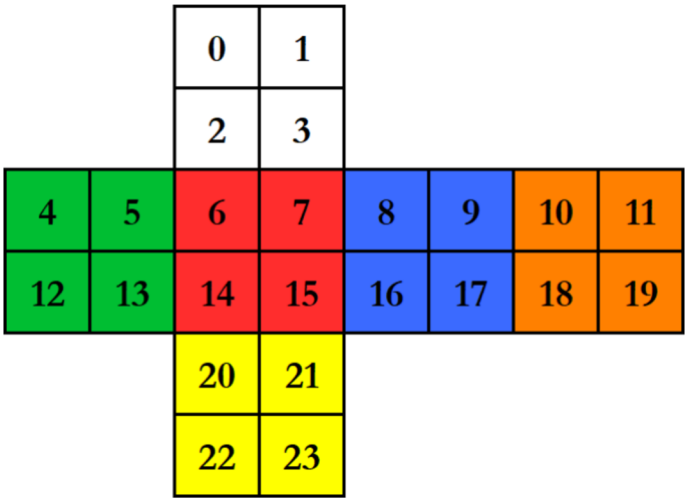

Fig. 1 Indices of the 2 × 2 × 2 Rubik's Cube facelets. The unfolded net of the puzzle is a usual representation for the rearrangement of the facelets after any rotation

online as 26 (Rokicki and Davidson 2014). The size of the 2 × 2 × 2 Rubik's Cube group allows us to generate its whole Cayley graph. Therefore, we know that 11 rotations are enough to solve any shuffle if we use half-turn metrics. In quarter-turn metrics, the required solution path does not need to contain more than 14 rotations (Cooperman et al. 1991).

3 Related work

In this section, we study the problem sets about games of skill where previously published visual analytics systems have provided new approaches. After that, the reader can review the different computer-aided solution techniques of the Rubik's Cube with their advantages and disadvantages.

3.1 Visual analytics of games of skill

As people can meet many games of skill, visual analytics reflects this variety with its very different approaches. For better understandability, this section focuses only on interactive state-space representations of mental games, which may seem to be the most proper category for Rubik's Cube analysis. We refer the readers to other studies for more details of excluded topics, like sport-related research (Perin et al. 2018; Du and Yuan 2021), or skill development via video gameplay analysis, including cases where the experiments focus on players' behavior rather than just interactive state-space analysis (Wallner and Kriglstein 2013; Nguyen et al. 2015; Agarwal et al. 2020).

From the category we are interested in, Rudolph-Lilith (2019) presents *ChessY*, a package for the *Mathematica* software. The package provides a graphical analysis of records of chess games. Its input is a PGN (Portable Game Notation) file, a compact representation of a chess game for computational parsing. The package can create a chess graph from this input with an 8 × 8 layout and a linear layout for explorative purposes. Here, the nodes of the graph are board squares with arguments like occupied by a white piece, occupied by a black piece, or the square is empty. Edges representing potentially performable legal moves of pieces.

Lu et al. (2014) implemented a chess visualization system with a global-to-local viewpoint. Their system contains an evolution graph, a score chart, and a chessboard view. In this case, the graph's nodes represent the whole chess position and not just single squares from the chessboard. Therefore, the player can easily recognize the game's progression, with branches that seem dangerous as they lead to the capturing of the king, which would mean the end of the game. The system also includes a score chart showing potential advantages through move sequences. Lastly, as nodes aggregate chess position into a marker symbol, a chessboard view references the piece placement, allowing exploration and realizing strategies for the player.

Sugiyama et al. (2003) have developed a visual layout that represents permutation puzzles and cyclic puzzles as planar graphs. One of the showcased graphs was based on the $2 \times 2 \times 2$ Rubik's Cube. Their graph portrays individual facelets as small circles, akin to nodes in a graph. Two nodes in this graph are connected if one facelet can be moved to the position of another through the execution of a valid 90° rotation. This mentioned graph can also be viewed as a visual expression of the cycle notation of permutations mentioned in Table 1.

In their work, Hinterreiter et al. (2021) present a different approach than graph–based visualizations: they use the concept of time curves. These trajectories display projection images of high-dimensional data points, and they construct interpolation curves through these points. The authors use this general approach to explore similarities between many game processes, as they analyze the embedded patterns in state spaces from various domains, including many games of skill, like chess and the $3 \times 3 \times 3$ Rubik's Cube. Their visualization allows users to compare different solution algorithms for the $3 \times 3 \times 3$ Rubik's Cube based on the length of time curves.

Our visualization system aims to present solution strategies for the $2 \times 2 \times 2$ Rubik's Cube, including the optimal solution. With our analytics system, the user can compare different solution strategies and identify the length of the shortest paths between puzzle shuffles.

3.2 Computer-aided solution strategies of the Rubik's Cube

We can use the computer to generate whole sequences of rotations and find the solution of the Rubik's Cube. For this purpose, several different techniques have been reported over the last decades.

Some algorithms relate to specific knowledge in artificial intelligence (Korf 1982; Fiat et al. 1989) and machine learning, including deep reinforcement learning and Q-learning (Li et al. 2019; Agostinelli et al. 2019; Lyu et al. 2022). Other experiments rely on graph theory (Khemani et al. 2019) or optimization algorithms from applied mathematics like evolutionary algorithms (El-Sourani et al. 2010) or simulated annealing (Saeidi 2018). Currently, these kinds of techniques loosely reflect a strategy that a human player would prefer. Reviewing previous papers, the reason for this can be apparent: the scientific research of the Rubik's Cube was mainly motivated by the searching for the exact number of required steps to determine the optimal solution length (Kunkle and Cooperman 2009; Rokicki et al. 2014; Rokicki 2014). In these papers, the visual guidance of the Rubik's Cube players seems not examined in such detail.

While collaborative machine learning seems a promising direction in the future (Agostinelli et al. 2021; Lakkaraju et al. 2022), a recent user evaluation involving 10 participants for a system employing an embodied AI-driven chatbot reveals that users' perceptions of the chatbot's utility, ease of use, and user's satisfaction remain at a moderate level, indicating room for improvement in the user interface (Wu et al. 2022). The few other published guidance system includes the work by Park and Park (2014). They present a system to help to solve the $2 \times 2 \times 2$ Rubik's Cube using augmented reality (AR) technology. The authors report that their system provides the steps of the optimal solution for the player; however, there are no details about the algorithms that calculated the solution path, and there is no user evaluation for feedback about usefulness. Later, the two authors reported their experiments examining smartphone-based head-mounted displays (Park and Park 2016). Another paper by Ajisaka et al. (2020) proposes a learning system displaying user operation history. The user can manipulate a virtual $3 \times 3 \times 3$ Rubik's Cube via gesture recognition. Meanwhile, the system enables the player to look back to the operation history, and they can restart the game process from any previous step. Based on the user study of their research, the review information can efficiently help users learn and achieve a higher completion rate. In summary, we can mention that these systems display conventional forms of the Rubik's Cube, similar to many representations in books and websites. These representations are projection images of the Rubik's Cube or unfolded nets of the puzzle.

In contrast, Hinterreiter et al. (2021) have applied abstract visualization techniques first in the domain of computer-aided solution strategies of the Rubik's Cube. They transform permutations into marker symbols

and rotations into trajectories via the t-SNE dimension reduction algorithm. Therefore, players can compare solution techniques and decide which one seems more efficient. In their work, the authors used a solver API from GitHub (Liberacki and Brannan 2015) that can simulate a human player using the so-called Fridrich method for the $3 \times 3 \times 3$ Rubik's Cube; additionally, the authors implemented the Beginner's method. This strategy, similar to the other one, reflects human problem-solving approaches. However, their visualization does not provide an exploration of projection errors, as Munz-Körner and Weiskopf (2024) mentioned, and there is no information about the optimal solvability of Rubik's Cube. In our work, we created a guidance system to the $2 \times 2 \times 2$ Rubik's Cube, where the user can select the optimal solution strategy. A player can find difficulties in performing the steps of the optimal solution on their own, so our system aims to display the required information in visual form. This visualization includes abstract representations, like trajectories of solution sequences and traditional guidances, as the animated 3D model of the Pocket Cube also appears. Since accurately assessing distances is critical for the shortest paths, our dashboard includes distance matrix-based visualizations where the projection errors can be evaluated.

4 Overview

We were motivated to create a system that would help us present the concept of God's number in a visual form. We have designed our solution to create a new opportunity for the community with this contribution: a player with basic knowledge of the $2 \times 2 \times 2$ Rubik's Cube, who may have never heard about God's number, can start their learning process, while this approach would inspire expert players as well for additional exploration. Our system is available online, so a comprehensive range of players can access it through a simple web browser.

4.1 Technical details

The basis of our system is a Python 3 backend. We created the architecture of our application with the Flask framework of Python (Ronacher 2010). This web framework is easy to learn and is one of the most modern and up-to-date ways to develop a web application. We use Microsoft's Azure App Service to deploy our application code. Our front end was developed using HTML5 and JavaScript technologies. We have implemented an application that uses components from D3.js by Bostock et al. (2011). Additionally, we have used the three.js library by Cabello (2010) to display the 3D model of the Pocket Cube. The system has two main parts: a dashboard and an optimal solver page. Our dashboard provides an interactive visualization context with strategies for managing efficient information retrieval. One such management tool is filtering the solution algorithms of the puzzle, allowing it to show specific sequences or leave them out from the screen during comparison. The dashboard panels are listed in detail in Sect. 6. Using the second page of the application, the user can color a Pocket Cube to calculate the optimal solution for any correct shuffle of the puzzle and watch the solution using an animation.

For optimal performance, the recommended setting of the screen resolution is 1920×1080. This version of the application has only been tested with the web browsers Google Chrome and Mozilla Firefox. At the time of writing, users can access the application at https://rubikvisapp.azurewebsites.net/.

4.2 Database and other data sources

The application works with data from the Pocket Cube. This set includes the following information:

- *Puzzle states* Individual shuffles that describe the color and the orientation of the corner cubies after the player performs rotations. This data has a string type. Here, we used the precalculated list of valid Pocket Cube permutations by Malone (2016).
- *Solution sequences* The sequence of puzzle states from the initial shuffle to the solved puzzle. Our system can generate three solution sequences using the $2 \times 2 \times 2$ Beginner's algorithm, the Ortega method, and the optimal solver. It is an array of strings.
- *Distance* The length of the minimum required rotation sequences from one puzzle state to another one. Integer value.

Regarding the Pocket Cube, we can generate 3 674 160 possible puzzle states. Here, measuring the distance between all possible node pairs would result in a distance matrix of size 3 674 160 $\times$ 3 674 160. To

reduce computational costs, we selected only 14 from the possible permutations, as they seem enough to represent our visual ideas and help us provide an easy-to-follow learning curve for our users. We randomly selected these puzzle states whose distance is 1, 2, 3,..., 14 from the solution state.

Using these 14 puzzle states as initial shuffles, we generated 42 solution sequences with the three different strategies. Additionally, we have used three different dimension reduction algorithms so the user can change the layout for the solution sequences. We stored these sequences using Azure's PostgreSQL relational database cloud service. Here, our first table contains individual puzzle states with an id and a string representation of the current shuffle. These records also have a column for the distance of the puzzle permutation from the solution. A second table contains information to identify the steps of solution sequences. Moreover, we created CSV files to store distance matrices based on the solution sequences.

5 Modeling

This section describes modeling Rubik's Cube shuffles and their rotations in our computer program before the visualization steps. We reveal the details of our implementations of the solution algorithms with particular attention to the optimal strategy, which provides the shortest paths from the current state to the solved state.

5.1 Representation of $2 \times 2 \times 2$ Rubik's Cube states and rotations

As players prefer using unfolded cube nets to overview the effect of rotations, we can adopt this methodology with some modifications. First, each facelet is assigned its own index number. Based on this index, we can arrange the facelets into a single row, represented as a string or array in the code. For example, each element of the string can represent the color of a facelet. For the $2 \times 2 \times 2$ Rubik's Cube, this string will have 24 characters. We define each character based on the initial letter of the color. For instance, if the facelet with index 0 is white, we store an uppercase *W* as the first character. Once we have defined our variable as a string with multiple characters, permutation rules can be applied to determine the rearrangement of these characters. In this way, we can model any sequence of rotations.

5.2 Implementing solution algorithms

The Pocket Cube has several strategies that allow the player to restore the facelets to their original positions. We include two well-known algorithms into our system: the $2 \times 2 \times 2$ Beginner's and Ortega methods. The main difference is that they define different subgoals:

- Beginner's method:
  - Solve the bottom layer of the cube:
    - Fix the green–yellow–orange cubie.
    - Solve the green–yellow–red cubie.
    - Solve the yellow–blue–red cubie.
    - Solve the yellow–blue–orange cubie.
  - Solve the white face of the cube.
  - Solve the top layer.
- Ortega method:
  - Solve the yellow face of the cube.
  - Solve the white face of the cube.
  - Solve the rest of the cube (permutation of both layers).

In our system, we created a JavaScript implementation of these algorithms. We have the following limitations: our Ortega method implementation follows the principle that it always restores the yellow facelets first due to the fixed orientation of the puzzle in our data storage method. However, in practice, this algorithm starts with checking every side and identifying the face with the most facelets with the same color because the players should start with that one, as this can save extra steps for them most efficiently. For this reason, our implementation can contain unnecessary steps compared to the possibilities of a physical

solution. The fixed orientation of the puzzle can also affect the Beginner's method, as in practice, we can rotate the $2 \times 2 \times 2$ Rubik's Cube much more freely, and we should not have to keep in mind the order of the faces but recognize colors and patterns about the position of the facelets. Therefore, the reader should consider the paths given by the solving algorithms accordingly; their suggestion may not always be the same as if the players had started solving a physical puzzle on their own.

The explanation of these restrictions involves the simplified problem space by fixing the puzzle's orientation in space, as discussed in Sub Sect. 2.3: allowing the entire cube to be rotated freely in space does not create more possible shuffles. We still have 3 674 160 valid permutations, but unnecessarily multiplying this by 24 results in 88 179 840 states. Importantly, 24 of these are always equivalent to an existing state. Thus, no additional information is gained, only a more complex state space. This complexity can be managed by fixing the cube's orientation in space. A direct consequence of this is that the mentioned solving algorithms become somewhat simplified compared to their real-life scenarios. For example, when the Beginner's method instructs to solve the bottom layer, there is no choice as to which side is the bottom; it is fixed. This means we lose the ability to dynamically select the side with the most correct colors as the bottom. However, the advantage is that we do not need to check if a given state is equivalent to another, as the fixed orientation inherently resolves this issue.

5.3 Calculating optimal solution sequences

We have modeled the puzzle as a graph problem to find the optimal solution for the Pocket Cube. Khemani et al. (2019) reported a similar mapping. In our model, the nodes are the possible shuffles. The edges represent their connection based on rotations *U, R, F, u, r, f*.

We have created an implementation of the BFS (Breath-First Search) algorithm. BFS helps find solutions at a specific depth level in the spanning tree and guarantees a shortest path from the root (initial shuffle) to a particular node (solved puzzle) in the problem space. We have also created a data model using the Neo4j (2020) graph database management system. Neo4j includes the function called *shortestPath*. Using this function, we can return the path between two puzzle shuffles with the fewest number of rotations. If more than one shortest path exists, the function picks one nondeterministically. We compared the output of the database queries and the output of our BFS implementation to check the correctness of the optimal solution length.

5.4 Generating distance matrices

A distance matrix represents how the length of the shortest paths varies during a rotation in different solution sequences (see Fig. 2). We have used this information in many visualizations. While both BFS and Neo4j queries can find a single optimal solution sequence from any initial position, we have faced increasing performance time when calculating multiple shortest paths. To construct a distance matrix, we changed our method to GPU-based calculation.

We have used the NVIDIA Graph Analytics library (nvGRAPH 2019), which contains graph algorithms optimized for the GPU. From this library, we selected the *nvgraphSssp* function that implements a single-source shortest path algorithm. The method can calculate the length of the shortest paths from one single source to all other vertices of the graph. The *nvgraphSssp* function uses the adjacency matrix of the problem space in CSC (Compressed Sparse Column) format as input. We stored our data in three separate text files to build such input: *source indices*, *destination offset*, and *weights*.

- An element in the *source indices* file identifies the source vertex for an edge. This file has e entries, where e equals the number of edges in the graph—the value of e is $6 \times 3 \times 674 \times 160$ for the Pocket Cube.
- The file *destination offset* contains $(n + 1)$ entries, where n is the number of nodes in the graph. In this file, one entry equals the index of the first incoming edge for this vertex based on the list of *source indices*. The $(n + 1)$st entry equals e, storing the number of total edges.
- We set each element to 1 in the *weights* file because we observed no difference between the edges. Regardless of which side of the puzzle we rotate, it does not incur additional costs (e.g., it does not take more time). The file *weights* has e entries, similar to *source indices*.

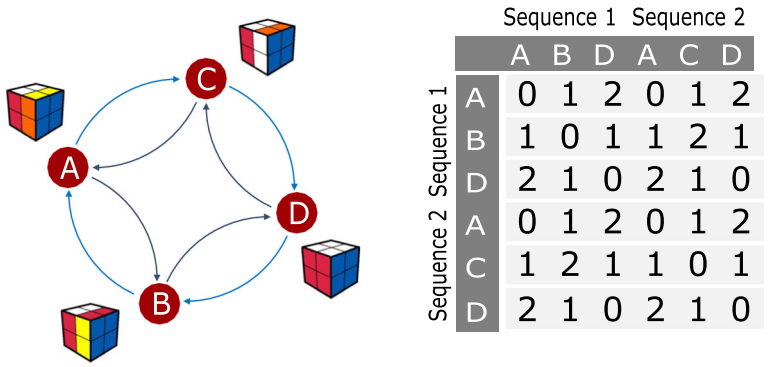

Fig. 2 Generate a distance matrix based on the game graph of the Pocket Cube. The example graph shows two possible solution sequences from node A to node D, rotating only the right side of the puzzle (performing *R* and *r*). The first (darker blue, inner shape) is ABD, and the second (lighter blue, outer circle) is ACD. The distance matrix summarizes the number of required rotations between shuffles

Then, we can use nvGRAPH to perform the shortest path calculations. Using this library, we can generate multiple distance matrices regardless of the length of the solution sequences. We store our matrices in CSV files.

5.5 Projection of solution sequences

According to Hinterreiter et al. (2021), we can visualize the progression of a mental game (or an algorithm) through different states as a trajectory in 2D space. They applied their technique to visualize high-dimensional processes from the $3 \times 3 \times 3$ Rubik's Cube, the chess, and the neural network training domain.

Before we can calculate the coordinates of the projected states (the steps of the progress), we have to represent the high-dimensional states as vectors. The proper representation can influence the meaning of any potential distance metric. This information can be crucial because a distance metric can help us preserve a specific characteristic of the high-dimensional game space when we embed the states of a solution sequence in 2D space. Hinterreiter et al. (2021) represent a Rubik's Cube state as a $6 \times (3 \times 3) \times 6$ tensor, where one face of the Rubik's Cube is a 3×3 matrix for each of the six faces. Moreover, they use a one-hot encoding representing the color of a facelet (e.g., (0, 0, 0, 0, 0, 1) for red). They chose Euclidean distance, the square root of the Hamming distance, as their metric. They found that Euclidean distance does not consider the number of applied rotations for their representation but gives a measure of the number of cube facelets with the wrong color. Then, they applied t-distributed stochastic neighbor embedding (t-SNE) for projection.

Our work aims to preserve information about the number of applied rotations in the high-dimensional space to present the shortest paths truthfully. We used the rows of distance matrices to represent the states of a solution sequence. Let us consider Fig. 2. If we want to create a trajectory for *Sequence 1*, we can represent node *A* with vector (0, 1, 2). The value 0 is the number of applied rotations from node *A* to reach the same state. Value 1 shows one rotation from node *A* to node *B*, and value 2 shows two rotations from node *A* to node *D*. Similarly, in *Sequence 1*, we can represent node *B* with vector (1, 0, 1). Our method can change the vector's length depending on the solution sequences' length. Therefore, when we want to project the nodes *A*, *B*, *C*, and *D* into one common 2D space, we can add the length of the solution sequences together so we can use vector (0, 1, 2, 0, 1, 2) for our first node because both *Sequence 1* and *Sequence 2* have three nodes. In this case, node *B* is vector (1, 0, 1, 1, 2, 1).

We have used the above vector representation and applied three different dimension reduction techniques for projection: multi-dimensional scaling (MDS), isometric mapping (ISOMAP), and t-SNE. We chose them to test their different properties: MDS and ISOMAP can be better for uncovering global structures of data. In contrast, t-SNE can reveal local features of the high-dimensional data space. MDS has the property that focuses on pairwise dissimilarities between data elements. Its goal is to locate similar data close together in the lower dimension and to put data far away when the elements are less similar. To calculate these pairwise distances, MDS uses Euclidean distance. ISOMAP is an extension of MDS; it determines pairwise geodesic distances between data elements. Therefore, it can handle nonlinear manifolds better than MDS (Burch et al. 2020; Anowar et al. 2021). We have used the *networkx* library by Hagberg

et al. (2008) and its open-source component *Graphviz* with the *neato* engine to create our layout. This tool is a variation of MDS (Gansner 2012). We performed t-SNE and ISOMAP by calling the proper functions from the *scikit-learn* library by Pedregosa et al. (2011). We saved the layout results into text files and loaded the coordinates into the application's database.

5.6 Evaluation of the accuracy of projections

Since the projection of high-dimensional data to 2D can create information loss, we usually cannot map the distances of the *n*-dimensional space to 2D correctly. Therefore, the applied approximations result in error (Munz-Körner and Weiskopf 2024). We have created datasets that help the players inspect projections and explore these errors more thoroughly. For this, we have calculated the pairwise Euclidean distances between the steps on our 2D trajectories and summarized these distances in a matrix form. Cutura et al. (2020) have introduced matrix visualization to compare and explore dimensionality reduction algorithms. They found that a matrix visualization can support data analysts in comparison tasks like comparing how differences between data points in high-dimensional space differ from their 2D projection or comparing different dimensionality reduction algorithms to each other.

In our work, we have constructed three different types of matrices: the distance matrices, which summarize distances in high-dimensional space; the distance matrices, which measure the Euclidean distances between data points on 2D trajectories after the dimension reduction process; and our discrepancy matrices to show the differences between the previous ones. Before calculating the differences, we transformed the Euclidean distances to discrete values between one and fourteen. This way, we can directly subtract the first matrix from the second one. Tables 2, 3 and 4 contain some statistics about the discrepancy matrices:

- The first column shows the number of test cases from one to fourteen. This number indicates the length of the corresponding optimal solution sequences.
- The second column is the dimension of the discrepancy matrix for the specific test case. The number of rows/columns includes one Beginner's solution path length, one Ortega path length, and one optimal solution sequence length.
- The third column is the *SUM*, where we summarize the values in the cells of the discrepancy matrix. A higher value means the projection algorithms generate higher dissimilarities between the high-dimensional distances and the distances from the 2D trajectories. For example, we experienced some cases where the two shuffles are neighbors, but they appear far away in the 2D drawing area. A case like this can increase the discrepancy.
- Other statistical values are the average of the dissimilarities, the median, the minimum value in the discrepancy matrix, and the maximum.

6 Visualizations

This section presents our visual analytics system, which was created with the previously detailed datasets. In our work, we have selected tools from the D3.js and the three.js libraries for implementing our dashboard and the user interface for the optimal solver. When the users open the dashboard, they can see five main visual analysis tools, presented in Fig. 3.

6.1 Trajectory visualization

The trajectory visualization (see panel A in Fig. 3) contains visual encoding for the three solution algorithms, representing the sequence of rotations from the initial state to the solution in their sequential order. Moreover, the user can gain insight into the distances in the high-dimensional space as we have applied dimension reduction algorithms to project the puzzle permutations into the 2D space. The visualization includes brownish circles in the drawing area, which are abstract forms of shuffles. To decode the meaning of a circle, the user can click on the item. Then, the visual Pocket Cube model will change its color to show the selected permutation. To change one shuffle into another, the user can choose one of the neighbor elements of the selected item. In this case, the virtual Pocket Cube does not simply change its color but performs an animation to show the completed rotation. We have connected the points to represent these rotations with an interpolating curve. A path can show whether the shuffles belong to the same solution

Table 2 Evaluation of the MDS dimensional-reduction algorithm for the fourteen discrepancy matrices

MDS						
	SIZE	SUM	AVG	MDN	MIN	MAX
1	6 × 6	234	6.5	6.5	0	13
2	9 × 9	450	5.5556	5	0	12
3	56 × 56	4640	1.4796	1	0	6
4	15 × 15	864	3.84	3	0	10
5	126 × 126	49,968	3.1474	3	0	12
6	91 × 91	24,358	2.9414	4	0	10
7	144 × 144	80,116	3.8636	2	0	12
8	79 × 79	14,370	2.3025	3	0	9
9	148 × 148	81,082	3.7017	3	0	13
10	117 × 117	45,508	3.3244	3	0	11
11	144 × 144	76,884	3.7078	3	0	12
12	113 × 113	41,310	3.2352	3	0	12
13	136 × 136	66,992	3.6220	3	0	11
14	91 × 91	24,496	2.9581	3	0	11

Table 3 Evaluation of the t-SNE algorithm for the fourteen discrepancy matrices

t-SNE						
	SIZE	SUM	AVG	MDN	MIN	MAX
1	6 × 6	234	6.5	6.5	0	13
2	9 × 9	594	7.3333	9	0	13
3	56 × 56	5108	1.4375	1	0	5
4	15 × 15	1350	6	6	0	12
5	126 × 126	56,154	3.5370	4	0	10
6	91 × 91	25,430	3.0709	3	0	10
7	144 × 144	79,284	3.8235	4	0	10
8	79 × 79	14,824	2.3753	2	0	8
9	148 × 148	79,294	3.6201	4	0	11
10	117 × 117	42,498	3.1045	3	0	9
11	144 × 144	83,516	4.0276	4	0	10
12	113 × 113	41,792	3.2729	3	0	10
13	136 × 136	57,562	3.1121	3	0	9
14	91 × 91	25,824	3.1185	3	0	8

Table 4 Evaluation of the ISOMAP dimensional-reduction algorithm for the fourteen discrepancy matrices

ISOMAP						
	SIZE	SUM	AVG	MDN	MIN	MAX
1	6 × 6	234	6.5	6.5	0	13
2	9 × 9	612	7.5556	11	0	12
3	56 × 56	4466	1.4241	1	0	5
4	15 × 15	1386	6.16	6	0	10
5	126 × 126	54,996	3.4641	3	0	12
6	91 × 91	19,502	2.3550	2	0	10
7	144 × 144	74,640	3.5995	3	0	11
8	79 × 79	12,932	2.0721	2	0	8
9	148 × 148	88,260	4.0294	4	0	13
10	117 × 117	43,018	3.1425	3	0	10
11	144 × 144	73,158	3.5281	3	0	10
12	113 × 113	37,570	2.9423	3	0	8
13	136 × 136	54,006	2.9199	3	0	10
14	91 × 91	21,704	2.6209	2	0	8

algorithm: the curve of the Beginner's method is purple, the path element encoding the Ortega method is green, and there is a yellow path representing an optimal solution sequence. Users can see the animation only if they select an element from the purple path on the basic setting. They can switch to the other algorithms using the radio buttons (F4) in panel F.

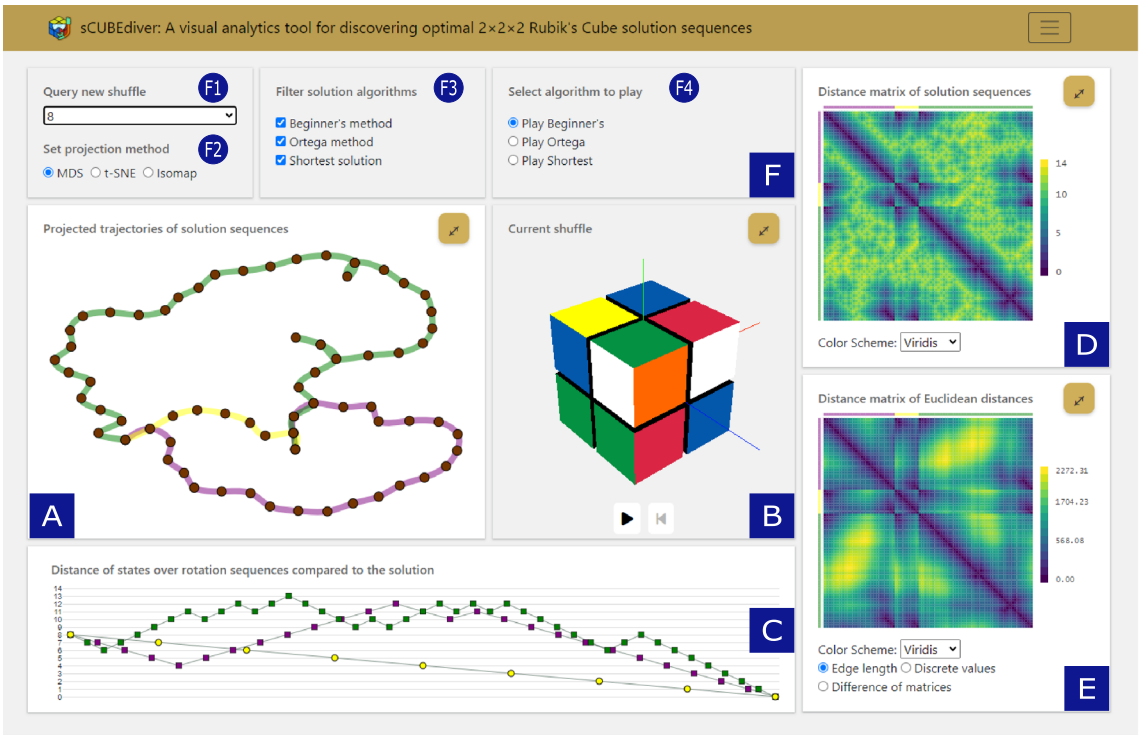

Fig. 3 The dashboard interface of our application contains a trajectory layout (**A**), a virtual Pocket Cube (**B**), a line chart (**C**), and two distance matrices (**D** and **E**). The user can find panels for settings (**F**), including a dropdown menu (**F1**) to show a new shuffled puzzle, radio buttons (**F2** and **F4**) to change the dimension reduction algorithm in the trajectory layout and to select a solution sequence for animation, and checkboxes (**F3**) to filter solution algorithms

6.2 Distance matrix of the high-dimensional game space

We have constructed a distance matrix based on the solution sequences from the Beginner's method, the Ortega method, and the optimal solution. The rows and columns of this matrix list the steps of the sequences one after another. The matrix cells contain the distances between the puzzle permutations in the high-dimensional space, which are the same as the number of the applied rotations to reach one state from another on the shortest paths. The matrix appears in panel D (see Fig. 3). The coloring of the matrix reflects the number values of its cells. Every color value is determined by the perceptually-uniform color schemes designed by van der Walt and Smith (2015). A color scaling bar also appears on the right side of the screen to help people understand the meaning of the colors. The users can open a dropdown list and select from four different schemes, namely *viridis*, *magma*, *inferno*, and *plasma*. This visualization can support users in exploring the high-dimensional game space in more detail and how the length of the shortest paths varies during the rotations in the different solution sequences. The user can hover their cursor to see the exact cell value.

6.3 Distance matrix of the 2D projections

The second distance matrix (see panel E in Fig. 3) measures Euclidean distances between the points on the trajectory visualization. The steps of the solution sequences appear as the rows and the columns of the matrix, following the same encoding method as the upper distance matrix. The coloring of the matrix cells reflects the measured Euclidean distances. The matrix uses the same colormap as the other one, but the user can also change the default property here. This visualization can help users explore distances in the 2D drawing space (see Fig. 4), and they can find different matrix views to compare the distances in the high-dimensional game space with the 2D space.

They can evaluate the dimension reduction algorithms in this way (see Fig. 5). For comparison, a *Discrete values* button under the matrix changes the data from a continuous interval into discrete values between zero and fourteen. The users can compare these discrete values directly to the values of the upper

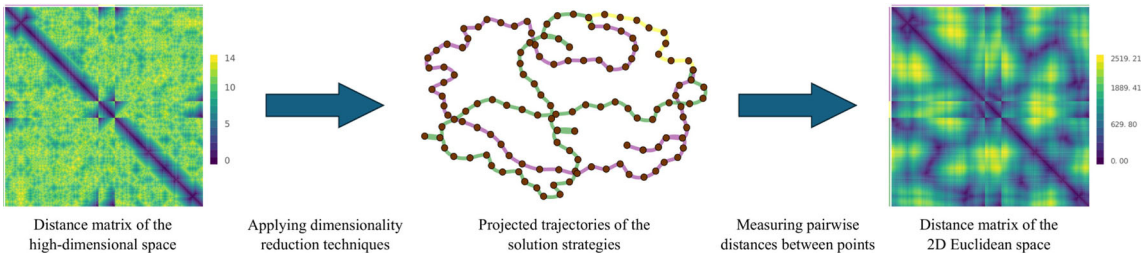

Fig. 4 Since the application of dimensionality reduction algorithms can result in information loss, a second matrix provides an overview of the distances between individual states within the 2D drawing area. Here, we illustrate how the distance matrix of the 2D Euclidean space is derived for test case number 11 using the MDS dimensionality reduction technique

matrix. When the user changes to the *Difference of matrices* view, they can generate a discrepancy matrix, where the matrix cells show the differences between the high-dimensional and the 2D space.

6.4 Line chart

The line chart (see panel C in Fig. 3) presents the three solution sequences composed on one joint x- and y-axis. The x-axis represents a continuous value, time, or the process of the solution. The y-axis is categorial. The values from zero to fourteen show the minimal steps required to reach the solution. Here, one puzzle shuffle appears as a marker symbol. We have used the same color coding on the trajectory visualization: purple for the Beginner's method, green encodes Ortega, and yellow shows the steps of an optimal solution sequence. Regarding the optimal solution, the shape of the marker symbols also differs from those used in other algorithms' encoding to provide an easier way to observe visual elements. This visualization can support users to explore the pattern of fluctuation when they apply a non-optimal solution algorithm. Therefore, they can quickly identify an optimal solution sequence as a straight descending line segment for any case.

6.5 Virtual Rubik's Cube and the net of the cube

The 3D model of the $2 \times 2 \times 2$ Rubik's Cube appears on the dashboard (panel B in Fig. 3) to help decode the initial shuffle. We created the mesh with tools from the WebGL-based library, three.js. The object is rotatable with the cursor, so the users can check every side of the virtual Rubik's Cube when they want to see the color of the facelets. The three coordinate axes also appear to help recognize the order of the faces. The virtual cube can simulate rotations when the users select two circles on the trajectory visualization next to each other. The panel of the mesh contains a play button. When the user clicks this button, they can play the animation of the whole solution processing, including not just one but all rotations of the selected solution method. The second button under the virtual puzzle can help users to jump back to the initial permutation.

A second virtual Pocket Cube appears on the optimal solver page (see Fig. 6), where the user can put in any possible permutation. Here, they can click and color the squares of a cube net. Next to the 2D representation of the cube, the 3D model helps users visualize their input, and later, they can use this virtual puzzle to watch the animation of the optimal solution for their initial shuffle.

6.6 User interactions

In our work, we have connected the dashboard panels and implemented the following interaction techniques to help users gain information more efficiently (see Fig. 3 to identify the mentioned panels):

1. In panel (A), when hovering the cursor over a circle, the color of the marker symbol changes to red, and the circle's radius increases. At the same time, in panels. (D) and (E), the colors of the rows in the matrix visualization fade away, except for the row that encodes the selected puzzle shuffle. This technique supports identifying the steps of the solution sequences on the different data views.
2. When clicking on the selected circle in panel (A), the 3D cube mesh in panel (B) changes its color, and the user can decode the proper permutation. When clicking on another circle next to the previous one, they can watch a short animation about transforming one puzzle state into another.

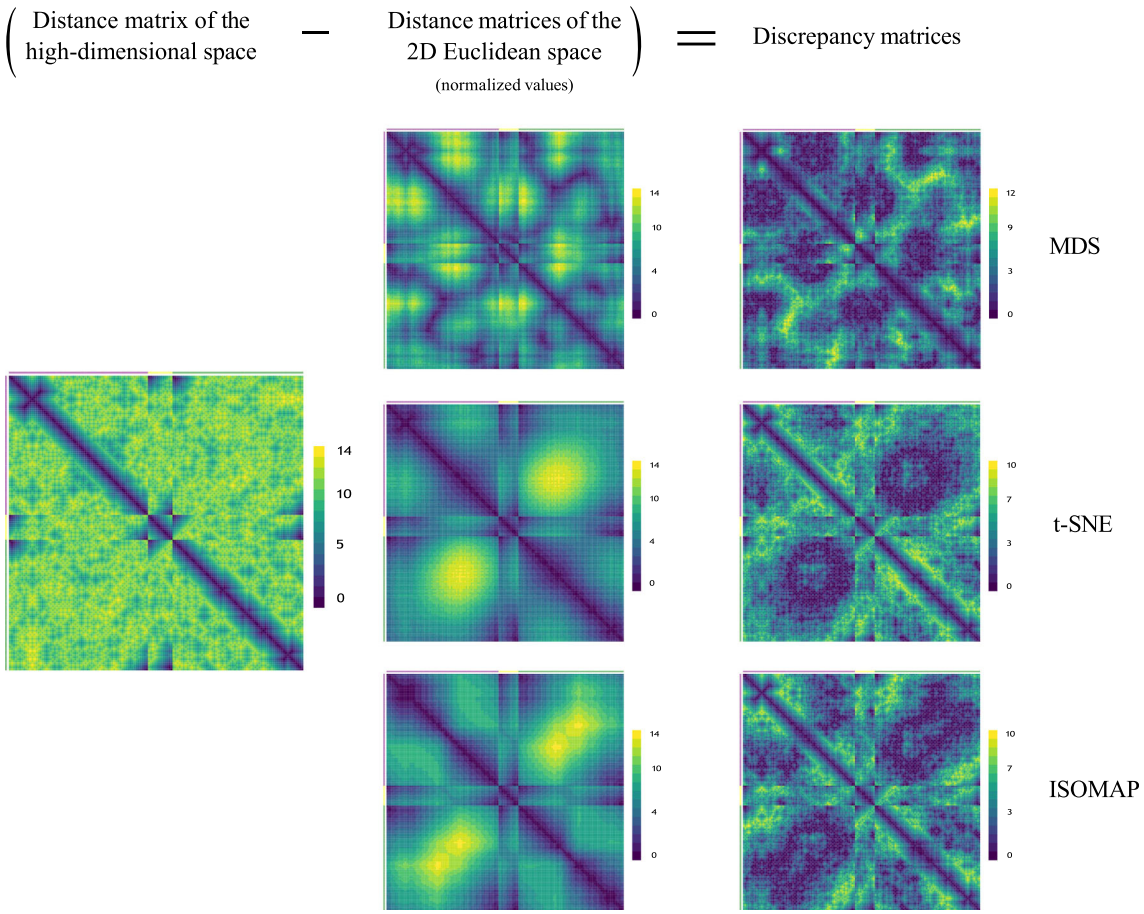

Fig. 5 Discrepancy matrices for the analysis of dimensionality reduction techniques. These matrices illustrate the discrepancies between pairwise distances in the high-dimensional state space of the Pocket Cube and its 2D projection. Dark blue areas indicate minimal discrepancies, while yellow regions indicate significant deviations, enabling the assessment of dimensionality reduction methods in different regions of the problem space

3. In panel (B), two buttons are available: one for playing the animation of the complete solution sequence and a second button to step back to the initial shuffle. Panel (F) contains radio buttons (F4) to select the proper algorithm to play.
4. The panels (D) and (E) are the distance matrices. Here, when the user hovers their cursor over a cell of any matrix, they can see a tooltip showing the cell's value. Moreover, highlighting two shuffles on the trajectory visualization (one by the matrix row and the second by the column) is possible. These elements appear in red in panel (A).
5. Both panels (D) and (E) contain a dropdown menu for changing the default color scheme. Panel (E) has three additional radio buttons to generate a discrepancy matrix view for efficient comparison.
6. Panels (A), (B), (D), and (E) have a colored button in their right upper corner. These buttons allow resizing the charts. The increased panels fit 50% of the entire screen. Therefore, generating a view where focusing on the two most significant information sources is possible.

7. In panel (F), there are additional user interface elements:

    - The dropdown menu (F1) allows selecting a new test case from the fourteen precalculated test cases.
    - The radio buttons under the dropdown menu (F2) allow the players to choose from different dimension reduction algorithms: MDS, t-SNE, or ISOMAP. When the user changes the dimension reduction algorithm, the system redraws the trajectory in panel (A) and the distance matrix of panel (E).

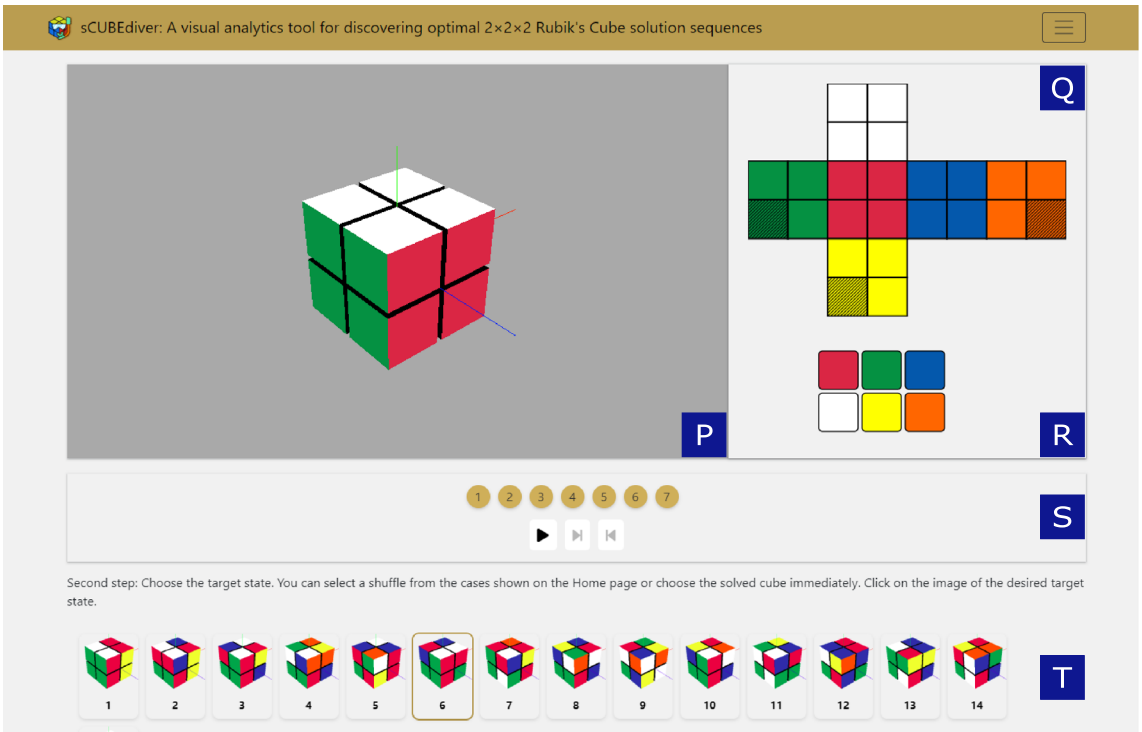

Fig. 6 The optimal solver interface of the application. The user can put in any possible permutation of the Pocket Cube using the net of the cube (**Q**) by clicking on a square and defining its color. The faces of the fixed corner cubie appear in a darker shade. The color palette (**R**) helps to select the proper color for the facelets. The 3D puzzle model (**P**) helps the users recognize their input during the coloring process. Before searching for the optimal solution, the users can select one from the precalculated test cases as output in panel (**T**). Later, they can watch the steps of the solution sequence by pressing the play button in panel (**S**)

- Using the checkboxes (F3), filtering the three solution algorithms is possible. The unchecked elements will disappear from panels (A), (C), (D), and (E).
- The users can select a node from the different paths in panel (A) after they switch the algorithm using the radio buttons (F4). These elements also change the animation to play for panel (B).

8. The dashboard can be changed to an optimal solver view (see Fig. 6), where a 3D cube and a 2D cube net are visible. When the user clicks on a square on the cube net, they can change the color of the virtual cube. Therefore, they can set any possible permutation as an input and find the optimal solution for their shuffle.

7 User evaluation

In order to examine users' thoughts on our system, we conducted a survey. Participants were asked to test the application and answer some questions based on this visual information. The participants received a description of the essential information about the purpose of the research and the circumstances of the study, along with a link to the application, before they started. During the survey, they provided their age and subjectively rated their experience with the Rubik's Cube. They also answered two task-specific questions and filled out a user experience questionnaire, allowing us to monitor their impressions in more detail.

A total of 110 participants answered the questions. The language of the survey was Hungarian because every participant spoke Hungarian as their native language. Their ages ranged from 19 to 38. Based on a 7-point Likert scale, most participants responded that they have relatively little game experience compared to expert knowledge. Still, we have found players at every level of our scale (see Fig. 7).

We collected the answers in two different ways: the first group accessed the application and completed the survey from home using their own devices. Alongside the application link, they were provided with an

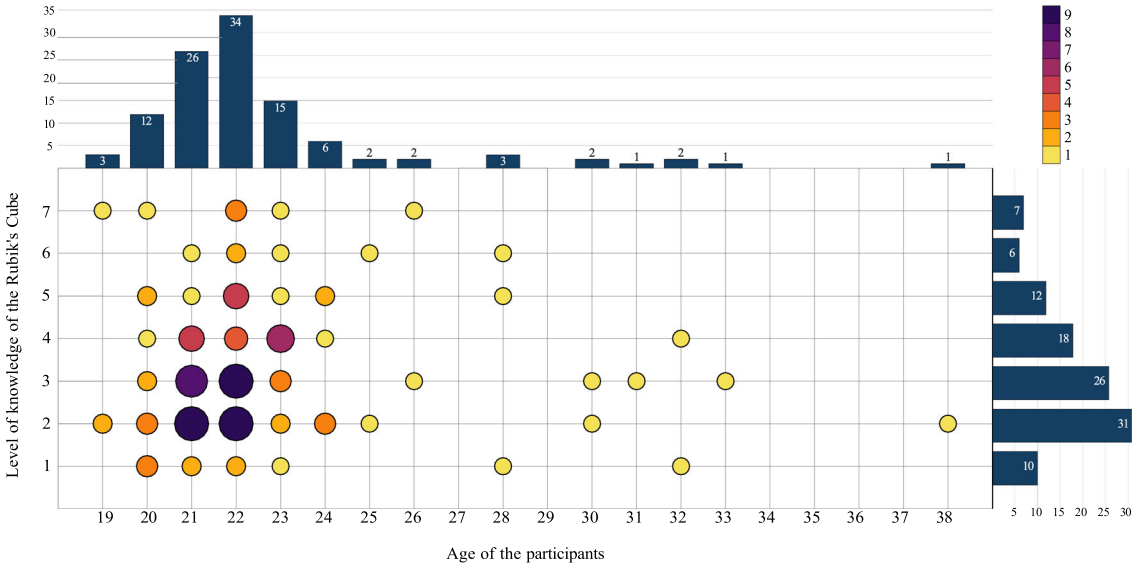

Fig. 7 As part of our user study, the participants provided their ages and self-assessed their level of knowledge regarding the Rubik's Cube. The horizontal axis represents the ages of the players, ranging from 19 to 38. Along the vertical axis, the "Level of knowledge of the Rubik's Cube" extends from 1 to 7. Here, a value of 1 indicates that players have never attempted to solve the Rubik's Cube, while a value of 7 suggests that players actively study the toy and can confidently apply various solution techniques

additional link to a narrated video explaining the application's most important features. Users could optionally use this video, as we did not monitor whether they watched it before using and evaluating the analytics system. The second group consisted of participants we met in person, who accessed the application and completed the survey in a supervised environment. For this group, the narrated video was replaced by a live presentation. Here, participants could also express their opinions orally after filling out the form. Twenty-two participants (20%) took part in the evaluation remotely, and eighty-eight participants (80%) attended the in-person sessions.

During the questionnaire series, users encountered two such questions where they had to solve practical problems related to determining a shortest path. The first question was about whether participants could reconstruct a scenario shown in a screenshot from the dashboard while using the application and determine the length of an optimal solution sequence. Here, we asked the user to type the corresponding number if they answered "Yes", or type "No" if they answered otherwise. With another question, we measured the users' confidence level while they were identifying rotations using our system. We formalized this question into a simple decision; therefore, the participants could answer "Yes" or "No" based on their perception.

On the first task, 59 participants (53.64%) answered correctly. Three participants (2.73%) answered "Yes", but they did not send a numerical solution. Fifteen answers (13.64%) proved incorrect as they missed the correct answer by one compared to the optimal solution length. Another 9 participants (8.18%) answered an incorrect number with a difference of more than one. The other 24 users (21.82%) answered "No" and they did not send any numerical solution. Based on these results, our system can provide enough information for tasks based on quantitative values (e.g., length of a solution path) for the players regardless of their skill level. Meanwhile, some users find abstract visualizations unusual and struggle to decode their meaning. The next question shows 85 "Yes" answers (77.27%) for the question of whether they can determine the required rotation and solve their Pocket Cube optimally using our system. Here, the users can use the virtual cube and the animation, which can be more usual for them. See the distribution of the answers in summary in Fig. 8.

The last part of the survey was a user experience questionnaire. The user experience is a highly subjective impression, but it greatly impacts a software's lifetime. Our motivation here is to measure if our idea is welcomed as a visual analytics system or if players feel that the currently available bibliography can provide all the necessary information for them in this subject. We used the User Experience Questionnaire (UEQ) items by Schrepp et al. (2017) during this part. We find advantages of the UEQ as it is a standardized measuring tool and contains a detailed Excel feature for efficient benchmarking. A disadvantage is that it

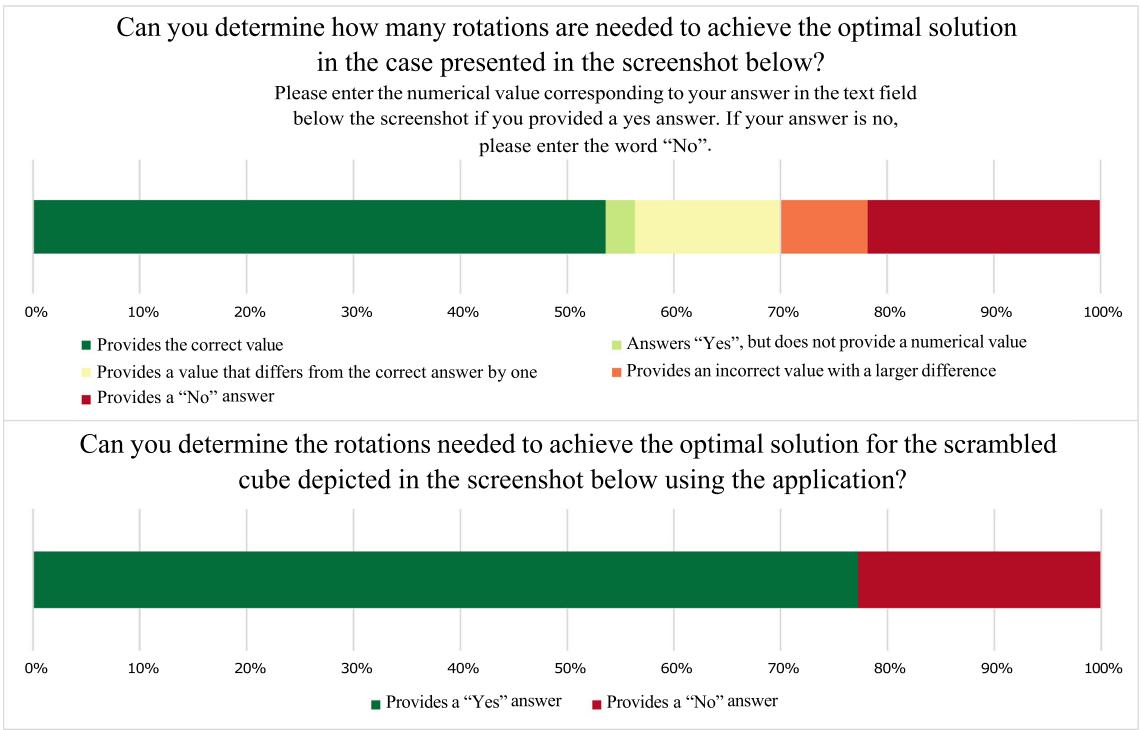

Fig. 8 The results of the task from the user evaluation survey. The participants answered two questions related with the optimal solution of the Pocket Cube

does not distinguish between different software categories, i.e., there is only one benchmark for business applications, household appliances, or even games.

We used the standard version of the UEQ, which contains 26 items. Every item is an opposite pair of terms on a 7-point Likert scale. Despite the number of items, users can express their impressions quickly. There is a range from -3 to 3, measuring whether a participant fully agrees with the negative term or fully agrees with the positive one. After we collected the rating values from the participants, the benchmark tool supported us to create five different scales for evaluation based on the 26 items:

- *attractiveness*: measuring overall impression
- *perspicuity*: measuring whether the tool is easy to learn and understand
- *efficiency*: measuring the effort saving of the users to solving their task
- *dependability*: measuring control over interaction
- *stimulation*: measuring whether the tool is fun and it motivates the users
- *novelty*: measuring whether the tool captures users' attention

The results of the user experience study are in Fig. 9. In this figure, we can see two bar charts for every scale to interpret the means. The darker gray bars show the impression of the remote participants. The lighter gray bars represent the participants who attended a live presentation before the evaluation. Values above 0.8 indicate *positive evaluations*. In our study, the highest mean belongs to *dependability* in both groups (1.511 and 1.713). Only one scale shows a neutral evaluation: *perspicuity* from the remote participants (0.455). Here, we measured an increased mean regarding the live sessions, where participants evaluated *perspicuity* positively (0.884). In our study, we have not measured negative evaluation. In the same figure, we represent confidence intervals as black line segments. The confidence intervals measure the precision of the estimated scale means. A smaller confidence interval means a higher precision, where we can trust more in our results (see Table 5).

We also welcomed any comments after the user evaluation, including features for the future. For example, one participant expressed their interest in searching God's number and shared their opinion on our system. They mentioned that it may be easier for mathematicians to understand aspects of the optimal solution, whereas it is challenging for non-specialists to interpret this concept. However, they believed that

Table 5 The 5% confidence intervals (CI) for the scale means of the user experience study, where n represents the sample size

Scale	Remote users(n = 22)		Live session users(n = 88)	
	Mean	CI	Mean	CI
Attractiveness	1.295	[0.820, 1.771]	1.316	[1.138, 1.494]
Perspicuity	0.455	[− 0.117, 1.026]	0.884	[0.643, 1.124]
Efficiency	1.284	[0.780, 1.788]	1.528	[1.327, 1.730]
Dependability	1.511	[1.049, 1.973]	1.713	[1.534, 1.892]
Stimulation	1.250	[0.764, 1.736]	1.170	[0.953, 1.388]
Novelty	1.045	[0.559, 1.532]	1.014	[0.795, 1.234]

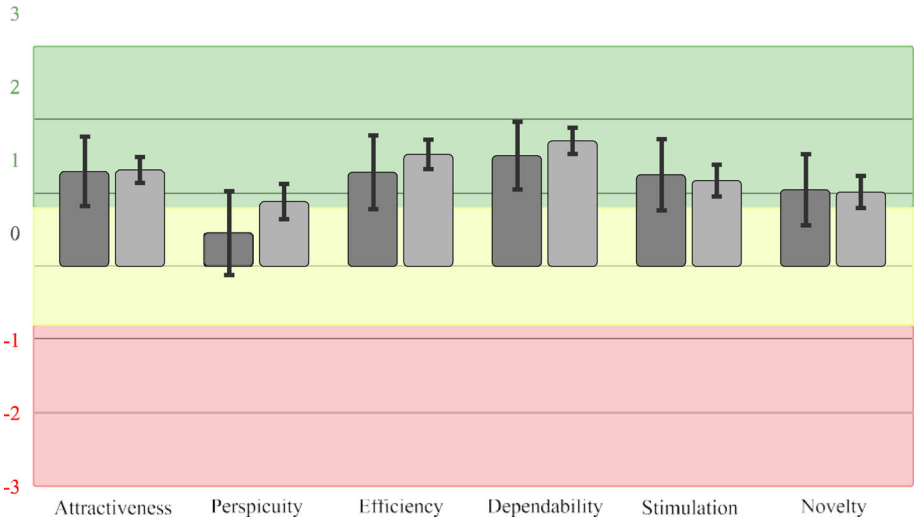

Fig. 9 The results of the user experience study for evaluating our visual analytics system. The darker gray bars depict the responses of remote participants, while the lighter gray bars represent those of participants who attended a live presentation before the evaluation. See Table 5 for additional details

the presented visualization approach may still be suitable for non-specialists. Another user expressed that they find interpreting many aspects of the system problematic and find it may be better for specialists.

For future features, many comments (12 participants, 10.9%) suggested that having more control over the speed of the cube animation in the next versions of the application would be useful. Other respondents (7 participants, 6.36%) mentioned that they would like to see more information integrated into the application, including infoboxes or a tutorial page explaining how to use the dashboard. Here, we encountered comments suggesting the addition of a tutorial feature to learn about the Beginner's method and other basic solution techniques. Some feedback (4 participants, 3.63%) focused on the design of the dashboard. In summary, they found the user interface to be rather conservative, noting the lack of unique color usage and style elements in the settings panel. They also mentioned that they would welcome a responsive design in the future, as well as full support for mobile devices.

Other ideas include thoughts on whether the application could allow users to scan the sides of a physical Rubik's Cube to recognize its current state. Some participants also expressed their interest in a visual analytics system to present God's number for the $3 \times 3 \times 3$ Rubik's Cube.

8 Conclusion

In this work, we have presented a visual analytics tool that assists users in exploring optimal solution sequences of the $2 \times 2 \times 2$ Rubik's Cube. Our application contains a dashboard with different dataset views, where users can overview different solution algorithms and predict the solution's distance. Our results also include an optimal solver implementation, where the players can input their shuffle and determine the optimal solution for the puzzle. The players can follow the required rotation instructions using an animated virtual model. Our dashboard helps users examine the differences between human-centered and

computer-generated strategies. Our motivation is to enlarge the currently available tools for this toy and help players improve their problem-solving skills while examining the supporting effect of information visualization in epistemic challenges. Through this study, we aim to visually convey the primarily mathematical concept of God's number to a broader audience, inspiring their interest in new perspectives and fostering creative problem-solving. To measure the effect of our system, we conducted a user study with a user experience questionnaire. This study shows that our visualization approach can be attractive regardless of the advanced mathematical background of the given problem for presenting the optimal solution of the Pocket Cube. However, people can need help to gain information from abstract visualizations. Therefore, some users found it hard to interpret the mathematical concept. Despite their frustration with this complexity, users generally felt that the dashboard *meets expectations*, and they evaluated the system as *good, efficient, inventive*, and *interesting* (see *Dependability, Attractiveness, Efficiency, Novelty*, and *Stimulation* in Fig. 9).

For future work, we plan to implement different visualizations based on the same concept of God's number. As our current system is designed directly to the $2 \times 2 \times 2$ Rubik's Cube, we would like to study how we can present the concept of God's number for the original $3 \times 3 \times 3$ puzzle visually. Currently, the value of God's number for higher dimensions ($4 \times 4 \times 4$, $5 \times 5 \times 5$, etc.) is still undetermined; we find an interesting question whether a visual approach could assist researchers in determining these values in the future.

We are also interested in changing our data generation method to a sensory-based approach. This approach could be beneficial for higher dimensions, as it enables real-time analysis. Specifically for the Pocket Cube, such an approach has the potential to offer valuable insights into various aspects, such as orientation adjustment of the puzzle and its implications. For the research community, this could provide an opportunity to understand how altering the initial hand positioning of the puzzle affects the Beginner's method (or the Ortega strategy), as well as to explore changes in the number of common states between a shortest path and the solution sequences of the Beginner's (Ortega) method relative to the initial orientation. This analysis could shed light on the impact of the initial puzzle orientation on solving strategies.

Acknowledgements The authors are grateful to all the respondents who participated in the user study.

Authors' contribution Conceptualization, BDE and RK; methodology, BDE and RK; software, BDE; validation, BDE and RK; formal analysis, BDE; investigation, BDE and RK; writing—original draft preparation, BDE; writing—review and editing, BDE and RK; visualization, BDE and RK; supervision, RK.

Funding Open access funding provided by University of Debrecen. Openaccess funding provided by University of Debrecen. No funding was received for conducting this study.

Data availability The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no competing interests.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Agarwal S, Wallner G, Beck F (2020) Bombalytics: visualization of competition and collaboration strategies of players in a bomb laying game. Comput Graph Forum 39(3):89–100. https://doi.org/10.1111/cgf.13965

Agostinelli F, McAleer S, Shmakov A et al (2019) Solving the Rubik's cube with deep reinforcement learning and search. Nat Mach Intell 1(8):356–363. https://doi.org/10.1038/s42256-019-0070-z

Agostinelli F, Mavalankar M, Khandelwal V, et al (2021) Designing children's new learning partner: collaborative artificial intelligence for learning to solve the Rubik's cube. In: IDC 21 interaction design and children. Association for computing machinery, pp 610–614 https://doi.org/10.1145/3459990.3465175
Ajisaka S, Hara S, Matsuchi M, et al (2020) Learning Rubik's cube through user operation history. In: 2020 Nicograph international (NicoInt). IEEE, pp 43–46 https://doi.org/10.1109/NicoInt50878.2020.00015
Anowar F, Sadaoui S, Selim B (2021) Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE. Comput Sci Rev 40(1):100-378:1. https://doi.org/10.1016/j.cosrev.2021.100378
Anzelak M, Frankl G, Ebner W (2006) Building up knowledge like a Rubik Cube. In: Feh´er P (ed) 7th European conference on knowledge management 2006 (ECKM 2006), vol 1. Academic Conferences Ltd., pp 10–18
Bostock M, Ogievetsky V, Heer J (2011) D^3 data-driven documents. IEEE Trans vis Comput Graph 17(12):2301–2309. https://doi.org/10.1109/TVCG.2011.185
Burch M, Kuipers T, Qian C, et al (2020) Comparing dimensionality reductions for eye movement data. In: VINCI '20: Proceedings of the 13th international symposium on visual information communication and interaction. association for computing machinery, pp 18:1–18:5 https://doi.org/10.1145/3430036.3430049
Cabello R (2010) Threejs–JavaScript 3D library (software), URL https://threejs.org/, [Online; last accessed on 24-April-2024]
Cooperman G, Finkelstein L, Sarawagi N (1991) Applications of Cayley graphs. In: Sakata S (ed) AAECC 1990: Applied algebra, algebraic algorithms and error-correcting codes. Springer Berlin Heidelberg, pp 367–378, https://doi.org/10.1007/3-540-54195-0_65
Cutura R, Aupetit M, Fekete JD, et al (2020) Comparing and exploring high- dimensional data with dimensionality reduction algorithms and matrix visualizations. In: AVI '20: Proceedings of the international conference on advanced visual interfaces. Association for computing machinery, https://doi.org/10.1145/3399715.3399875
Deci EL, Betley G, Kahle J et al (1981) When trying to win: competition and intrinsic motivation. Personal Soc Psychol Bull 7(1):79–83. https://doi.org/10.1177/014616728171012
Du M, Yuan X (2021) A survey of competitive sports data visualization and visual analysis. J vis 24(1):47–67. https://doi.org/10.1007/s12650-020-00687-2
El-Sourani N, Hauke S, Borschbach M (2010) An evolutionary approach for solving the Rubik's Cube incorporating exact methods. Applications of evolutionary computation. Springer, Berlin Heidelberg, pp 80–89. https://doi.org/10.1007/978-3-642-12239-2_9
Fang C, Zhang E, Zhang J (2021) Do women give up competing more easily? Evidence from speedcubers. Econ Lett 205(1):109943:1-109943:5. https://doi.org/10.1016/j.econlet.2021.109943
Fiat A, Moses S, Shamir A, et al (1989) Planning and learning in permutation groups. In: Proceedings of the 30th annual symposium on foundations of computer science. IEEE Computer Society, pp 274–279 https://doi.org/10.1109/SFCS.1989.63490
Frey AH Jr, Singmaster D (1982) Handbook of Cubik math. Enslow Publishers, Hillside, New Jersey, United States
Gansner ER (2012) Drawing graphs with Graphviz, technical report, AT&T Bell laboratories, Murray, URL https://www.graphviz.org/pdf/oldlibguide.pdf, [Online; last accessed on 24 April 2024]
Griffiths M (2000) The psychology of games. Psychol Rev 7(2):24–26
Hagberg A, Swart P, Chult DS (2008) Exploring network structure, dynamics, and function using NetworkX, No. LA-UR-08-05495; LA-UR-08-5495, Los Alamos National Lab (LANL), Los Alamos, NM (United States), URL https://www.osti.gov/biblio/960616, [Online; last accessed on 24 April 2024]
Harackiewicz JM, Elliot AJ (1993) Achievement goals and intrinsic motivation. J Pers Soc Psychol 65(5):904–915. https://doi.org/10.1037/0022-3514.65.5.904
Hinterreiter A, Steinparz C, Schöfl M et al (2021) Projection path explorer: exploring visual patterns in projected decision-making paths. ACM Trans Interact Intell Syst 11(3–4):1–29. https://doi.org/10.1145/3387165
Joyner D (2008) Adventures in group theory: Rubik's Cube, merlin's machine, and other mathematical toys, 2nd edn. The John Hopkins University Press, Baltimore, Maryland, United States
Khemani C, Doshi J, Duseja J, et al (2019) Solving Rubik's Cube using graph theory. In: Computational intelligence: theories, applications and future directions, volume I. Advances in intelligent systems and computing, vol 798. Springer Singapore, pp 301–317 https://doi.org/10.1007/978-981-13-1132-1_24
Korf RE (1982) A program that learns to solve Rubik's Cube. In: AAAI'82: Proceedings of the second AAAI conference on artificial intelligence. AAAI Press, pp 164–167
Kunkle D, Cooperman G (2009) Harnessing parallel disks to solve Rubik's cube. J Symb Comput 44(7):872–890. https://doi.org/10.1016/j.jsc.2008.04.013
Lakkaraju K, Hassan T, Khandelwal V, et al (2022) ALLURE: A multi-modal guided environment for helping children learn to solve a Rubik's Cube with automatic solving and interactive explanations. In: Proceedings of the 36th AAAI conference on artificial intelligence. AAAI Press, pp 13 185–13 187, https://doi.org/10.1609/aaai.v36i11.21722
Li T, Xi W, Fang M et al (2019) Learning to solve a Rubik's Cube with a dexterous hand. In: 2019 IEEE international conference on robotics and biomimetics (ROBIO), pp 1387–1393 https://doi.org/10.1109/ROBIO49542.2019.8961560
Liberacki L, Brannan T (2015) Rubik's Cube solver coded in Python, URL https://gi thub.com/CubeLuke/Rubiks-Cube-Solver, [Online; last accessed on 24 April 2024]
Lu WL, Wang YS, Lin WC (2014) Chess evolution visualization. IEEE Trans vis Comput Graph 20(5):702–713. https://doi.org/10.1109/TVCG.2014.2299803
Lyu Z, Liu Z, Khojandi A et al (2022) Q-learning and traditional methods on solving the pocket Rubik's cube. Comput Ind Eng 171:108452:1-108452:11. https://doi.org/10.1016/j.cie.2022.108452
Malone T (2016) Generating the full list of valid states of a 2 × 2 Rubik's Cube. lime- juice|one guy's essay into arduino, URL https://web.archive.org/web/20240214233054/https://tim.id.au/limejuice/generating-the-full-list-of-valid-states-of-a-2x2-rubiks-cube/, [Online; last accessed on 24 April 2024]

Martinez JE (2011) Technology rich learning environments. A performatory approach to teaching. Learning and Technology Sense Publishers, Rotterdam, The Netherlands, pp 45–58
Meinz EJ, Hambrick DZ, Leach JJ et al (2023) Ability and nonability predictors of real-world skill acquisition: the case of Rubik's cube solving. J Intell 11(1):18. https://doi.org/10.3390/jintelligence11010018
Munz-Körner T, Weiskopf D (2024) Exploring visual quality of multidimensional time series projections. Vis Inform 8(2):27–42. https://doi.org/10.1016/j.visinf.2024.04.004
Neo4j (2020) Neo4j desktop (v1.2.5), URL https://neo4j.com/download-center/, [Online; last accessed on 24 April 2024]
Nguyen TH, El-Nasr MS, Canossa A (2015) Glyph: visualization tool for understanding problem solving strategies in puzzle games. In: Zagal JP, MacCallum-Stewart E, Togelius J (eds) Proceedings of the 10th international conference on the foundations of digital games (FDG 2015). Society for the Advancement of the Science of Digital Games, pp 64:1–64:9
nvGRAPH (2019) nvGRAPH Library User's Guide (DU-08010–001 v10.1), URL https://docs.nvidia.com/cuda/archive/10.1/pdf/nvGRAPH_Library.pdf, [Online; last accessed on 24 April 2024]
Park J, Park C (2014) Guidance system using augmented reality for solving Rubik's Cube. In: Stephanidis C (ed) HCI international 2014—posters' extended abstracts, vol 1. Springer International Publishing, pp 631–635 https://doi.org/10.1007/978-3-319-07857-1_111
Park J, Park C (2016) Augmented reality based guidance for solving Rubik's Cube using HMD. In: Stephanidis C (ed) HCI international 2016—posters' extended abstracts, vol 2. Springer International Publishing, pp 524–529 https://doi.org/10.1007/978-3-319-40542-1_85
Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830
Perin C, Vuillemot R, Stolper CD et al (2018) State of the art of sports data visualization. Comput Graph Forum 37(3):663–686. https://doi.org/10.1111/cgf.13447
Rokicki T (2014) Towards god's number for Rubik's cube in the quarter-turn metric. The Coll Math J 45(4):242–253. https://doi.org/10.4169/college.math.j.45.4.242
Rokicki T, Davidson M (2014) God's number is 26 in the quarter-turn metric. URL http://www.cube20.org/qtm/, [Online; last accessed on 24 April 2024]
Rokicki T, Kociemba H, Davidson M et al (2014) The diameter of the Rubik's cube group is twenty. SIAM Rev 56(4):645–670. https://doi.org/10.1137/140973499
Ronacher A (2010) Flask–the pallets projects (software), URL https://palletsprojects.com/p/flask/, [Online; last accessed on 24 April 2024]
Rudolph-Lilith M (2019) Chessy: a mathematica toolbox for the generation, visualization and analysis of positional chess graphs. SoftwareX 9(1):39–43
Saeidi S (2018) Solving the Rubik's cube using simulated annealing and genetic algorithm. Int J Educ Manag Eng 8(1):1–10. https://doi.org/10.5815/ijeme.2018.01.01
Schrepp M, Hinderks A, Thomaschewski J (2017) Construction of a benchmark for the user experience questionnaire (UEQ). Int J Interact Multimed Artif Intell 4(4):40–44. https://doi.org/10.9781/ijimai.2017.445
Singmaster D (1981) Notes on Rubik's magic cube.In: Lecturer in mathematical sciences and computing polytechnic of the south bank, London
Sugiyama K, Hong SH, Maeda A (2003) The puzzle layout problem. Graph drawing GD 2003 Lecture notes in computer science, vol 2912. Springer, Berlin Heidelberg, pp 500–501. https://doi.org/10.1007/978-3-540-24595-7_50
Sun T, Zheng C (2015) Computational design of twisty joints and puzzles. ACM Trans Graph 34(4):101:1-101:11. https://doi.org/10.1145/2766961
Valerie J, Aylward G, Varma K (2020) I solved it! using the Rubik's Cube to support mental rotation in a middle school science classroom. In: Gresalfi M, Horn IS (eds) The interdisciplinarity of the learning sciences, 14th international conference of the learning sciences (ICLS) 2020, vol 2. International society of the learning sciences., pp 653–656
Wallner G, Kriglstein S (2013) Visualization-based analysis of gameplay data—a review of literature. Entertain Comput 4(3):143–155. https://doi.org/10.1016/j.entcom.2013.02.002
Wallner G, Kriglstein S (2014) PLATO: a visual analytics system for gameplay data. Comput Graph 38:341–356. https://doi.org/10.1016/j.cag.2013.11.010
van der Walt S, Smith N (2015) Matplotlib colormaps, URL https://bids.github.io/colormap/, [Online; last accessed on 24 April 2024]
Wu D, Tang H, Bradley C, et al (2022) AI-driven user interface design for solving a Rubik's cube: a scaffolding design perspective. In: HCI international 2022—late breaking papers. Design, user experience and interaction. Springer International Publishing, pp 490–498 https://doi.org/10.1007/978-3-031-17615-9_34
Zeng DX, Li M, Wang JJ et al (2018) Overview of Rubik's cube and reflections on its application in mechanism. Chin J Mech Eng 37(1):77:1-77:12. https://doi.org/10.1186/s10033-018-0269-7