From Scramble to Solution

Developing an Optimal Solver for
the Kilominx, a Rubik's Cube Variant

Samuel Borg

Supervisor: Ian Gent

March 29, 2025

School of Computer Science
University of St Andrews

Abstract

The Rubik's Cube is the most well-known combination puzzle of all time, and has inspired numerous variants of the puzzle. One such puzzle is the Kilominx, a 2x2 dodecahedron-shaped puzzle with 12 faces and 20 cubies (individual cube elements of the puzzle). While the Rubik's Cube has already been extensively researched, the Kilominx has not.

This report details my work in creating an optimal solver for the Kilominx, guaranteeing solutions in as few moves as possible. A separate program I wrote precomputes several pattern databases, each containing the number of moves required to solve different sets of cubies in all possible states. The solver uses an iterative-deepening A* (IDA*) algorithm to find the optimal solution, using the pattern databases as a lower-bound heuristic. In order to efficiently index into the pattern databases, the solver computes a Lehmer code for the puzzle state, which is then used to calculate the state's sequential index.

In theory, the solver can optimally solve any state given enough time, but in practice it can only find solutions up to a depth of 14 within a reasonable timeframe. As solutions require longer sequences of moves to solve, the solver takes exponentially more time to find the optimal solution. With further optimisation, the solver could be used to help tighten the bounds on the Kilominx's God's Number - the maximum number of moves needed to solve the puzzle from any given state.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is [insert word count] words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Acknowledgements

I would like to thank my supervisor, Ian Gent, for his invaluable guidance, insight and support throughout this project. His expertise and enthusiasm have made this project an incredibly engaging and rewarding journey.

I would also like to thank my parents, Juliet and Peter, my brother, Tom, and my amazing friends Emil, Ben, Zoe, Josh, Maya, Jess and Alfie for their unwavering support and belief in me all the way through my studies.

Contents

1. Introduction

When the Rubik's Cube was globally released in 1980, it became an instant success, selling around 200 million units by the end of 1983 [1]. The Rubik's Cube also quickly became a popular subject of research for computer scientists, in part due to the massive 43 quintillion possible states [2] that the cube can be in. Many different algorithms were designed to try and solve the cube in as few a number of moves as possible, but it wasn't until 1997 that Richard E. Korf published a paper [3] describing a method to solve the Rubik's Cube optimally (the shortest possible number of moves to solve any given cube state) by using large lookup tables called pattern databases [4] as a heuristic function to guide an IDA* search algorithm.

Another key area that researchers focused on had to do with a concept called "God's Number", which is the maximum number of moves needed to solve the cube from any given state. Throughout the '90s and '00s, researchers were able to tighten the bounds on the God's Number for the Rubik's Cube, until in 2010 it was finally proved that the cube's God's Number is exactly 20 [5]. This means that for any optimal solver (such as Korf's algorithm), the solver will always be able to find a solution in 20 moves or less (given enough time to find the solution).

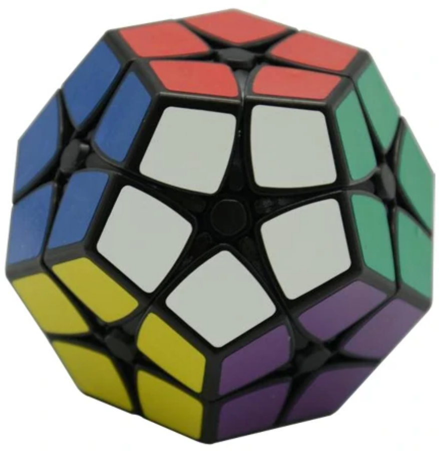

Figure 1: An image of a Kilominx[1].

The widespread success of the Rubik's Cube also led to the creation of a number of different variants of the puzzle, each working similarly to the Rubik's Cube but of different shapes and sizes. The Kilominx is one of these variants, part of the larger "minx" family of dodecahedron-shaped puzzles. The Kilominx is a 2x2 dodecahedral

[1]This image is taken from the "Kilominx" article on the Speed Cube Solving wiki at Fandom and is licensed under the Creative Commons Attribution-Share Alike License.

puzzle with 12 faces and 20 cubies (individual cube-shaped elements of the puzzle). Although some research has been done on tightening the bounds of its God's Number [6], no optimal solver has previously been created for the puzzle.

The main goal for this project was to create an optimal solver for the Kilominx, applying similar principles to Korf's algorithm for the Rubik's Cube - using pattern databases to guide an IDA* search algorithm. An additional goal was to use the solver to help increase the lower bound of Kilominx's God's Number by trying to find an optimal solution of a greater number of moves than the current lower bound.

1.1 Project Objectives

Below is a list of the objectives for this project, which are grouped based on their priority. It should be noted that the priorities of the objectives are slightly different to the original priorities as outlined in the DOER - these were updated after the project was started to better reflect which objectives were the most important for the project.

Primary Objectives:

- Create an optimal solver for the Rubik's Cube, using an IDA* search algorithm and pattern databases (as described in Korf's paper).
- Create an optimal solver for the Kilominx, using the same principles as used in the Rubik's Cube solver.

Secondary Objectives:

- Use the solver to tighten the bounds of the Kilominx's God's Number.
- Create a simple GUI to allow users to interact with a Rubik's Cube and Kilominx, and use the solvers to find optimal solutions for both puzzles.

Tertiary Objectives:

- Survey users about their experience using the GUI and solver to gather feedback on its ease of use.
- Investigate better ways to allow users to input a Kilominx state into the solver (e.g. with computer vision).

2. Context Survey

2.1 Terminology

Cubie - An individual cube-shaped element of the puzzle. The Rubik's Cube has 26 cubies (excluding the non-visible cubie at the centre of the cube), and the Kilominx has 20 cubies (sometimes referred to as "**kubies**" - Kilominx cubies).

Face - A face of the puzzle, which can be twisted to change the state of the puzzle. The Rubik's Cube has 6 faces, while the Kilominx has 12 faces.

Twist - A twist is a rotation of a face of the puzzle. A twist is only valid if the face is aligned with the rest of the puzzle when the twist is completed. Twists are the basic moves that can be made on the puzzle to change its state. The Rubik's Cube has 6 faces, and each face can be twisted in 3 different ways (90 degrees clockwise, 90 degrees counterclockwise, or 180 degrees). The Kilominx has 12 faces, and each face can be twisted in 4 different ways (72 degrees clockwise, 72 degrees counterclockwise, 144 degrees clockwise, or 144 degrees counterclockwise).

Move Notation - Twists on the Rubik's Cube can be represented in a shorthand notation [7], where a letter represents a **clockwise** twist of the face represented by that letter. If the letter is followed by an apostrophe (e.g. U, read as "U Prime"), it represents a **counterclockwise** twist of the face. If the letter is followed by the number 2 (e.g. $U2$), it represents **two clockwise** twists of the face (a 180-degree twist). The letters used to represent the faces of the Rubik's Cube are as follows:

- U - Up
- L - Left
- F - Front
- R - Right
- B - Back
- D - Down

The notation for the Kilominx is similar, but instead uses up to three letters to represent the faces of the puzzle. In addition, to represent a **double counterclockwise** twist, the face is followed by a 2 and an apostrophe (e.g. $U2'$). The letters used to represent the faces of the Kilominx are as follows:

- U - Up
- L - Left

- F - Front
- R - Right
- BL - Back Left
- BR - Back Right
- DL - Down Left
- DR - Down Right
- DBL - Down Back Left
- DBR - Down Back Right
- DB - Down Back
- D - Down

2.2 Korf's Algorithm for the Rubik's Cube

As mentioned briefly in the introduction, Richard E. Korf published a paper in 1997 [3] which described a method to find optimal solutions for the Rubik's Cube, using a combination of an IDA* search algorithm and large memory-based lookup tables called pattern databases, which served as the lower-bound heuristic function for the IDA* search. The paper was a landmark in the field of Rubik's Cube research, as it was the first to find optimal solutions for the cube, and it also introduced a number of important concepts that would be used in later research on the cube.

2.2.1 Representing the Cube State

The Rubik's Cube is made up of 26 cubies (not including the cubie at the centre of the cube). Six of these cubies are centre cubies, which are fixed in place and do not move, while the other cubies are either corner cubies (of which there are 8) or edge cubies (of which there are 12). The corner cubies can be oriented in 3 different ways, while the edge cubies can be oriented in 2 different ways. Korf suggests representing the state of the cube as an array of 20 elements, where each element represents a cubie on the cube, encoding the cubie's initial position (or index) and orientation. While this representation is less intuitive than a more object-oriented approach, it is much more efficient for the IDA* search algorithm, as it allows for faster access and manipulation of the cube's state. In order to make moves on the cube, specific cubies are swapped around in the array, and their orientations are updated accordingly.

2.2.2 Iterative-Deepening A*

In order to find an optimal solution for the cube, a search algorithm with an admissible heuristic function is needed. An admissible heuristic function is one that never overestimates the cost of reaching the goal state from the current state (i.e. the heuristic function is always less than or equal to the actual cost). The A* search algorithm is a well-known search algorithm that uses an admissible heuristic function to guide the search towards the goal state. However, the A* search algorithm has an exponential space complexity of $O(b^d)$, where b is the branching factor and d is the depth of the search tree. This means that the algorithm can quickly run out of memory for large search spaces, such as with the Rubik's Cube.

Instead, Korf uses an IDA* search algorithm, which is a combination of the A* search algorithm and iterative deepening search. IDA* works by performing a depth-first search with a depth limit, using an admissible heuristic to prune branches if their estimated cost exceeds the current bound. After the search space is fully explored, the depth limit is increased and the search is repeated. This allows the algorithm to use less memory than A*, as it only needs to store the current path in memory, rather than the entire search tree. The IDA* search algorithm has a space complexity of $O(bd)$, which is much more manageable for large search spaces.

Korf also notes that when iterating over possible moves in the IDA* search, we can skip over moves that are redundant, where twisting the same face twice in a row is equivalent to just one twist (e.g. $FF2$ is equivalent to F'), and moves that aren't in a specific order: since twists on opposite faces are independent and commutative (e.g. twisting the front face and then the back face is the same as twisting the back face and then the front face), we can define an arbitrary order for each pair of opposite faces, and only consider moves which are in that order (e.g. we allow F then B, but not B then F). This allows us to reduce the number of possible moves that need to be considered in the IDA* search, resulting in a reduced branching factor for the search algorithm.

2.2.3 Pattern Databases

2.2.4 Sequential Indexing with Lehmer Codes

2.3 Feather's Algorithm

2.4 God's Number

2.5 The Kilominx

3. Requirements Specification

The following requirements were derived from the primary and secondary project objectives (as outlined in **Section 1.1**).

Requirements for Primary Objectives:

- Model the behaviour of a Rubik's Cube and a Kilominx, allowing them to be manipulated by making moves.
- Create representations for the pattern databases, which calculate the Lehmer rank of a permutation of cubie indices to determine the permutation's index in the pattern database.
- Populate the pattern databases by implementing a breadth-first search (BFS) algorithm or an iterative deepening depth-first search (IDDFS) algorithm.
- Implement an IDA* search algorithm to find the optimal solution for a given puzzle state, using the pattern databases as a heuristic function.
- Create a command-line interface which lets the user interact with the puzzles and the solver, allowing the user to: view the current state of the puzzle; enter commands to make moves; and use the solver to find an optimal solution for a given puzzle state.

Requirements for Secondary Objectives:

- Use the solver to increase the lower bound of the Kilominx's God's Number by finding an optimal solution of a greater number of moves than the current lower bound.
- Create a simple GUI program which uses the previously created classes to allow users to interact with a Rubik's Cube and Kilominx.
- Connect the solver to the GUI to find the optimal solution for the puzzle, and then display the solution to the user.

4. Software Engineering Process

4.1 Software Development Process

An incremental development process was used for this project in order to ensure steady progress throughout both semesters. This approach allowed me to slowly build up the solver in small, functional increments, where each increment was thoroughly tested and documented before moving on to the next. This method helped ensure that the final solution was robust and well-documented.

At the start of the project, I began by creating a reference guide for creating an optimal solver for the Rubik's Cube. This guide detailed each of the steps required to model the Rubik's Cube and then create the optimal solver for it, highlighting any important design decisions that needed to be made along with concise justifications for the decisions. This guide was used as a roadmap for the development of the solver, ensuring that I stayed on track towards the final goal.

After creating the reference guide, I followed it to implement an optimal solver for the Rubik's Cube. While creating a solver for the Rubik's Cube was not part of the project objectives, it was a necessary step in order to gain a better understanding of the process required to create an optimal solver. I took care while implementing the classes and methods to ensure that they were modular and reusable - this allowed me to easily extend the both the solver and other classes to work with the Kilominx later on. The Rubik's Cube solver was thoroughly tested to ensure that it was working correctly before moving on to implementing the Kilominx solver.

Finally, I implemented the Kilominx solver following the same steps from the reference guide as I did for the Rubik's Cube solver. I reused many of the classes and methods from the Rubik's Cube solver, extending them where necessary to work with the Kilominx. The Kilominx solver was also thoroughly tested to ensure that it was working correctly.

4.2 Choice of Language

I decided to use Java as the primary programming language for the solver. This decision was made based on my familiarity with Java and its object-oriented principles, such as abstract classes and inheritance, which I believed would be helpful for allowing parts of the code to be reused and extended. Additionally, Java is a relatively fast language and is well-suited for the type of computation required for solving the Rubik's Cube and Kilominx. There was some consideration given to using C++ due to its speed and efficiency, but due to my lack of experience with the language, I ultimately decided that Java would be the best choice for this project.

5. Ethics

The original project objectives involved surveying users about their experience with the web application, so the Artefact Evaluation form was filled out and submitted. However, as this objective was moved down to a lower priority, the user survey was ultimately not conducted. The Artefact Evaluation form can be found in **Appendix A**.

6. Design

6.1 Design Overview

6.2 Representing State

6.3 Making Moves

6.4 Populating Pattern Databases

6.5 Lehmer Codes

6.6 Searching for Solutions

6.7 Interacting with the Puzzles

7. Implementation

8. Evaluation

8.1 Experimental Results

8.2 Evaluation Against Objectives

8.3 Comparison to Existing Work

9. Conclusions

A. Artefact Evaluation Form

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
ARTIFACT EVALUATION FORM

Title of project

Rubik's Cube Variant Solvers

Name of researcher(s)

Samuel Borg

Name of supervisor

Ian Gent

Self audit has been conducted [✓ **YES**] **NO** ☐

This project is covered by the ethical application CS15727.

Signature Student or Researcher

S. Borg

Print Name

Samuel Borg

Date

25 / 09 / 2024

Signature Lead Researcher or Supervisor

[illegible]

Print Name

IAN GENT

Date

27/9/2024

B. Testing Summary

C. User Manual

References

[1] *The Lighter Side of Mathematics*. URL: https://archive.org/details/lightersideofmat0000unse/page/340/mode/1up. Last accessed on 25/03/25.

[2] *Mathematics of the Rubik's Cube*. URL: https://ruwix.com/the-rubiks-cube/mathematics-of-the-rubiks-cube-permutation-group/. Last accessed on 25/03/25.

[3] R. E. Korf. "Finding Optimal Solutions to Rubik's Cube Using Pattern Databases". In: *AAAI'97*. 1997. URL: https://www.cs.princeton.edu/courses/archive/fall06/cos402/papers/korfrubik.pdf. Last accessed on 25/03/25.

[4] J. C. Culberson and J. Schaeffer. "Pattern Databases". In: *Computational Intelligence* 14.3 (1998). URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/0824-7935.00065. Last accessed on 26/03/25.

[5] T. Rokicki et al. *God's Number is 20*. URL: https://www.cube20.org/. Last accessed on 26/03/25.

[6] *God's Algorithm - Optimal solutions of the Rubik's Cube*. URL: https://www.speedsolving.com/wiki/index.php/God's_Algorithm#Minxes. Last accessed on 26/03/25.

[7] *Rubik's Cube Notation*. URL: https://ruwix.com/the-rubiks-cube/notation/. Last accessed on 29/03/25.