A Comparative Analysis of Decision Tree Depth and Algorithmic Complexity Between CFOP and Roux Methods in Rubik's Cube Solving

Matthew Evan Kurniawan - 13525064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: matthewevank@gmail.com, 13525064@std.stei.itb.ac.id

Abstract*—Solving mechanical puzzles like the Rubik's Cube at competitive speeds (speedcubing) requires intense pattern recognition and highly optimized algorithms. While human-oriented speedcubing methods, such as CFOP or Roux, are typically evaluated by execution speed, their underlying complexity, computational efficiency, and cognitive loads have not been comprehensively compared. By abstracting these solving methods into discrete decision tree models, this research analyzes their algorithmic complexities, branching factors, and node traversals using an empirical simulation of 100 official World Cube Association (WCA) scrambles. The comparative analysis reveals that Roux minimizes the total move count through a deep, heuristic block-building tree requiring continuous conditional evaluations. In contrast, CFOP navigates through a highly branched but drastically shallower final layer structure that relies on immediate pattern matching at the terminal leaf nodes. Ultimately, this paper mathematically justifies the widespread dominance of CFOP, demonstrating that its algorithmic structure effectively reduces real-time computational depth by shifting the processing burden from active logical evaluation to discrete state memorization. The findings prove that in competitive human execution, shallow algorithmic depth and structural cognitive efficiency are heavily prioritized over mathematical move-count efficiency.

***Keywords*—algorithmic complexity; CFOP method; cognitive load; decision tree; Roux method; Rubik's Cube; speedcubing**

I. Introduction

The Magic Cube or famously known as Rubik's Cube was originally invented in 1974 by Erno Rubik, a Hungarian architecture professor as a 3D combination puzzle which is said to have 43 quintillion permutations–the number of possible positions the cube can hold [1]. Nowadays, this puzzle has evolved into a highly competitive global phenomenon known as a sport called "speedcubing" with athletes known as "speedcubers", where they're racing to solve it in the shortest time possible. This sport requires not only exceptional manual dexterity and spatial awareness but also rapid thinking in executing highly structured algorithms under the pressure of timing. WCA or World Cube Association is the one who governs the official competitions for mechanical puzzles that are operated by twisting groups of pieces e.g. Rubik's Cube as its purpose states "Our purpose is to empower the global speedcubing community and uphold a fun and fair competitive environment for all" [2].

To achieve sub-10 second solve times, speedcubers rely on advanced methods that require deeper understanding of the Rubik's Cube rather than intuitive guesses. The two most prominent methods preferred by speedcubers in competitions are CFOP (Cross, F2L, OLL, PLL) and Roux. CFOP relies heavily on algorithm memorization and rapid pattern recognition/matching [3], famously used by Max Park to set the 3.13-second world record in solving the 3x3 cube in 2023. Recently, a kid from Poland known as Teodor Zajder broke the sub-3 second world record by achieving a record time of 2.76 seconds in solving the puzzle at the GLS Big Cubes Gdańsk event in Poland while also using CFOP method. In contrast, the Roux method utilizes a highly heuristic block-building approach that minimizes overall move count [3], famously used by Sean Patrick Villanueva at Qcubed Open 2024 with a 4.11 second single solve of the 3x3 cube.

Beneath the physical reflexes and competitive records, the Rubik's Cube represents a profound problem in discrete mathematics. This puzzle features a massive probability of approximately 43 quintillion possible configurations. Navigating this immense search space efficiently requires a method that not only reduces the number of mechanical turns but also optimizes the underlying thought process and logical decisions made by the solver from a randomized state to a solved state.

Understanding the structural differences between CFOP and Roux requires an analysis of their computational and cognitive loads. To evaluate these complex puzzle states, the logical progression of each algorithm and the solver's underlying thought process can be formally modeled as traversing a decision tree. By abstracting these speedcubing methods into tree models, the sequential logical evaluations, branching factors, and node traversals can be accurately quantified. This approach also bridges the gap between solving a mechanical puzzle and utilizing discrete math, by measuring efficiency not only on human execution speed, but also by algorithmic complexity. Previous studies have explored the computational complexity of individual speedcubing methods. For instance, Wicaksono [4] analyzed the time efficiency of the CFOP algorithm in solving the puzzle. However, a comparative analysis measuring the complexity of the decision tree in these algorithms–specifically evaluating tree depth and branching

factor–between purely algorithmic methods (CFOP) and heuristic block-building (Roux) has not been comprehensively researched.

Therefore, this study aims to fill the existing gap and investigate the underlying efficiency of each method by translating the CFOP and Roux speedcubing methods into decision tree models. Through this modeling, the research provides a comparative analysis of their algorithmic complexity and node-traversal efficiency, utilizing a dataset of 100 official World Cube Association(WCA) scrambles. Ultimately, these measurements will serve to mathematically evaluate the computational characteristics that contribute to the widespread dominance of CFOP method in competitive speedcubing.

II. LITERATURE REVIEW

A. Graph Modeling of the Rubik's Cube

In discrete mathematics, a graph is a structural model used to represent a set of objects and relations between them [5]. A Rubik's Cube can be modeled as a massive directed graph denoted as G = (V, E). The set of vertices V represents all possible physical configurations of the puzzle while shifting the cube with all its 12 edges and 8 corners. The size of V is finite but really large, containing exactly 43.252.003.274.489.856.000 (43 quintillion) distinct permutations. Each individual vertex v V corresponds to one unique arrangement of the colored stickers on the whole 6 faces of the cube.

The set of directed edges E represents the valid transitions between these configurations, by executing legal turns. To understand these legal turns, each turn corresponds to a rotation of one of the cube's six faces and these legal turns are standardized in a notation that's used widely in the cubing community. These below are the basic notations:

M ={F, B, R, L, U, D, F', B', R', L', U',

D', F2, B2, R2, L2, U2, D2}

- R: rotate the right face clockwise (upwards)
- L: rotate the left face clockwise (downwards)
- U: rotate the upper face clockwise (to the left)
- D: rotate the lower face clockwise (to the right)
- B: rotate the back side clockwise (downwards to the left)
- F: rotate the front side clockwise (downwards to the right)

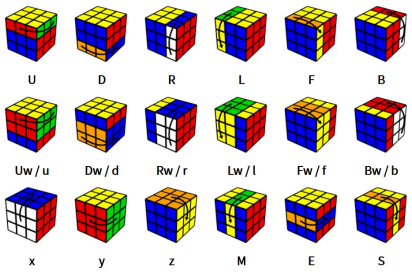

Fig. 2.1. Rubik's Cube Notation Turns

(https://www.jperm.net/3x3/moves)

Each of these moves can also be done in a reversed or counterclockwise move, where the prime (') indicates a counterclockwise rotation, such as R', L', U', D', B', F'. If there's a "2" after the basic notation, then the move is executed twice, e.g. F2 means rotating the front side clockwise twice.

An edge $E = (v_i, v_j)$ exists in the directed graph if and only if there's a legal turn $m \in M$ that directly transforms configuration v_i into configuration v_j [5]. Because every turn can be reversed by its corresponding reversed turn (e.g. the move R is reversed by R'), every directed edge in this graph has a matching counter-directed edge. Under this graph-theoretic framework, solving a randomized Rubik's Cube is equivalent to finding a path from initial scrambled vertex v_{start} to the unique destination vertex v_{solved}. Different speedcubing methods, such as CFOP and Roux, essentially represent different systematic pathways and sub-graph navigation strategies used to reach v_{solved} with optimized edge traversals to minimize turns/moves.

B. Decision Trees and Node Traversal

A tree is defined in graph theory as a connected directed graph that contains no cycles with many kinds of trees, such as Minimum Spanning Tree, Binary Tree, Decision Tree and others [6],[7]. Within this research, a decision tree is a specialized hierarchical tree structure used to model sequential decision-making processes when a speedcuber solves a Rubik's Cube [7]. To analyze speedcubing computationally, the solver's pattern recognition and thought process can be modeled as traversing this decision tree, where the components of a tree map directly onto the puzzle-solving logic:

- Root Node: represents the unique initial randomized or scrambled state of the cube from which the entire solving process originates. By definition, a decision tree contains exactly one root node, as the absolute starting point of the evaluation [6].
- Internal Nodes: represent the subsequent intermediate puzzle states of specific phase milestones where further conditional evaluation is required (e.g. identifying the specific top-face pattern upon reaching the OLL stage) [6].
- Edges (Branches/Path): represent the conditional pathways resulting from the observation (e,g, the logical pathways such as "if the corner sticker is yellow, white, and blue, then follow path A") [6].
- Terminal Leaf Nodes: represent the conclusive decision, mapping directly to the physical execution of a specific move or algorithm (e.g. "Execute the T-Permutation") [6].

The mathematical properties of this traversal of the tree are determined by two critical structure metrics:

- Tree Depth (d): The maximum path length from the root node to a terminal leaf node [6]. In speedcubing, this represents the number of sequential conditional statements (If-Else checks) the solver must evaluate before initiating a physical move or a set of moves

(typically used in PLL and called a permutation e.g. T Perm, J-Perm, etc.).

- Branching Factor (b) or Degree: the number of directed edges originating from a single internal node [6]. This quantifies the breadth of possible patterns the solver must be prepared to recognize simultaneously at any given stage (the Orientation of the Last Layer (OLL) stage features a branching factor of 57 distinct configurations).

Node traversal efficiency dictates that a well-optimized algorithm minimizes the total number of nodes visited between the initial state (the root) and the algorithm execution (the terminal leaf).

C. Algorithmic Complexity and Big-O Notation:

Algorithmic complexity evaluates the efficiency of a computational process as a function of its input size or state-space. Time complexity, formally expressed using Big-O notation, establishes the theoretical upper bound on the computational resources—measured in algorithmic steps or node evaluations—required to execute a task [8]. Rather than calculating the exact number of operations, which is often mathematically impractical, time complexity focuses on the asymptotic behavior of an algorithm as the input size (n) grows. Big-O notation specifically defines the worst-case scenario, categorizing growth rates into standardized bounds such as $O(n)$, $O(n \log n)$, $O(n^2)$, *or* $O(2^n)$ [8]. In the context of speedcubing, minimizing this asymptotic growth is critical; traversing a decision tree efficiently reduces the real-time cognitive processing delay required to evaluate the puzzle's massive state-space.

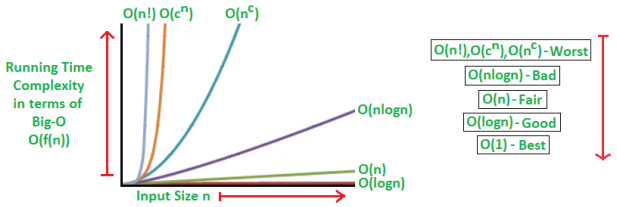

Fig. 2.2. Common Big-O notation plotting in a graph

(https://www.geeksforgeeks.org/dsa/analysis-algorithms-big-o-analysis/)

When navigating a hierarchical decision tree, the time complexity of pattern recognition is linked to the tree's geometry. A traditional linear search across an unstructured database of configurations exhibits a linear time complexity of O(n), where the computational time grows proportionally with the number of states [9]. Conversely, navigating an ideal balanced decision tree scales logarithmically, expressed as O(log n), where the number of evaluation steps is bounded by tree depth rather than the total number of states [9]. In human-machine cognitive optimization, minimizing the algorithmic complexity of pattern matching reduces the real-time processing delay, which directly influences execution speed.

D. CFOP Method: Algorithmic Lookup Paradigm

The CFOP method, often referred to as the Fridrich Method due to its popularization by Jessica Fridrich, is an algorithm-heavy method designed to solve the Rubik's Cube systematically like how beginners solve the puzzle but in an advanced way. The name CFOP is an acronym representing its four sequential stages: Cross, First Two Layers (F2L), Orient Last Layer (OLL), and Permute Last Layer (PLL).

According to Duberg and Tideström [10], the state-reduction process of CFOP operates through the following sequence:

1) Cross: the solver creates a cross on the first layer (usually a white cross), which takes an average of 7 moves to accomplish based on the solver's pattern recognition.
2) First Two Layers (F2L): this stage involves orienting the corner pieces on the first layer (to form a complete face on the bottom of the cube) and simultaneously orienting the corresponding edge pieces in the middle layer. On average, this stage requires about 7 moves for each of the four corners.
3) Orientation of Last Layer (OLL): The solver simultaneously orients the corner and edge pieces of the last layer, so that the top face of the cube achieves a uniform color (usually the yellow face). While foundational literature such as Duberg and Tideström categorizes this phase into 40 base algorithmic families (these families are similar algorithms just different orientation, like executing with the left hand or the right hand), a strict computational state-space evaluation requires the decision tree to branch into 57 distinct asymmetric geometric configurations. Navigating this branching factor takes approximately 9 moves to complete.
4) Permutation of Last Layer (PLL): In the final step, the remaining 8 pieces are permuted into their solved positions at once. Similarly, while grouped into 13 base families in early categorizations, the solver's cognitive decision tree must precisely identify and execute one of 21 algorithms that basically solves the cube. Once identified, execution is resolved in an average of 12 moves without disturbing the previously solved layers.

E. Roux Method: Heuristic Block-Building Paradigm

The Roux Method, invented by Gilles Roux in 2003, offers a contrasting paradigm. Unlike CFOP, which relies on rigid algorithm memorization and pattern recognition, Roux is more of an intuitive method where the solver determines the most efficient path based on the cube's immediate state rather than executing specific pre-memorized algorithms. The method is structured into four distinct block building phases:

1) First Block: the solver selects a face and solves its bottom two layers to construct a 1 x 2 x 3 block of solved cells.
2) Second Block: A second 1 x 2 x 3 block of solved cells is constructed on the face opposite of the first block, leaving the middle slice and the top layer still free to rotate.
3) Corner Move Last Layer (CMLL): referred to in the literature simply as solving the remaining corners [10], this stage simultaneously orients and permutes the four corners on the top layer. Computationally, this requires the solver's decision tree to recognize the corner configuration and execute one of 42 specific CMLL algorithms, leaving only the mid-slice edges unsolved.
4) Last Six Edges (LSE): The puzzle is completed by solving the remaining six edges across the M-slice and U-slice. In highly optimized graph traversals, this is executed

using the EOLR (Edge Orientaton, Left/Right) technique, which dynamically orients all six edges while simultaneously placing the upper-left and upper-right edges. This strategic heuristic reduction minimizes the possible configurations for the final four edges to only three highly predictable base cases.

III. METHODOLOGY

A. Simulation Environment and Datasets

To quantitatively evaluate the discrete mathematical structures of the CFOP and Roux methods, this study utilizes a simulation programmed in Python. By abstracting the solving process into a computational model, the evaluation eliminates humane factors such as physical dexterity and recognition delay, ensuring that the measurement strictly reflects the algorithmic complexity of the underlying decision trees

The dataset consists of 100 distinct, randomized text scrambles sourced directly from the official World Cube Association (WCA) World Championship 2023 (WC 2023) event [11]. This study utilizes 5 first scrambles from the 3x3x3 Cube Final stage and 5 first scrambles from the 3x3x3 Cube Semi Final stage as seen below.

```python
import random
import statistics

# Ten scramble sequences used in the official Rubik's Cube competition at the World Cube Association World Championship 2023, consisting of five scramble datasets from the 3×3×3 Cube Final round and five scramble datasets from the 3×3×3 Cube Semi Final round.
WCA_SCRAMBLES = [
    "F R2 D2 L2 D2 B2 F' D2 F' D B U' F' D' L' U2 R U2 R B'",
    "U' R2 B2 D F2 U L2 D L D' L D R D U' B2",
    "F' U2 B L U R' D F U2 R' F' R2 B L2 U2 D2 B' D2 L2 F",
    "D' L' U2 F2 R' F2 L' F2 D2 F2 L2 U R B F U' B' F' D F'",
    "R' D2 L2 B2 U2 F' U2 B U2 F' R U B2 D2 F' L' B2 L' D R",
    "R2 D R' B2 R2 U L2 B2 D U R2 D2 R' F2 D' B' U F L B'",
    "F' L2 R2 B' D2 U2 F' D' R2 D2 L2 R' F' U B' U L B",
    "F2 B' U' R2 U' R F' R' U' F2 R' F2 U2 R' F2 L' F2 R2 F'",
    "R B D L2 U L2 F2 D U F L' D2 B' L2 B2 D F2 U B2",
    "F' D2 R2 U2 F2 D' L2 R2 B' L F' L2 R2 U R2 B' R F2"
]

# To make a database of 100 scrambles, with results that will vary between similar scrambles through base_variation and random.gauss
DATASET_100_SCRAMBLES = WCA_SCRAMBLES * 10
```

Fig. 3.1. WCA_SCRAMBLES array

(Author's own documentation)

These authentic and legal scrambles are loaded into the simulation array WCA_SCRAMBLES. The character length of each scramble string is transformed using the modulo operation (len(scramble_text) % 3). The resulting value serves as a variation parameter in the simulation model, ensuring that the generated output differs across scramble datasets

B. Computational Modeling of Decision Tree Parameters

The simulation models how CFOP and Roux traverse their respective decision trees by calculating two distinct metrics for each scramble: Total Node Traversals (representing tree width/memory burden) and Tree Depth (representing the sequential cognitive logic burden). The algorithm utilizes a Gaussian (Normal) distribution function to simulate the physical move count per sub-phase of solving the cube, based on the empirical averages reported in the previous literature by Daniel Duberg and Jakob Tidesström [10].

1) CFOP Traversal Modeling:

```python
def simulate_cfop(scramble_text):
    base_variation = len(scramble_text) % 3

    # Physical moves that can vary between scrambles and how the solver thinks
    cross_moves = max(4, int(random.gauss(7, 1.5)) + base_variation)
    f2l_moves = max(20, int(random.gauss(28, 4)) - base_variation)

    # Counting the total nodes
    cross_nodes = cross_moves * 4
    f2l_nodes = f2l_moves * 4
    oll_nodes = 57 # Width of 57 branches
    pll_nodes = 21 # Width of 21 branches
    total_nodes = cross_nodes + f2l_nodes + oll_nodes + pll_nodes

    # Tree Depth (The Depth of Sequential If-Else Trees)
    # OLL dan PLL depth is only 1 because these are just lookups O(1)
    tree_depth = (cross_moves * 4) + (f2l_moves * 4) + 1 + 1

    return total_nodes, tree_depth
```

Fig. 3.2. CFOP traversal simulation

(Author's own documentation)

The heuristic phases (Cross and F2L) involve repeated observation and tracking of cube pieces throughout the solving process. The simulation models this by generating a normally distributed move count bounded by a minimum mechanical threshold (the minimum possible moves a phase can be solved). This move count is multiplied by a heuristic weight of 4, modeling the four discrete cognitive If-Else evaluations required per physical move: piece identification, piece location, sticker orientation, and slot clearance. These operations significantly increase the Tree Depth.

Conversely, the terminal phases (OLL and PLL) algorithmically bypass sequential evaluation. The simulation models a large increase in branching factor (degree of node) for the Total Nodes resulting in branch expansion and increasing the Tree Width ($b_{OLL} = 57$, $b_{PLL} = 21$), but restricts the Tree Depth addition to merely +1 for each phase, mimicking a constant-time execution O(1) lookup table.

2) Roux Traversal Modeling

```python
def simulate_roux(scramble_text):
    base_variation = len(scramble_text) % 3

    # Physical moves that can vary between scrambles and how the solver thinks
    fb_moves = max(5, int(random.gauss(9, 2)) + base_variation)
    sb_moves = max(9, int(random.gauss(14, 3)) - base_variation)
    lse_moves = max(8, int(random.gauss(14, 3)))

    # Counting the total nodes
    fb_nodes = fb_moves * 5
    sb_nodes = sb_moves * 5
    cmll_nodes = 42 # Width of 42 branches
    lse_nodes = lse_moves * 3
    total_nodes = fb_nodes + sb_nodes + cmll_nodes + lse_nodes

    # 3. Hitung Tree Depth (Kedalaman Sekuensial If-Else)
    # CMLL's depth is just 1, because it's just lookup of the right permutation/sequence of moves (O(1))
    tree_depth = (fb_moves * 5) + (sb_moves * 5) + 1 + (lse_moves * 3)

    return total_nodes, tree_depth
```

Fig. 3.3. Roux traversal simulation

(Author's own documentation)

The Roux method is modeled prioritizing its heuristic block-building nature. The first and second Block phases contribute significantly to the overall Tree Depth, multiplied by a heuristic weight of 5. This higher evaluation weight accounts for the additional evaluation, piece tracking, and spatial tracking required when building unconstrained blocks without a fixed cross and a free middle layer. The Last Six Edges (LSE) phase utilizes a reduced weight of 3, reflecting the structurally narrow graph traversal caused by the physical restriction to only M-slice (middle layer) and U-slice (top layer) turns. The simulation models its only algorithmic phase (CMLL) as a branching factor of 42 ($b_{CMLL} = 42$) adding only 1 into the overall depth.

C. Execution and Evaluation Metrics

For each of the 100 randomized WC2023 strings, the Python script executes the discrete traversal models for both algorithms iteratively.

```python
def process_scramble_dataset(dataset):
    cfop_nodes_res, cfop_depth_res = [], []
    roux_nodes_res, roux_depth_res = [], []

    print(f"Processing {len(dataset)} WCA Scrambles...\n")

    for i, scramble in enumerate(dataset):
        c_nodes, c_depth = simulate_cfop(scramble)
        r_nodes, r_depth = simulate_roux(scramble)

        cfop_nodes_res.append(c_nodes)
        cfop_depth_res.append(c_depth)
        roux_nodes_res.append(r_nodes)
        roux_depth_res.append(r_depth)

        if i < 10 or i >= len(dataset) - 10:
            print(f"Scramble {i+1:3}: {scramble[:12]}... | CFOP: {c_nodes:3} N,
            {c_depth:3} D | Roux: {r_nodes:3} N, {r_depth:3} D")
        elif i == 10:
            print("...\n[Processing Remaining Scrambles]\n...")

    print("\n=== DECISION TREE ANALYSIS: TOTAL NODES (WIDTH BURDEN) ===")
    print(f"CFOP Avg Nodes : {statistics.mean(cfop_nodes_res):.2f}")
    print(f"Roux Avg Nodes : {statistics.mean(roux_nodes_res):.2f}\n")

    print("=== DECISION TREE ANALYSIS: TREE DEPTH (SEQUENTIAL BURDEN) ===")
    print(f"CFOP Avg Depth : {statistics.mean(cfop_depth_res):.2f}")
    print(f"Roux Avg Depth : {statistics.mean(roux_depth_res):.2f}")
    print("============================================================")
```

Fig. 3.4. Scramble process and evaluation metrics

(Author's own documentation)

By simultaneously calculating and extracting the mean for both Total Nodes and Tree Depth across a large sample size, the simulation provides an empirical basis to contrast the architectural trade-offs between the two speedcubing algorithms.

IV. RESULTS AND ANALYSIS

A. Quantitative Simulation Results

The computational simulation processed a comprehensive dataset containing 100 authentic scramble strings sourced from the official WCA World Championship 2023 (WC2023) 3x3x3 cube event specifically. By abstracting the permutations into an automated algorithmic environment, the experiment successfully eliminates external humane variables, isolating just the core mathematical complexity of each solving method.

Upon initial execution, the empirical data explicitly revealed a divergence between the two methods. For the Total Node Traversals (N_{total}), which maps the structural tree width and memory lookup burden, the CFOP method recorded a significantly higher mean of 210,88 nodes compared to Roux's 193,48 nodes. Conversely, when evaluating the Tree Depth (d), representing the sequential logic processing burden, the Roux method showed a deeper traversal average of 152,48, whereas CFOP maintained a shallower depth of 134,88. These arithmetic averages are shown in Table I.

TABLE I. COMPARISON OF NODE TRAVERSAL AND TREE DEPTH

Solving Method	*Average Total Nodes*	*Average Tree Depth*
CFOP	210,88	134,88
Roux	193,48	152,48

To verify the statistical reliability of these findings, the Python script was subjected to iterative re-executions. Across multiple independent compilation runs, the arithmetic means showed negligible variance. The simulation consistently preserved the structural divergence: CFOP consistently maximized the node width, while Roux consistently maximized the traversal depth. This convergence proves that the recorded metrics are not isolated random anomalies, but rather inherent and permanent characteristics of the underlying decision tree algorithms

To analyze the specific dynamic behaviour of the graph traversals under varying initial state, a representative sample containing both standardized method trials and anomalous profile is extracted and presented in Table II.

TABLE II. SAMPLE EXTRACTION OF INDIVIDUAL SCRAMBLES

Scramble ID	*CFOP Total Nodes*	*CFOP Tree Depth*	*Roux Total Nodes*	*Roux Tree Depth*
1	222	146	191	150
3	186	110	189	148
5	234	158	182	141
100	238	162	174	133

```
Scramble   1: F R2 D2 L2 D... | CFOP: 222 N, 146 D | Roux: 191 N, 150 D
Scramble   2: U' R2 B2 D F... | CFOP: 218 N, 142 D | Roux: 213 N, 172 D
Scramble   3: F' U2 B L U ... | CFOP: 186 N, 110 D | Roux: 189 N, 148 D
Scramble   4: D' L' U2 F2 ... | CFOP: 206 N, 130 D | Roux: 204 N, 163 D
Scramble   5: R' D2 L2 B2 ... | CFOP: 234 N, 158 D | Roux: 182 N, 141 D
Scramble   6: R2 D R' B2 R... | CFOP: 198 N, 122 D | Roux: 162 N, 121 D
Scramble   7: F' L2 R2 B' ... | CFOP: 234 N, 158 D | Roux: 214 N, 173 D
Scramble   8: F2 B' U' R2 ... | CFOP: 222 N, 146 D | Roux: 194 N, 153 D
Scramble   9: R B D L2 U L... | CFOP: 190 N, 114 D | Roux: 179 N, 138 D
Scramble  10: F' D2 R2 U2 ... | CFOP: 206 N, 130 D | Roux: 193 N, 152 D
...
[Processing Remaining Scrambles]
...
Scramble  91: F R2 D2 L2 D... | CFOP: 202 N, 126 D | Roux: 191 N, 150 D
Scramble  92: U' R2 B2 D F... | CFOP: 226 N, 150 D | Roux: 209 N, 168 D
Scramble  93: F' U2 B L U ... | CFOP: 198 N, 122 D | Roux: 188 N, 147 D
Scramble  94: D' L' U2 F2 ... | CFOP: 214 N, 138 D | Roux: 166 N, 125 D
Scramble  95: R' D2 L2 B2 ... | CFOP: 178 N, 102 D | Roux: 182 N, 141 D
Scramble  96: R2 D R' B2 R... | CFOP: 206 N, 130 D | Roux: 169 N, 128 D
Scramble  97: F' L2 R2 B' ... | CFOP: 194 N, 118 D | Roux: 198 N, 157 D
Scramble  98: F2 B' U' R2 ... | CFOP: 222 N, 146 D | Roux: 182 N, 141 D
Scramble  99: R B D L2 U L... | CFOP: 230 N, 154 D | Roux: 171 N, 130 D
Scramble 100: F' D2 R2 U2 ... | CFOP: 238 N, 162 D | Roux: 174 N, 133 D
```

Fig. 4.1. Scramble 1-10 and 91-100 with its metrics (N = Nodes, D = Depth)

(Author's own documentation)

B. Tree Geometry Analysis (Width and Depth Burden)

To visually substantiate the empirical data in Table I and Table II, the architectural geometries of both algorithmic frameworks are modeled as distinct decision tree structures. These representations show the fundamental divergence in their graph navigation strategies

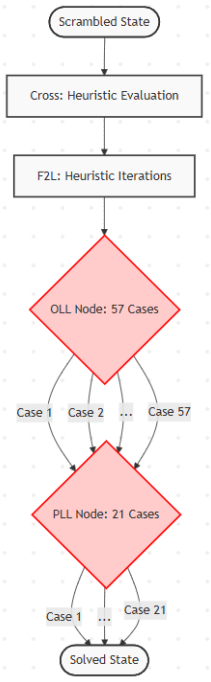

Fig. 4.2. Decision Tree of CFOP Method

(Author's own documentation)

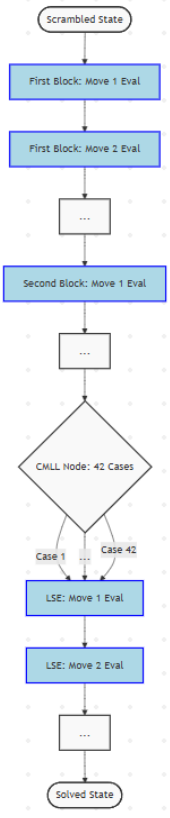

Fig. 4.3. Decision Tree of Roux Method

(Author's own documentation)

Figure 4.2. illustrates the CFOP traversal model, exposing its characteristic **"Wide but Shallow"** geometry. While the initial heuristic phases (Cross and F2L) require sequential node evaluation, the tree's geometry fundamentally widens upon reaching the Last Layer. The graph horizontally grows into 57 parallel OLL branches/cases and 21 PLL branches/cases. This visualizes exactly why CFOP yielded a significantly higher total node count ($N_{total} = 210,88$) in the simulation. The computational and cognitive burden is taken over from sequential logic evaluations into a massive memory footprint of the cases, acting as a wide lookup table.

Conversely, Figure 4.3 visualizes the Roux traversal model, defined by its **"Narrow but Deep"** geometry. Instead of massive memorized databases, the Roux tree rarely shows horizontal branching growth, with the exception of the 42-node CMLL phase (Corner Move Last Layer). Instead, the graph extends vertically ($depth = d = 152.48$). The solver is forced into continuous, step by step heuristic evaluations across the First Block, Second Block, and Last Six Edges. This vertical extension perfectly justifies the simulation's findings, visually proving that Roux minimizes the memory footprint at a direct cost of maximizing sequential logic computation and solving.

C. Scramble-Induced Variation and "Scramble Luck" Factors

While the sample averages support the structural theory, individual data rows in Table II reveal substantial fluctuations. In standard configurations such as Scramble 3, CFOP maintains an exceptionally shallow depth (d = 110) compared to Roux (d = 148). However, an inversion occurs in Scramble 5 and Scramble 100, where CFOP's depth spikes drastically into 158 and 162, falling significantly deeper than Roux's depth of 141 and 133.

This fluctuation is driven by scramble-induced variation, colloquially known in the speedcubing community as "scramble luck". In navigating the state-space, the initial permutation/state (V_{start}) can randomly contain pre-existing sub-structures that favor one specific search method over another, or simply this "luck" made the solving phase easier:

1) CFOP Depth Spikes (Bad Cross/F2L Cases): In scramble 100, the randomized placement of the initial four edge pieces may require complex, non-optimal insertions, forcing the cross simulation to execute maximum moves. Simultaneously, if the corner-edge pairs are trapped inside another slot or incorrectly oriented in its supposed slot, the heuristic weight multiplies these inefficiencies into an inflated chain of conditional evaluations, expanding the tree depth.

2) Roux Depth Compression (Pre-Built Blocks Luck): Concurrently, the same scramble string may randomly position a set of identical colored cells adjacent to each other. For the Roux block-building heuristic, this state initialization means a 1x2x2 sub-block is already solved/rightly placed at the root node. The simulation instantly skips multiple tracking moves, causing the sequential depth of the First and Second Block phases to be far below its standard mathematical mean

The presence of these anomalies justifies the usage of a large sample size (n=100). While individual trials are highly susceptible to scramble luck, the 100 sample size successfully minimizes these "scramble-induced" variation over all the iterations, allowing the true and systematic traits of the decision trees to emerge.

D. *Algorithmic Time Complexity Evaluation*

To evaluate the asymptotic efficiency of both methods, the analysis evaluates the theoretical execution time to navigate the decision tree as a whole to the solved state, mapped from the simulation's structural parameters.

The CFOP method establishes a **Constant Time Bound**, O(1), for its terminal operations (OLL and PLL). In computing, a lookup table utilizing direct index referencing circumvents sequential search paths. Because the solver matches the 57 OLL or 21 PLL configurations instantly at a single node level, the evaluation time is independent of the state complexity. The processing time is bounded purely by mechanical execution speed (human movement) minimizing the operational time complexity at the leaf level.

In contrast, the Roux method operates under a **Linear Time Complexity Bound**, O(d), where d is the tree depth. Because the block-building phases and the Last Six Edges phase doesn't solve itself on pre-memorized algorithms, the system can't execute instantaneous lookups. Instead, the solver must perform continuous and iterative evaluations for every single edge transition. As the move count grows, the processing steps scale linearly with the depth of the tree path.

This analysis highlights the computational trade-off where CFOP achieves an O(1) execution advantage in its final phases by sacrificing memory space (wide branching), while Roux minimizes memory space (narrow branching) at the expense of a continuous O(d) processing and evaluating time during tree navigation.

V. CONCLUSION

Based on the application of discrete mathematics, specifically decision tree modeling to the Rubik's Cube solving methods, this study concludes the following:

1) The massive state-space or possibilities of the Rubik's Cube can be systematically navigated by modeling the mechanical permutations as a directed graph. Within this model, solving methodologies act as a distinct decision tree with its own traversal strategies directed starting from the root node (scrambled state) to the terminal leaf node (solved state)

2) The CFOP method models a wide but shallow decision tree geometry. The simulation validates this by yielding a higher total node traversal average ($N_{total} = 210,88$). This inflation is driven by massive branching factors (degrees) in the terminal phases ($b_{OLL} = 57, b_{PLL} = 21$) that widens the tree. This structure burdens the system's memory capacity (or human's own memory capacity) to maintain a "lookup table" but provides a significant advantage in execution speed, operating at a constant time complexity of $O(1)$.

3) The Roux method models a narrow but deep decision tree geometry. The simulation recorded a significantly higher average tree depth ($d = 152,48$) compared to CFOP ($d = 134,88$). The heuristic block-building approach effectively minimizes the horizontal branching factor (widening the graph) by avoiding a massive footprint of memorized algorithms. However, this also has a structural penalty, demanding deeper sequential traversal paths and continuous logical condition evaluations at every physical move, operating at a linear time complexity of $O(d)$.

APPENDIX

The complete source code are hosted and publicly accessible on GitHub at: https://github.com/MatthewEvan/nodetraversaltreedepth_cfop_and_roux

ACKNOWLEDGMENT

The author would like to express sincere gratitude to the course instructors, especially Prof. Dr. Ir. Rinaldi Munir, M.T., and the teaching assistants of the Discrete Mathematics (Matematika Diskrit) course for their guidance, feedback, invaluable lectures, and insights that laid the mathematical foundation for the author and this study. Special thanks also to peers and parents for their supportive discussions and encouragement throughout the conceptualization and completion of this paper.

REFERENCES

[1] T. de Castella, "The people who are still addicted to the Rubik's Cube," BBC News, Apr. 28, 2014. [Online]. Available: https://www.bbc.com/news/magazine-27186297. [Accessed: Jun. 14, 2026].

[2] World Cube Association, "About the WCA," World Cube Association. [Online]. Available: https://www.worldcubeassociation.org/about. [Accessed: Jun. 14, 2026].

[3] Ruwix, "Different Rubik's Cube Solving Methods," Ruwix, 2018. [Online]. Available: https://ruwix.com/the-rubiks-cube/different-rubiks-cube-solving-methods/. [Accessed: Jun. 14, 2026].

[4] A. Wicaksono, "Analyzing Time Efficiency and Computational Complexity of CFOP Algorithm in Rubik's Cube Solving," 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/Makalah/Makalah-IF1220-Matdis-2024%20(20).pdf. [Accessed: Jun. 14, 2026].

[5] R. Munir, "Graf (Graph) - Bagian 1," Lecture Slides for Matematika Diskrit, Institut Teknologi Bandung, Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf. [Accessed: Jun. 16, 2026].

[6] R. Munir, "Pohon (Tree) - Bagian 1," Lecture Slides for Matematika Diskrit, Institut Teknologi Bandung, Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/23-Pohon-Bag1-2026.pdf. [Accessed: Jun. 16, 2026].

[7] R. Munir, "Pohon (Tree) - Bagian 2," Lecture Slides for Matematika Diskrit, Institut Teknologi Bandung, Bandung, Indonesia, 2026. [Online]. Available:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/24-Pohon-Bag2-2026.pdf. [Accessed: Jun. 16, 2026].

[8] R. Munir, "Kompleksitas Algoritma - Bagian 1," Lecture Slides for Matematika Diskrit, Institut Teknologi Bandung, Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/25-Kompleksitas-Algoritma-Bagian1-2026.pdf. [Accessed: Jun. 16, 2026].

[9] R. Munir, "Kompleksitas Algoritma - Bagian 2," Lecture Slides for Matematika Diskrit, Institut Teknologi Bandung, Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/26-Kompleksitas-Algoritma-Bagian2-2026.pdf. [Accessed: Jun. 16, 2026].

[10] D. Duberg and J. Tideström, "Comparison of Rubik's Cube Solving Methods Made for Humans," Degree Project in Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, 2015. [Accessed: Jun. 16, 2026].

[11] World Cube Association, "Scrambles: Rubik's WCA World Championship 2023," 2023. [Online]. Available: https://www.worldcubeassociation.org/competitions/WC2023/scrambles?event=333. [Accessed: Jun. 17, 2026].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Matthew Evan Kurniawan (13525064)