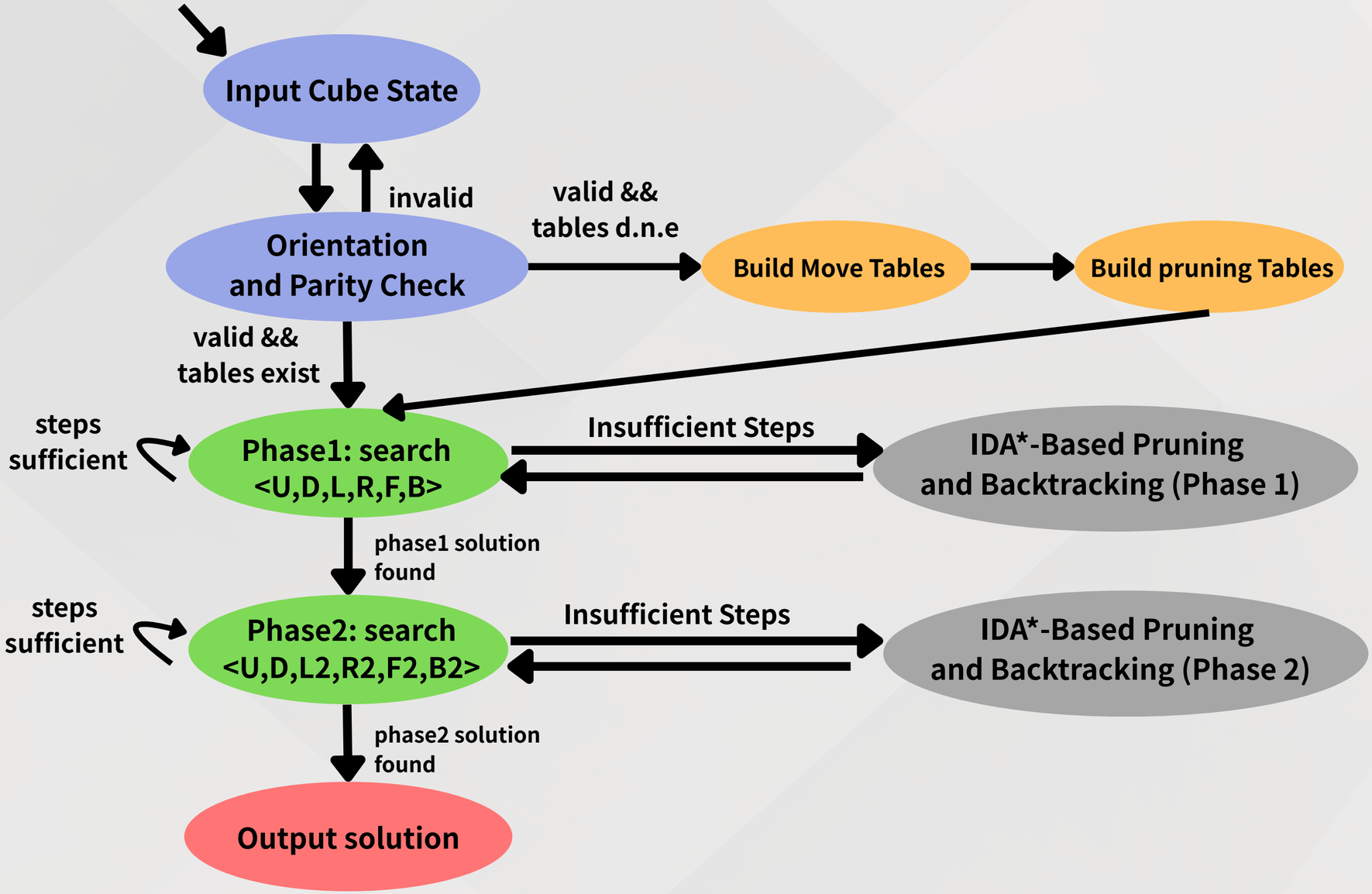
Implementation of a Rubik's Cube Solver Based on Cube Structure Analysis and Two-Phase IDA* Search

指導教授: 謝孫源 教授
專題生: 劉睿紘

Flowchart


Input Cube State
invalid
Orientation and Parity Check
valid && tables d.n.e
Build Move Tables
Build pruning Tables
valid && tables exist
steps sufficient
Phase1: search <U,D,L,R,F,B>
Insufficient Steps
IDA*-Based Pruning and Backtracking (Phase 1)
phase1 solution found
steps sufficient
Phase2: search <U,D,L2,R2,F2,B2>
Insufficient Steps
IDA*-Based Pruning and Backtracking (Phase 2)
phase2 solution found
Output solution

Structure

guilogger.py
main.py
main_window.py
worker.py

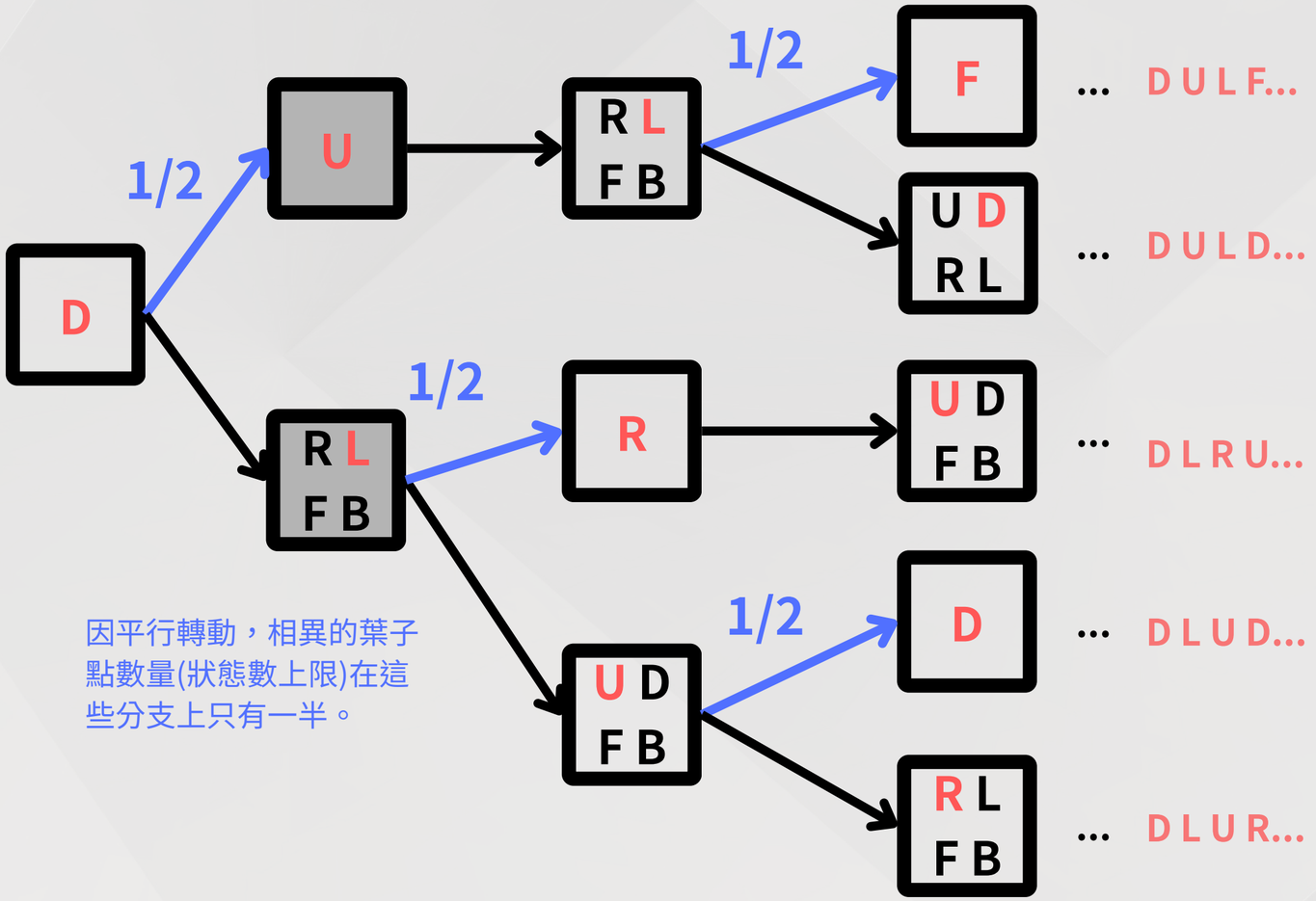
core/
- cube.py
- notation.py
- turns.py

solver/
- mainsolver.py
- phase1solver.py
- phase2solver.py

table/
- distance_table.py
- move_table_manager_p1.py
- p1_table.py
- move_table_manager_p2.py
- p2_table.py

ui/
- cube_tab.py
- cube_visualizer.py

Lower Bound Analysis of God's Number



因平行轉動，相異的葉子點數量(狀態數上限)在這些分支上只有一半。

恰n步轉動時的狀態上限 $f(n) =$

$$3^n \left(\frac{3+\sqrt{6}}{4}(2+\sqrt{6})^n + \frac{3-\sqrt{6}}{4}(2-\sqrt{6})^n \right)$$

$$\sum_{k=1}^{18} f(n) > c > \sum_{k=1}^{17} f(n)$$

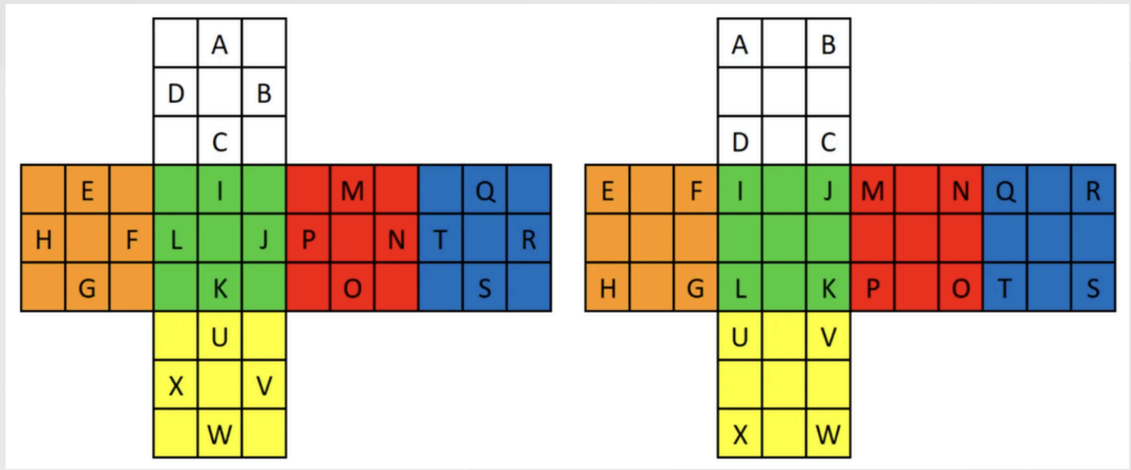
(c = 魔術方塊可能的排列數量)

Input

(1) Scramble

e.g U L2 B' L2 B D2 B2 D2 R2 F D2 B' R2 U' B' L' R B' D2 B' U2

(2) Pieces Definition



Solver

scramble

Domino
Reduction

solve

Phase 1
(orientation)

Phase 2
(permutation)

- **Domino Reduction**

By orienting all the corners and performing 2-axis EO, reducing the cube from <U,D,L,R,F,B> to <U,D,L2,R2,F2,B2>. (Group reduction)

equivalent to

1. The top and bottom faces are either yellow or white.

2. The edge orientation (EO) on the equator is correct.

Input State Validation

(1) Edge Orientation

Summation of orientation values must be even

→ s%2==0

(2) Corner Orientation

Summation of orientation values must be multiple of 3

→ s%3 == 0

(3) Permutation Parity

Edge Permutation Parity + Corner Permutation Parity = even

→ bubble sort(consider total swap number)

Move table

The table records state transitions for all possible moves. Each of the 18 moves is assigned a unique index (0–17).

Given a current cube state represented by an integer N, applying a move with index idx leads to a new state stored at position `N * 18 + idx` in the table.

Then, to determine how the cube changes, there is no need to copy or reconstruct the cube; we can simply look up the result in the table.

Pruning table

- The table stores the minimum number of moves required to solve the twist-slice and flip-slice states.

- It is precomputed using BFS, allowing the search phase to rely solely on efficient table lookups.

- The heuristic value for phase1 is defined as

```
max(fs_table.get(fs_val), ts_table.get(ts_val))
```

  which provides a lower bound on the number of moves required to reach the solved state.

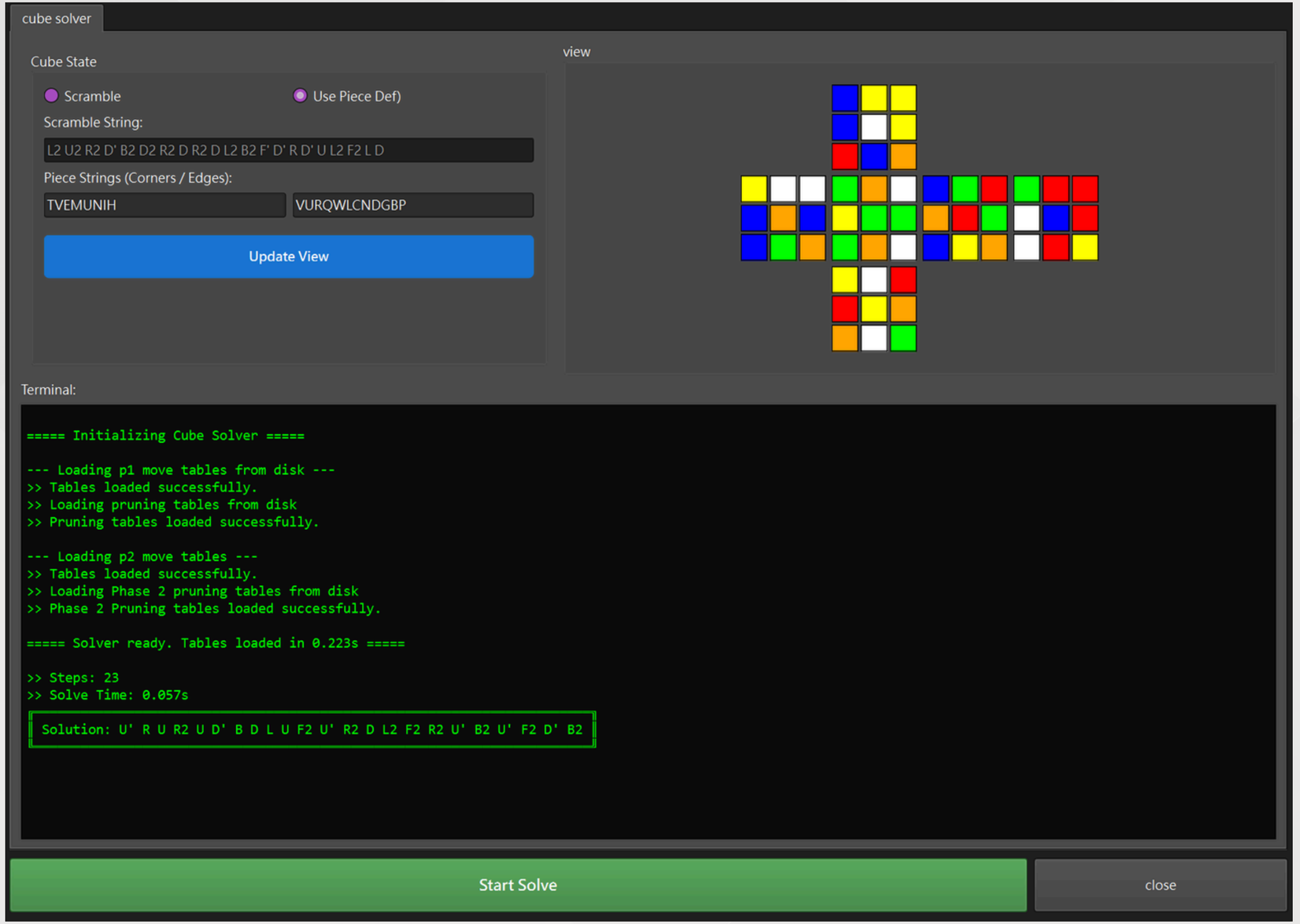
Solver: DFS + IDA*

- heuristic value h
  h = max(fs_table.get(fs_val), ts_table.get(ts_val))

- Given a limit(max depth for DFS), current moves: dist
  if dist + h > limit, return False(cut this path)

- If next turn face is same as previous → continue
- If next turn face is opposite to previous → compare idx
  (U D) = (D U)

Solver Execution Results



Execution with Existing Table

```
===== Initializing Cube Solver =====

--- Loading p1 move tables from disk ---
>> Tables loaded successfully.
>> Loading pruning tables from disk
>> Pruning tables loaded successfully.

--- Loading p2 move tables ---
>> Tables loaded successfully.
>> Loading Phase 2 pruning tables from disk
>> Phase 2 Pruning tables loaded successfully.

===== Solver ready. Tables loaded in 0.173s =====

>> Scramble: L2 U2 R2 D' B2 D2 R2 D R2 D L2 B2 R U F D R' L
>> Steps: 18
>> Solve Time: 0.055s
```

```
Solution: L' R D' F' U' R' B2 L2 D' R2 D' R2 D2 B2 D R2 U2 L2
```

Execution without Existing Table

```
===== Initializing Cube Solver =====

--- Starting p1 Table Generation ---
>> Building Flip-Slice Move Table...
>> Building Twist-Slice Move Table...
>> P1 move tables built and saved in 15.058 seconds.
>> Starting p1 Pruning Table Generation
>> Building Flip-Slice Pruning Table...
>> Building Twist-Slice Pruning Table...
>> P1 Pruning tables built and saved in 3.783 seconds.

--- Starting p2 Table Generation ---
>> Building CPMP Move Table...
>> Building EPMP Move Table...
>> Phase 2 Move tables built in 13.381s
>> Starting Phase 2 Pruning Table Generation
>> Building CP-MP Pruning Table...
>> Building EP-MP Pruning Table...
>> P2 Pruning tables built and saved in 2.098 seconds.

===== Solver ready. Tables loaded in 34.337s =====

>> Scramble: L2 U2 R2 D' B2 D2 R2 D R2 D L2 B2 F' D' R D' U L2 F2 L D
>> Steps: 23
>> Solve Time: 0.061s
╔═══════════════════════════════════════════════════════════════════╗
║ Solution: U' R U R2 U D' B D L U F2 U' R2 D L2 F2 R2 U' B2 U' F2 D' B2 ║
╚═══════════════════════════════════════════════════════════════════╝
```

Future Directions

- Improve solving efficiency through simultaneous computation across six search entry points.
  (3 axes × forward/reverse directions)

- Refine the heuristic function to better integrate A* with BFS by introducing an exploration factor e(n):

$$f(n) = g(n) + h(n) - e(n)$$