A Constant Time Algorithm for Solving Simple Rolling Cube Mazes

Randal Tuggle* Davis Murphy† Nicholas Lorch‡

Abstract

In a rolling cube maze, a cube is placed on a board and the task is to roll it to a desired final space. Many variations of this puzzle exist. In this paper, we establish formal notation regarding rolling cube mazes and solve a simple variant: find a shortest path that puts a desired label on top at the final space. Utilizing several symmetries and reductions, we then produce a description of the solution path in constant time. This provides a framework for future researchers to develop algorithms to efficiently solve more complex mazes.

1 Introduction

Rolling cube puzzles were first popularized by Martin Gardner [3]. They consist of a labelled cube on a board with some task in mind. Mathematician Robert Abbott built on this to create a "Rolling Cube Maze", which considers an initial space and a final space, and asks to find a path to the final space. Rolling Cube Mazes have many variations, two of which are shown in Figure 1. In the left image, every space is labelled and a condition is applied such that when the cube lands on that space, the space's label must be face up before flipping onto that space (the spaces with asterisk mean any label is allowed). In the right image, there are no labelled spaces, but instead an initial and final space for the cube to start and end on respectively.

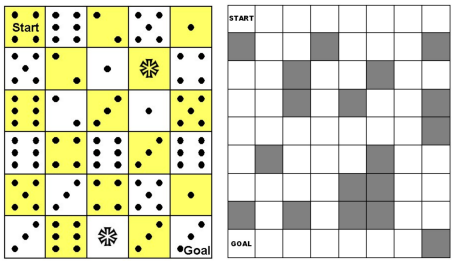

Figure 1: Example Rolling Cube Mazes (Images from Buchin et al. [2])

*Department of Computer Science, University of North Carolina at Chapel Hill, rtuggle99@gmail.com
†Department of Mathematics, Berry College, dkmurphy@outlook.com
‡Department of Statistics, University of Georgia, lorchnd@gmail.com

The rolling cube mazes we consider in this paper have a labelled final space and no blocked spaces. The aim is to find a sequence of moves that takes a cube from an initial position (x_i, y_i) to a final position (x_f, y_f) in the fewest moves such that the cube visits (x_f, y_f) only on the final move and the desired label ℓ ends on top. An important distinction to note is that in our problem, the final label must be face up after flipping onto the final space, not before.

In Section 2, we define notation. In Section 3, we present four techniques that allow us to greatly simplify the problem. In Section 4, we describe solutions for (x_i, y_i) and (x_f, y_f) that are sufficiently far apart. In Section 5, we describe solutions for all other (x_i, y_i) and (x_f, y_f). Finally, in Section 6, we prove that the complexity of our algorithm is O(1). We can formally define the problem as follows:

> **Problem:** Simplified Rolling Cube (SRC).
> **Instance:** board height m, board width n, initial space (x_i, y_i), final space (x_f, y_f), and desired final label ℓ, with the assumption that the cube starts in the standard orientation described in Section 2.1.
> **Solution:** A string description of moves that takes the cube from initial space (x_i, y_i) to final space (x_f, y_f) with desired label ℓ on top in the fewest moves without crossing over (x_f, y_f), or False if there is no solution.

2 Notation

2.1 Describing Faces and Assigning Labels

First, we name the faces of our cube according to the net provided. We define the North face to be the face that points North, the East face to be the face that points East, and so on. Then we define the Top face and Bottom face to be the face pointing away from and touching the board respectively. We assign labels to the starting faces of our cube according to the labeling of a standard right-handed die and, without loss of generality, create a standard starting orientation, pictured in Figure 2:

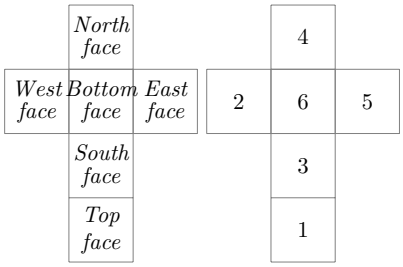

Figure 2: Face descriptions and starting labels

2.2 Describing Moves and Paths

We define North (N), South (S), East (E), and West (W) moves as flipping the cube onto the space immediately north/south/east/west of the cube's current position respectively. We also define an identity (I) move which leaves the cube in its current orientation and position. We represent a sequence of moves from one space to another as a Generalized Path String, or GPS. We can use the following grammar rules to define a GPS with Z being the start symbol

$$\begin{aligned} Z &\rightarrow (M)\{EXP\}Z \mid MZ \mid \epsilon \\ M &\rightarrow MM \mid I \mid N \mid E \mid S \mid W \\ EXP &\rightarrow \Delta + D \mid \Delta - D \mid D \\ \Delta &\rightarrow \Delta_x \mid \Delta_y \\ D &\rightarrow DD \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

We define $\Delta_x = |x_f - x_i|$ and $\Delta_y = |y_f - y_i|$. When we return the string, however, the string contains the literal characters "Δ_x" and "Δ_y" rather than the numbers they represent. We do this because Δ_x and Δ_y can be arbitrarily large, and we want the length of the string to be bounded by a constant. Δ is either Δ_x or Δ_y. EXP is an expression of the form Δ plus or minus some $D \in \mathbb{N}$ (our algorithm never uses $D > 4$). EXP evaluates to some number $d \in \mathbb{N}$. M is simply any sequence of the five basic moves described above. When we have $(M)\{EXP\}$, we take this to mean that we perform the moves M in parentheses consecutively d times.

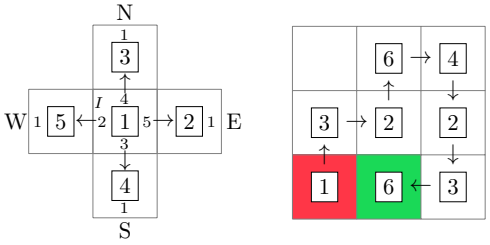

Figure 3: The moves N, E, S, W, I, and the path 'NENESSW' visualized

3 Simplifying the Problem

In this section, we introduce several techniques and a partitioning that together simplify our problem significantly. The first technique is "face-saving" which allows us to keep track of our desired final top label over arbitrarily long distances. The second technique is "Quadrant Mapping" which allows us to assume that $x_f \geq x_i$ and $y_f \geq y_i$. The third technique is a series of reductions which allows us to focus only on solving for $\ell = 1$ or $\ell = 6$. Following these techniques, we partition the displacements into two sets, large and small, which we handle differently.

3.1 Face Saving

In later proofs, we utilize the technique of "saving" the desired final top label onto one of the two faces that are unchanged by moving only along a single axis. This allows us to move the cube an arbitrary number of moves in either direction along that axis and still know exactly the face on which the desired final label is saved.

Definition 1 *A label ℓ is said to be **saved with respect to an axis** A if and only if moving along A keeps label ℓ on the same face.*

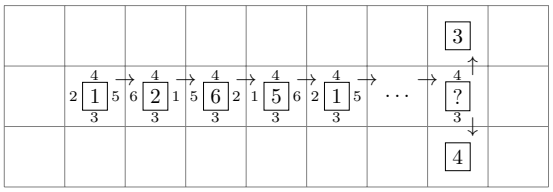

Figure 4: Saving 4 on the North face with respect to the E-W axis

3.2 Quadrant Mapping

We say (x_f, y_f) is in quadrant 1 if $x_f \geq x_i$ and $y_f \geq y_i$, in quadrant 2 if $x_f < x_i$ and $y_f \geq y_i$, in quadrant 3 if $x_f < x_i$ and $y_f < y_i$, and in quadrant 4 if $x_f \geq x_i$ and $y_f < y_i$.

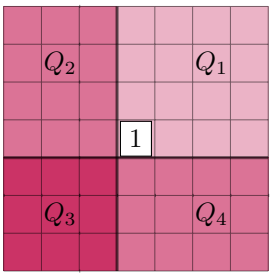

Figure 5: Quadrants

We can perform reflections to create a (x'_f, y'_f) and ℓ' that fall in quadrant 1 via the following steps:

- Step 1: start at (x_i, y_i) in the standard orientation and show (x_f, y_f) with desired label ℓ where (x_f, y_f) is
- Step 2: reflect the board across N-S axis if (x_f, y_f) began in $Q3$ or $Q4$
- Step 3: reflect the board across E-W axis if (x_f, y_f) began in $Q3$ or $Q2$
- Step 4: relabel cube to be in standard orientation and find ℓ' corresponding to the new labeling.

As seen in the pseudo code for *mapToQuad*1 in the appendix, we can define a tuple $q = (x_i < x_f, y_i < y_f)$ in which the first or second element of q is true if we are reflecting over the N-S or E-W axis respectively. After we generate our GPS, we can switch the E's with W's and N's with S's as needed to find an analogous path for the original quadrant. This conversion can be seen in the pseudo code for *convGPS* in the appendix.

3.3 Reductions

We can reduce the number of cases by noting certain symmetries. First, any path to (x_f, y_f) ending on $\ell = 2$ on an $m \times n$ board is analogous to a path to (y_f, x_f) ending on $\ell = 3$ on an $n \times m$ via swapping $N's$ with $E's$ and $S's$ with $W's$. The $\ell = 4$ and $\ell = 5$ cases share the same symmetry. The pseudo code for handling labels 2 and 5 is shown in the appendix. In the remainder of this section, we reduce the $\ell = 3, 4$ cases to either the $\ell = 1$ or $\ell = 6$ case.

Lemma 1 *Let $\ell = 3, 4$. For any GPS G that places ℓ on top in k moves, there exists a GPS that begins with $(E)\{i\}N$ or $(E)\{i\}S$ or $(W)\{i\}N$ or $(W)\{i\}S$ for some $i \in \{0, 1, 2, 3\}$ that also places ℓ on top at (x_f, y_f) in k moves.*

The idea for the reduction is that if there is some GPS G that places ℓ on top at (x_f, y_f), then there exists some GPS that begins with $i \in \{0, 1, 2, 3\}$ E or W moves followed by a N or S move that places ℓ on the top or bottom at some (x_r, y_r) and places ℓ on top at (x_f, y_f) in the same amount of moves as G.

Theorem 2 *The $\ell = 3$ and $\ell = 4$ cases can reduce to the $\ell = 1, 6$ cases in constant time*

3.4 Displacement Types

For ease of analyzing displacements, we denote the displacements before the reduction to $\ell = 1$ or $\ell = 6$ as Δ_x and Δ_y and the displacements after the reduction as Δ'_x and Δ'_y. That is, $\Delta_x = \Delta'_x + \delta_x$ and $\Delta_y = \Delta'_y + \delta_y$ for some natural numbers δ_x, δ_y. Furthermore, we define (x_r, y_r) to be the initial space after the reduction.

We denote the bottom left square of the $m \times n$ boards as $(1, 1)$. Note that if we ignore ending labels, going from (x_r, y_r) to (x_f, y_f) takes at least $|x_r - x_f| + |y_r - y_f|$, or $(\Delta'_x + \Delta'_y)$, moves. Unfortunately, finding a path to (x_f, y_f) with label ℓ on top in $(\Delta'_x + \Delta'_y)$ moves is not always possible.

For $\ell = 1, 6$, we define threshold values $\Delta_{row}, \Delta_{col}$ for Δ'_y, Δ'_x in Table 1 to separate our problem into cases requiring different GPS templates. Note that there are two sets of threshold values for $\ell = 6$. This is because we define two possible paths for $\ell = 6$ in Section 4, one starting with a N move and one starting with an E.

ℓ	# of rows apart (Δ_{row})	# of cols apart (Δ_{col})
1	2	2
6_N	4	2
6_E	2	4

Table 1: Threshold values for displacement categories

We can now define the large displacements (Section 4) to be the cases where $\Delta'_y \geq \Delta_{row}$ and $\Delta'_x \geq \Delta_{col}$ and small displacements to be all remaining cases.

4 Large displacements

To begin, we list the shortest string of moves required to get $\ell = 1, 6$ saved on either the North or East face:

ℓ	Prefixes	Face ℓ is on
1	N or E	North or East
6	EEN or NNE	North or East

Table 2: Prefixes to use for $\ell = 1, 6$

Once ℓ is saved on either the North face or the East face on some space, we can follow one of the paths depicted in the figure below:

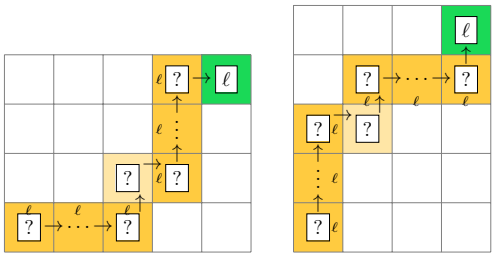

Figure 6: Large-displacement method once ℓ is saved on the North face (left) or East face (right)

Then, to generate a GPS, we can apply the aforementioned prefix to the path found from this face saving process. Doing this, we can describe the solutions for large displacements:

ℓ	Generalized Path Strings	
1	N	$(E)\{\Delta'_x - 2\}NE(N)\{\Delta'_y - 2\}E$
6_N	NNE	$(N)\{\Delta'_y - 4\}EN(E)\{\Delta'_x - 2\}N$
6_E	EEN	$(E)\{\Delta'_x - 4\}NE(N)\{\Delta'_y - 2\}E$

Table 3: Large Displacement GPS's

An important observation is that the number of moves in these GPS's is always $\Delta'_x + \Delta'_y$. Thus, these GPS's use the fewest moves to get from (x_r, y_r) to (x_f, y_f) with ℓ on top.

5 Small Displacements

When dealing with large displacements we did not need to worry about potentially rolling off the board or using a GPS that contains a value such as $\Delta'_x - 2$ which may now be negative. Thus, we must come up with a different way to find solutions when $\Delta'_x < \Delta_{col}$ or $\Delta'_y < \Delta_{row}$.

5.1 Breadth First Search Approach

One possible approach is to check all possible paths to (x_f, y_f) using a brute force algorithm and pick the shortest one. Using this approach, it would be impossible to find solutions in $O(1)$ time since since the complexity depends on m and n. So we do not use this approach in our actual algorithm. However, we use this approach to prove that some of our results are in fact solutions of fewest moves. In order to brute force all paths, we create a graph with one vertex for each possible position and orientation combination, or *state* [2], of a cube on a given board. Edges are between two vertices if their corresponding states are one N, E, S, or W move apart.

5.2 Systematic Approach

We begin our systematic approach by proving the following Theorem.

Theorem 3 *Assume there exists a GPS that begins at (x_i, y_i) on an $m \times n$ board and places ℓ on top at $(x_i + \Delta_x, y_i + \Delta_y)$ in $\Delta_x + \Delta_y + 2k$ moves for some $k \in \mathbb{N}$. Then for all (x_c, y_c) such that $x_c \geq k$ and $y_c \geq k$ on any board large enough to allow (x_c, y_c) and $(x_c + \Delta_x, y_c + \Delta_y)$ to exist, all GPS's from (x_c, y_c) to $(x_c+\Delta_x, y_c+\Delta_y)$ that place ℓ on top in the fewest moves are contained within an $(\Delta_y + 2k) \times (\Delta_x + 2k)$ rectangle such that the bottom left corner of this rectangle is $(x_c - k, y_c - k)$ and the upper right corner is $(x_c + \Delta_x + k, y_c + \Delta_y + k)$.*

The idea with Theorem 3 is that once we have a GPS for a given Δ'_x and Δ'_y, we can get an upper bound for the size of boards we need to check in order to find a GPS of fewest moves for the given Δ'_x and Δ'_y regardless of board size.

5.2.1 Symmetry

Because $\ell = 1, 6$ begins on the top and bottom faces respectively, any path to (x_f, y_f) with $\Delta'_y > \Delta'_x$ is analogous to a path to (y_f, x_f) with $\Delta'_x > \Delta'_y$, just by swapping Ns with Es and Ss with Ws. Therefore, for the following cases, we assume $\Delta'_x \geq \Delta'_y$.

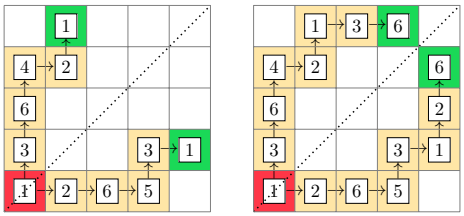

Figure 7: $\ell = 1$ (left) and $\ell = 6$ (right) GPS symmetry about diagonal

We now handle the $\ell = 1, 6$ small displacement cases. We begin by defining the function χ such that $\chi(\ell)$ is the number of E moves required to place ℓ on top from the starting orientation. For $\ell = 1, 6$ we find that $\chi(1) = 0$ and $\chi(6) = 2$. These are the main cases we use for small displacements:

1. $\Delta'_x = 0$ and $\Delta'_y = 0$
2. $\Delta'_x \equiv_4 \chi(\ell)$
3. $\Delta'_y = 0$
4. $\Delta'_y = 1$
5. $\Delta'_y = 2$
6. $\Delta'_y = 3$

To handle the small displacement cases we go through the above enumeration in order, handling a case only if the previous case has not been met.

5.2.2 $\Delta'_x = 0$ and $\Delta'_y = 0$

When $\ell = 1$, return I. Our formal statement of SRC allows the cube to be on the final space only when the correct label is on the top face, so return False when $\ell = 6$.

5.2.3 $\Delta'_x \equiv_4 \chi(\ell)$

By the definition of χ, using $4k + \chi(\ell)$ for all $k \in \mathbb{Z}$ consecutive E moves places ℓ on top when our cube is in starting orientation. Thus, when $\Delta'_x \equiv_4 \chi(\ell)$, a solution can be obtained by rolling to $(x_f - 1, 1)$ to save ℓ on the West face, then performing Δ'_y N moves, and finally placing ℓ on top via an E move as shown in Figure 8.

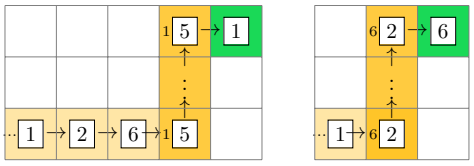

Figure 8: $\ell = 1$ (left) and $\ell = 6$ (right) when $\Delta'_x \equiv_4 \chi(\ell)$

5.2.4 $\Delta'_y = 0$

Theorem 4 *All paths from (x_i, y_i) to (x_f, y_f) require $\Delta_x + \Delta_y + 2k$ moves for some $k \in \mathbb{N}$.*

Theorem 5 *When $m > 1$, $\Delta'_y = 0$, $\Delta'_x > \chi(\ell)$, and $\Delta'_x \not\equiv_4 \chi(\ell)$, the shortest path from (x_r, y_r) to (x_f, y_f) with final label $\ell = 1, 6$ is exactly $\Delta'_x + \Delta'_y + 2$ moves.*

It follows that when $\ell = 1$, going either N or S then E as far as needed and then finally S or N, we get a GPS of fewest moves. We can also do something similar for $\ell = 6$ as shown in Figure 9:

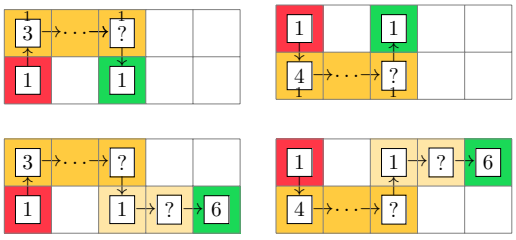

Figure 9: $\ell = 1$ (top) and $\ell = 6$ (bottom) when $m > 1$, $\Delta'_x \not\equiv_4 \chi(l)$ and $\Delta'_y = 0$

When $\ell = 6$ there is not room to do this when $\Delta'_x = 1$. Using the brute force algorithm, we checked on boards up to size 7×8 and found that the GPS's listed in *small1case4* and *small6case4* in the appendix were the shortest paths that place ℓ on top at (x_f, y_f). We know by Theorem 3 that these are the shortest paths.

5.2.5 $\Delta'_y = 1$

Theorem 6 *For the $\ell = 1, 6$ small displacement case, when $\Delta'_y = 1$, $\Delta'_x > 1$ and $\Delta'_x \not\equiv_4 \chi(\ell)$, there is no path that places ℓ on top at (x_f, y_f) in $\Delta'_x + \Delta'_y$ moves.*

Notice that the paths in Figure 10 place ℓ on top in $\Delta'_x + \Delta'_y + 2$ moves. Therefore, by Theorem 6, these GPS's are the shortest paths that place ℓ on top at (x_f, y_f) for $\ell = 6$ when $\Delta'_x > 1$ and for $\ell = 1$ when $\Delta'_x > 3$.

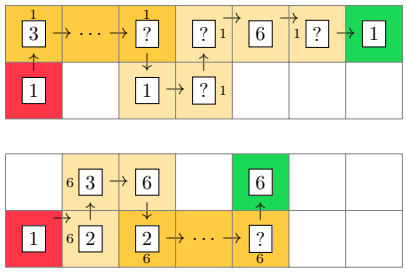

Figure 10: $\ell = 1$ (left) and $\ell = 6$ (right) when $m > 1$, $\Delta'_x \not\equiv_4 \chi(\ell)$ and $\Delta'_y = 1$

What remains of the $\Delta'_y = 1$ case is when $\ell = 1$ and $\Delta'_x \leq 3$ or when $\ell = 6$ and $\Delta'_x = 1$. Applying our brute force algorithm, we found the paths required $\Delta'_x + \Delta'_y + 6$ moves when $\ell = 1, 6, \Delta'_x = 1$. Thus by Theorem 3, we need to check only boards up to size 8×8 to find solutions of fewest moves.

5.2.6 $\Delta'_y = 2$

Note that if $\ell = 1$, we are not in the small displacement case. Therefore, $\ell = 6$. We can use the GPS depicted in Figure 11:

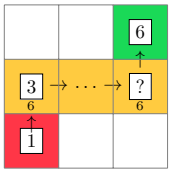

Figure 11: $\ell = 6$ when $\Delta'_y = 2$

5.2.7 $\Delta'_y = 3$

The only time we are in this case is when $\ell = 6$ and $\Delta'_x = 3$. The GPS depicted in Figure 12 is of fewest moves:

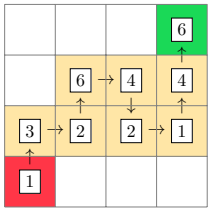

Figure 12: $\ell = 6$ when $\Delta'_y = \Delta'_x = 3$

5.2.8 Putting the cases together

The pseudo code *small*1 and *small*6 put the small displacement cases together via the helper functions seen in the appendix.

6 Finalized Approach

The main approach to solving a rolling cube maze is as follows. Reduce inputs to a quadrant 1 problem, reduce the inputs to a $\ell = 1$ or $\ell = 6$ problem, determine the displacement type, create a GPS, and invert the quadrant mapping to return a final GPS. Pseudo code can be seen in the appendix.

Theorem 7 *The length of the GPS returned by SRC is bounded by a constant and is generated in constant time*

7 Future Work

We believe that future work can now easily be done on other rolling cube mazes by utilizing the fact that a path from any space to any other space with a desired label can be generated in constant time. We implemented SRC and created the following visualization available on github to aid any potential researchers interested in exploring mazes with blocked spaces.

8 Acknowledgements

We would like to thank Professor Jack Snoeyink for reading our drafts and for giving helpful feedback.

References

[1] Robert Abbott SuperMazes: Mind Twisters for Puzzle Buffs, Game Nuts, and Other Smart People. *Prima Publishing*, 1997.

[2] Kevin Buchin and Maike Buchin and Erik D. Demaine and Martin L. Demaine and Dania El-Khechen and Sandor Fekete and Christian Knauer and André Schulz and Perouz Taslakian On rolling cube puzzles. *In Proc. 19th Canad. Conf. Comput. Geom.*, pages 141–148, 2007.

[3] Martin Gardner Mathematical games column. *Scientific American*, 209(6):144, 1963.

[4] Jiawei Yao Research on the Minimum Moves of Rolling Cube Puzzles *JAIST*, 2021.

Appendix

Proofs

Proof. (Lemma 1) Assume $\ell = 3, 4$ and assume some GPS G gets ℓ on top in k moves. Since ℓ is saved with respect to the $E - W$ axis, there must be a N or S in G in order to get ℓ on top. Without loss of generality, assume a N move exists in G. The N move is performed after some amount of E and/or W moves. As stated previously, an E move can be thought of as a negative W move. Thus, the sequence of E and W moves prior to the first N move can be written entirely as E's or entirely as W's. Without loss of generality, assume the moves can be represented entirely by E's. That is, the GPS G can be written as $(E)\{c\}NG_s$ for some c where G_s is the substring of G after the first N move.

We can write $c \equiv_4 i$ for some $i \in \{0, 1, 2, 3\}$. Note $(E)\{c\}$ will keep ℓ face saved. Then N will put $\ell = 3, 4$ on either the top or bottom respectively. Note that if we instead did $(E)\{i\}N$ first, the same would be true. Then $(E)\{c - i\}$ would keep $\ell = 3$ on top and $\ell = 4$ on bottom because every four E moves puts the labels back on the faces they started on. Thus the GPS $G = (E)\{c\}NG_s$ can be written as $(E)\{i\}N(E)\{c - i\}G_s$.

An analogous argument could have been made if G used W moves rather than E moves at the beginning. Furthermore, an analogous argument could be made if the move before G_s were a S move rather than a N move. ∎

Proof. (Theorem 2) As noted in the proof of Lemma 1, when $\ell = 3$, a N or S move must be performed at some point in order to get label ℓ on top. Note that when a first N move is performed, $\ell = 3$ is on top at some $(x_r, y_i + 1)$ and this can be treated as an instance of the $\ell = 1$ case, which is $SRC(m, n, x_r, y_i + 1, x_f, y_f, 1)$. Similarly, note that when a first S move is performed, $\ell = 3$ is on bottom and this can be treated as an instance of the $\ell = 6$ case. By Lemma 1, it follows that we only need to consider any paths that begin with 0,1,2, or 3 consecutive E's/W's followed by a N or S. Since we perform moves before reducing to $\ell = 1$ or $\ell = 6$, we are changing what Δ_x and Δ_y are. To take care of this, we can define a helper function *SRCReduction* which takes the same inputs as *SRC* along with two inputs δ_x and δ_y so that the number of moves in the returned GPS is correct. Also, since the E moves or W moves may change the quadrant (x_f, y_f) is in, we need to do quad mapping again before *SRCReduction* handles the $\ell = 1, 6$ case. This is not a problem because when $\ell = 1, 6$, quad mapping does not change the desired label. We will have that δ_x is the difference of $|x_f - x_i|$ and $|x_f - x_r|$ and δ_y is the difference of $|y_f - y_i|$ and $|y_f - y_r|$ where (x_r, y_r) is the position of the cube when the reduction is called. The body of *SRCReduction* can be seen in the appendix. Following these results, we get the pseudo code *handle*3 in the appendix, which handles the $\ell = 3$ case. Since $\ell = 3$ reduces to the $\ell = 1$ or $\ell = 6$ case and the $\ell = 2$ case is analogous to the $\ell = 3$ case, it follows that the $\ell = 2$ case reduces to the $\ell = 1$ or $\ell = 6$ case. By an analogous argument, we can handle the $\ell = 4, 5$ cases, as seen by the pseudo code for *handle*4 in the appendix. ∎

Proof. (Theorem 3) Assume there exists a GPS starting at (x_i, y_i) and placing ℓ on top at (x_f, y_f) in $\Delta_x + \Delta_y + 2k$ moves on an $m \times n$ board. Now consider a GPS G of fewest moves from (x_c, y_c) to $(x_c + \Delta_x, y_c + \Delta_y)$ ending with the same ℓ on top, for some $x_c > k$ and $y_c > k$. Assume for the sake of contradiction that G requires moving west or south of $(x_c - k, y_c - k)$ or north or east of $(x_c + \Delta_x + k, y_c + \Delta_y + k)$. Without loss of generality, assume G requires moving west of $(x_c - k, y_c - k)$. That is, there exists a position $(x_c - k - 1, y_d)$ for some y_d that is visited by G. Note that at a minimum,

this requires $|(x_c)-(x_c-k-1)| = k+1$ moves. Then, at the very minimum, going from this space (x_c-k-1, y_d) to the ending space $(x_c+\Delta_x, y_c+\Delta_y)$ requires $(x_c+\Delta_x)-(x_c-k-1)+(y_c+\Delta_y)-(y_d) = \Delta_x+\Delta_y+k+1+y_c-y_d$ moves. Thus in total G requires $(k+1)+(\Delta_x+\Delta_y+k+1+y_c-y_d) = \Delta_x+\Delta_y+2(k+1)+y_c-y_d \geq \Delta_x+\Delta_y+2(k+1)+0$ moves. Thus G requires more than $\Delta_x+\Delta_y+2k$ moves. This is a contradiction because we assumed G required the fewest moves and we already know a path of $\Delta_x+\Delta_y+2k$ exists.

Proof. (Theorem 4) This statement is equivalent to Theorem 1 in "Research on the Minimum Moves of Rolling Cube Puzzles" [4].

Proof. (Theorem 5) Consider a path from (x_r, y_r) to (x_f, y_f) with final label $\ell = 1, 6$ on a board with $m > 1$, such that $\Delta_y = 0$, $\Delta'_x > \chi(\ell)$ and $\Delta'_x \not\equiv_4 \chi(\ell)$. Since $\Delta'_y = 0$ and $\Delta'_x \not\equiv_4 \chi(\ell)$, in order to place ℓ on top at (x_f, y_f), a N and S move must be performed eventually. Thus, a solution must have more than $\Delta'_x + \Delta'_y$ moves. By Theorem 4, then, we know any solution must be at least $\Delta_x + \Delta_y + 2$ moves. Because $m > 1$, there is either a row above (x_r, y_r) or below (x_r, y_r). If there exists a row above (x_i, y_i), the GPS $N(E)\{\Delta'_x - \chi(\ell)\}S(E)\{\chi(\ell)\}$ places ℓ on top in $\Delta_x + \Delta'_y + 2$ moves. Similarly, if there exists a row below (x_r, y_r), the GPS $S(E)\{\Delta'_x - \chi(\ell)\}N(E)\{\chi(\ell)\}$ places ℓ on top in $\Delta'_x + \Delta'_y + 2$ moves. Thus, the shortest path under these conditions is exactly $\Delta'_x + \Delta'_y + 2$ moves.

Proof. (Theorem 6) Consider the $\ell = 1, 6$ small displacement case and assume $\Delta'_y = 1$, $\Delta'_x > 1$ and $\Delta'_x \not\equiv_4 \chi(\ell)$. Note that a path to (x_f, y_f) using $\Delta'_x + \Delta'_y$ moves must have exactly 1 N move, which will come after c E moves and be followed by $\Delta'_x - c$ E moves. If $c \equiv_4 0, 2$, we find that $\ell = 1$ will be saved on the North or South faces, respectively, and $\ell = 6$ will be saved on the South or North faces, respectively. This means that the remaining E moves cannot get $\ell = 1, 6$ on the top face. If $c \equiv_4 1, 3$, we see that $\ell = 1$ will be on the East and West faces, respectively, and remain there after the N move. Similarly, $\ell = 6$ will be on the West and East faces, respectively, and will also remain there after the N move. For both $c \equiv_4 1, 3$, the label ℓ will only be on top when the total number Δ'_x of E moves is of the form $4k + \chi(\ell)$. Since we know by initial assumption that this is not true of Δ'_x, we see that using only $\Delta'_x + \Delta'_y$ moves leaves us unable to place $\ell = 1$ or $\ell = 6$ on top at (x_f, y_f). Thus, by Theorem 4, we need at least $\Delta_x + \Delta'_y + 2$ moves to place $\ell = 1, 6$ on top at (x_f, y_f).

Proof. (Theorem 7) To avoid the returning a string whose length depends on the number of digits in $|x_f - x_i|$ and $|y_f - y_i|$, we return a GPS that uses the literal characters Δ_x and Δ_y. Note further that δ_x and δ_y are at most 3, so that is why we can use the numbers they represent rather than also using the literal characters. Therefore, the length of the GPS returned is bounded by a constant.
We have shown that we perform quadrant mapping and reduce any instance of the problem to the $\ell = 1, 6$ cases in constant time. Clearly, the large displacement case can be handled in constant time. As for the small case, as stated in Section 5.1, the complexity of the brute force algorithm used in some of our proofs depends on m and n. However, this algorithm is only necessary for a finite number of scenarios. We have provided the output for each of these scenarios and thus our proposed algorithm finds these solutions by performing a simple lookup. Finally, we can invert the quadrant mapping in constant time. Therefore, the final GPS is generated in constant time.

Algorithms

```
def mapToQuad1 (m,n,xi,yi,xf,yf,ℓ):
    set newL = ℓ and set q = [0,0]
    if xf < xi:
        q[0] = 1
        if ℓ = 2:
            newL = 5
        elif ℓ = 5
            newL = 2
    if yf < yi:
        q[1] = 1
        if ℓ = 3:
            newL = 4
        elif ℓ = 4
            newL = 3
    newXi = (n - xi) + 1 if q[0] == 1 else  xi
    newYi = (m - yi) + 1 if q[1] == 1 else  yi
    newXf = (n - xf) + 1 if q[0] == 1 else  xf
    newYf = (m - yf) + 1 if q[1] == 1 else  yf
    return (newXi ,newYi),(newXf,newYf),newL,q

def convGPS (q,GPS):
    if q[0] == 1
        switch all Es and Ws in GPS
    if q[1] == 1
        switch all Ns and Ss in GPS
    return GPS

def handle2(m,n,xi,yi,xf,yf):
    return handle3(n,m,yi,xi,yf,xf) swapping Ns with Es, Ss with Ws,
    and Δx with Δy

def handle5(m,n,xi,yi,xf,yf):
    return handle4(n,m,yi,xi,yf,xf) swapping Ns with Es and Ss with Ws,
    and Δx with Δy

def SRCReduction(m,n,xr,yr,xf,yf,ℓ,δx,δy):
    (m,n,xr,yr,xf,yf,ℓ,q) = mapToQuad1(m,n,xr,yr,xf,yf,ℓ)
    if q[0] == 1:
        δx = −δx
    if q[1] == 1:
        δy = −δy
    if ℓ == 1:
        GPS =  handle1(m,n,xr,yr,xf,yf,δx,δy)
    elif ℓ == 6:
        GPS =  handle6(m,n,xr,yr,xf,yf,δx,δy)
    return ConvGPS(q,GPS)

def handle3(m,n,xi,yi,xf,yf)
    initialize a list of paths
    for i in [0,1,2,3]:
        path1 = (E){i}N +
        SRCReduction(m,n,xi + i,yi+1,xf,yf,1,
        ||xf − xi| − |xf − (xi + i)||, ||yf − yi| − |yf − (yi + i)||)
        path2 = (E){i}S +
        SRCReduction(m,n,xi + i,yi−1,xf,yf, 6,
        ||xf − xi| − |xf − (xi + i)||, −||yf − yi| − |yf − (yi − i)||)
        path3 = (W){i}N +
        SRCReduction(m,n,xi - i,yi+1,xf,yf,1,
        −||xf − xi| − |xf − (xi − i)||, |yf − yi| − |yf − (yi + i)|)
        path4 = (W){i}S +
        SRCReduction(m,n,xi - i,yi−1,xf,yf,6,
        −||xf − xi| − |xf − (xi − i)||, −||yf − yi| − |yf − (yi − i)||)
        add each path to the list of paths
return the computed path of fewest moves or false if none exist

def handle4(m,n,xi,yi,xf,yf):
    initialize a list of paths
    for i in [0,1,2,3]:
        path1 = (E){i}N +
        SRCReduction(m,n,xi + i,yi+1,xf,yf,6,
```

```
        ||x_f − x_i| − |x_f − (x_i + i)||, ||y_f − y_i| − |y_f − (y_i + i)||)
        path2 = (E){i}S +
        SRCReduction(m,n,xi + i,yi-1,xf,yf,1,
        ||x_f − x_i| − |x_f − (x_i + i)||, −||y_f − y_i| − |y_f − (y_i − i)||)
        path3 = (W){i}N +
        SRCReduction(m,n,xi - i,yi+1,xf,yf,6,
        −||x_f − x_i| − |x_f − (x_i − i)||, ||y_f − y_i| − |y_f − (y_i + i)||)
        path4 = (W){i}S +
        SRCReduction(m,n,xi - i,yi-1,xf,yf,1,
        −||x_f − x_i| − |x_f − (x_i − i)||, −||y_f − y_i| − |y_f − (y_i − i)||)
        add each path to the list of paths
return the computed path of fewest moves or false if none exist

def handle1(m,n,xr,yr,xf,yf,δ_x,δ_y):
        dxPrime = xf - xr, dyPrime = yf - yr
        if dxPrime > 1 and dyPrime > 1:
            return the large displacement template
        else:
            return the small displacement template

def handle6(m,n,xr,yr,xf,yf,δ_x,δ_y):
        dxPrime = xf - xr,  dyPrime = yf - yr
        if dxPrime > 3 and dyPrime > 1 or dxPrime > 1 and dyPrime > 3:
            return the large displacement template
        else:
            return the small displacement template

def large1(δ_x,δ_y):
    return N(E){Δ_x - δ_x- 2}NE(N){Δ_y - δ_y - 2}E

def large6(dxPrime,dyPrime,δ_x,δ_y):
    if  dxPrime > 1 and dyPrime > 3:
        return NNE(N){Δ_y - δ_y - 4}EN(E){Δ_x - δ_x - 2}N
    elif dxPrime > 3 and dyPrime > 1:
        return  EEN(E){Δ_x - δ_x - 4}NE(N){Δ_y - δ_y - 2}E

def small1symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime)
    answer = small1(n,m,yr,xr,yf,xf,dyPrime,dxPrime,δ_y ,δ_x )
    return answer but switch Ns with Es and Ss with Ws and Δ_x with Δ_y

def small6symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime)
    answer = small6(n,m,yr,xr,yf,xf,dyPrime,dxPrime,δ_y ,δ_x )
    return answer but switch Ns with Es and Ss with Ws and Δ_x with Δ_y

def small1case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime): return I

def small6case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime): return False

def small1case2(m,n,xr,yr,xf,yf,dx,dy,δ_x,δ_y)
    return (E){Δ_x - δ_x - 1}(N){Δ_y - δ_y}E

def small6case2(m,n,xr,yr,xf,yf,dx,dy,δ_x,δ_y)
    return (E){Δ_x - δ_x - 1}(N){Δ_y - δ_y}E

def small1case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    if yf < m:
        return N(E){Δ_x - δ_x}S
    elif yr > 1:
        return S(E){Δ_x - δ_x}N
    else:
        return False

def small6case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    if m == 1:
        return False
    elif dxPrime > 1 and yf < m:
        return N(E){Δ_x - δ_x - 2}SEE
    elif dxPrime >  1 and yr > 1:
        return S(E){Δ_x - δ_x - 2}NEE
    elif dxPrime == 1:
        if xr > 1:
            return NWSEE
        elif yf < m - 1:
            return NENWSSE
        elif yr > 2
            return SESWNNE
        elif xr < n - 1 :
            if yf < m:
                return NEEESWW
            elif yr > 1:
                return SEEENWW
            elif (xr > 1 and yr > 1):
                return SWNESEENW
            else:
                return False

def small1case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    if dxPrime > 3:
        return N(E){Δ_x - δ_x - 4}SENEEE
    elif dxPrime == 2 or dxPrime == 3:
        if yf < m:
            return N(E){Δ_x - δ_x - 2}NESE
        elif yr > 1:
            return ESEN(E){Δ_x - δ_x - 2}N
        else:
            return ENWS(E){Δ_x - δ_x}N
    elif dxPrime == 1:
        if yr > 1:
            return SENWNE
        elif xr > 1:
            return WNESEN
        elif yf < m - 1:
            return NNENWSES
        elif xf < n - 1:
            return EENESWNW
        else:
            return False

def small6case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime, δ_x, δ_y)
    if dxPrime > 1:
        return ENES(E){Δ_x - δ_x - 2}N
    elif dxPrime == 1:
        if yr > 1:
            return ESWNEN
        elif xr > 1:
            return NWSENE
        elif yf < m-2:
            return NNNESWSE
        elif xf < n-2:
            return EEENWSWN
        else:
            return False

def small6case5(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    return N(E){Δ_x - δ_x}N

def small6case6(m,n,xi,yi,xf,yf,dx,dy)
    return NENESENN

def small1(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        if dyPrime > dxPrime:
                return small1symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif (dxPrime,dyPrime) == (0,0):
                return small1case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dxPrime == 0 mod 4:
                return small1case2(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 0:
                return small1case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 1:
                return small1case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)

def small6(m,n,xr,yr,xf,yf,dx,dy,δ_x,δ_y):
        if dyPrime > dxPrime:
            return small6symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif (dxPrime,dyPrime) == (0,0):
                return small1case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dxPrime == 2 mod 4:
            return small6case2(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 0:
            return small6case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 1:
            return small6case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime.δ_x,δ_y)
        elif dyPrime == 2:
            return small6case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 3:
            return small6case6(m,n,xr,yr,xf,yf,dxPrime,dyPrime, δ_x,δ_y)

def SRC(m,n,xi,yi,xf,yf,ℓ):
        xi,yi,xf,yf,ℓ,q = mapToQuad1(m,n,xi,yi,xf,yf,ℓ)
        if ℓ == 1:
                GPS = handle1(m,n,xi,yi,xf,yf,0,0)
        elif ℓ == 2:
                GPS = handle2(m,n,xi,yi,xf,yf)
        elif ℓ == 3:
                GPS = handle3(m,n,xi,yi,xf,yf)
        elif ℓ == 4:
                GPS = handle4(m,n,xi,yi,xf,yf)
        elif ℓ == 5:
                GPS = handle5(m,n,xi,yi,xf,yf)
        elif ℓ == 6:
                GPS = handle6(m,n,xi,yi,xf,yf,0,0)
        return ConvGPS(q,GPS)
```