Master's Thesis

Research on the Minimum Moves of Rolling Cube Puzzles

Jiawei Yao

Supervisor Ryuhei Uehara

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March 2021

Abstract

Rolling cube puzzles are a puzzle game that rolls the die which is placed on a board to complete a goal followed certain rules. A universal goal is to roll the die which is at the upper left corner of the board over all labeled cells of the board exactly once. When the die rolls over a labeled cell, the number on the top face of the die must be the same as the number of the cell on which it lies.

Rolling cube puzzles have a long history. Rolling cube puzzles were popularized by Martin Gardner in his mathematical games columns, which is published in *Scientific American*. In 2007, Kevin Buchin et al. researched the rolling cube puzzles and proved that solving rolling cube puzzles is NP-complete where rolling cube puzzles are defined by a six-sided die on a square grid board. However, there are several related puzzles and open problems posed in the paper by Kevin Buchin et al.

In this research, we focus on one of these open problems, that is, the problem to find the minimum moves of solving rolling cube puzzles when the initial state and the final state of the die are given. In order to get some properties of this problem, we refine this problem and introduce new motions of this problem. We only discuss the situation that the initial state of the die and the final state are given and the die can be rolled freely on a board without inaccessible cells. Then, we divide this problem into two cases. One is that the die can roll along a path, whose length is equal to Manhattan distance without any detours. The other is that the die can roll in all four directions on an unlimited board. By the experiment on the first case, we found the area, that is we can roll a die to find all the possible states on a cell only along a path of length equal to Manhattan distance. We also found the optimal number of the minimum moves to find all the possible states on a cell in the other area.

According to these results of the experiments, we give an algorithm for finding a general solution for the problem and analysis the complexity of this algorithm. In the algorithm, we divide the problem into four subproblems. The first one asks whether the path from the initial state to the final state exists or not. The second one asks if a die can reach to all the final states from the initial state along a path of length equal to the Manhattan distance or not. Thirdly, the algorithm determines if a die can reach the final state with a detour of length not greater than 2 addition to its Manhattan distance. The complexity of this part will take $O(4^{d(T)})$ time, where $d(T)$ is the depth of the BFS tree T, because we need to explore the four moving directions

of the die. However, we know the upper bound of the maximum steps of a detour to find all the possible states on a cell. That means we only need to explore the area along the paths of the length of the Manhattan distance with the maximum detour. Furthermore, if the algorithm records all the states of the die after it has rolled and the cell of these states, the running time is drastically reduced in the manner of dynamic programming. Meanwhile, we verify that the new state does not duplicate in the table. After these optimizations, the complexity of this part will be reduced to $O(D_m^2 \times d(T))$ time, where D_m is the Manhattan distance between the initial cell and the goal cell. Finally, we only have the situation left, that is the minimum moves are equal to Manhattan distance plus 4. For this last subproblem, we transform the goal to find the corresponding state on the cell next to the initial cell. Follow these steps, we can get a general solution to find a shortest path between the starting point and the ending point when the initial state of the die and the final state are given and the die can be rolled freely on a board without inaccessible cells.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Rolling cube puzzles are a puzzle game that rolls the die which is placed on a board to complete a goal followed certain rules. For example, Figure 1.1 is a simple rolling cube puzzle posed by Erik D. Demaine and Joseph O' Rourke at CCCG 2005. In this puzzle, the objective is to roll the die which is at the upper left corner of the board over all labeled cells of the board exactly once. When the die rolls over a labeled cell, the number on the top face of the die must be the same as the number of the cell which it lies on. In addition to labeled cells, there are some cells called free cells which are white without any marks. The die can enter and exit these cells freely. That means the die may roll on free cells multiple times, and there is no limit to the number of the top face.

Also, according to some other rules, we can get the other kinds of rolling cube puzzles as in Figure 1.2.

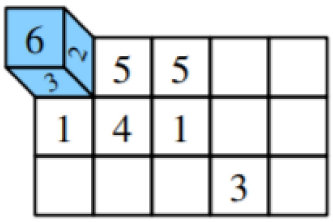

Figure 1.1: Rolling cube puzzles posed by Joseph O' Rourke at CCCG 2005[3]

Rolling cube puzzles have been investigated with a long history. They were popularized by Martin Gardner[1]. Over the past 20 years, considerable attention has been paid to rolling cube puzzles. In 2007, Kevin Buchin et al.

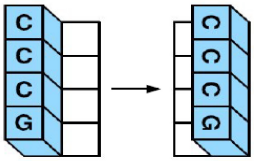

Figure 1.2: Rolling cube puzzles posed by Martin Demaine at CCCG 2005[3]

researched the rolling cube puzzles. In [1], they proved that solving rolling cube puzzles is NP-complete in the circumstance that rolling cube puzzles defined by a six-sided die on a square grid board. In 2011, Kevin Buchin and Maike Buchin extended their proof for rolling cube puzzles to show the complexity of rolling block maze puzzles[4]. In the same year, rolling cube puzzles were generalized by Akihiro Uejima and Takahiro Okada. They discussed the computational complexity of an expansion problem on rolling cube puzzles and substantiated the generalized problem is NP-complete when using different kinds of board such as a triangular grid board or when using some other types of dice such as eight-sided and twenty-sided die[2].

However, there are several related puzzles and open problems posed in the paper by Kevin Buchin et al. As can be seen, it still requires further research how to resolve these problems.

1.2 Research Purposes

In this research, we focus on one of these open problems, that is, the problem to find the minimum moves of solving rolling cube puzzles when the initial state and the final state of the die are given. In this problem, we know the coordinate of the initial cell is (x, y) and the coordinate of the final cell is (x', y'). We also know the initial state of the die is k and the final state is k'. These states are dependent on the location relations of different faces of a die when it lies on the initial cell or the final cell. For these given conditions, we are devoted to find a shortest path to reach the final state k' from the initial state k by only one die when the die can be rolled freely on a board without inaccessible cells. Figure 1.3 is an example of rolling cube puzzles when the initial state and the final state of the die are given.

Specifically, we discuss the details of the moves and procedures for solving rolling cube puzzles and make the processes explicit when the initial state and final state of the die are given. Based on this, we give an algorithm for finding a general solution for the problem and analysis the complexity of this algorithm.

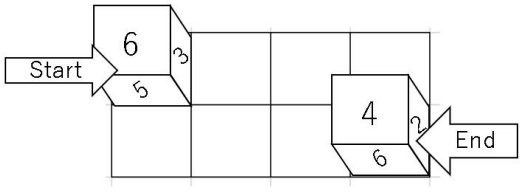

Figure 1.3: An example of rolling cube puzzles when the initial state and the final state of the die are given

Rolling cube puzzles are a special puzzle with extraordinary properties which are critically different from other conventional puzzles. Rolling cube puzzles require a three-dimensional way of thinking, because of not only the complexity of the board but also using a die. Elucidation of these properties provides a new and useful perspective on the characterization of complexity classes. There is no doubt that solutions to rolling cube puzzles will unlock the critical bottleneck of solving the other problem with the same nature of rolling cube puzzles.

1.3 Composition

In Chapter 2, we use a simple puzzle (Figure 1.1) to show the basic properties and characteristics of rolling cube puzzles. For instance, the state of a die and the parity property of rolling cube puzzles will be introduced. In addition, we explain a part of the proof about the complexity of rolling block maze puzzles with only one block[4].

In Chapter 3, we analyze rolling cube puzzles without inaccessible cells when the initial state of the die and the final state are given.

In Chapter 4, we show an algorithm for finding a shortest path to solve our problem and analysis the complexity of this algorithm.

Chapter 2

Preliminaries

In this chapter, we introduce the basic definition of the rolling cube puzzles in Section 2.1. In Section 2.2, we illustrate some basic properties which are central to solving the problem investigated in Chapter 3.

2.1 Definition of Rolling Cube Puzzles

2.1.1 General Rolling Cube Puzzles

In general, a rolling cube puzzle is composed by one or more dice, a board, a goal, and some rules. In this research, we restrict ourselves to puzzles with one die. There are many kinds of dice, here we use the standard die, which is more common. A standard die is accounted as a cube with six faces labeled one to six and the sum of the two numbers labeled on opposite faces is always seven (Figure 2.1). and Figure 2.2 is a geometry net of this standard die.

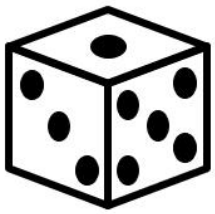

Figure 2.1: A standard die

There are two kinds of the standard dice. According to how the number on the labeled faces are arranged, they can be classified as right-handed or left-handed dice. In particular, in the case, that is, the numbers from 1 to 3 are arranged counterclockwise when we gaze the vertex in the center of

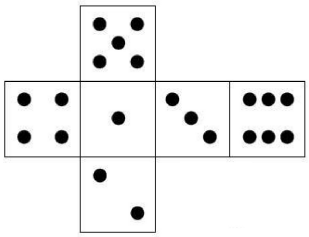

Figure 2.2: Net of a standard die[6]

the faces labeled by 1, 2, and 3, the die is called right-handed die. On the contrary, when they are lined clockwise, the die is left-handed die. Figure 2.3 is an illustration.

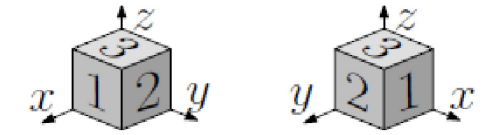

Figure 2.3: The two standard dice[1]

The board is a grid with three types of cells: labeled cells, free cells and blocked cells. As we introduced at the beginning of this paper, labeled cells must be accessed exactly once when the number on the top face of the die should be the same as the label of the cell it lies on. Free cells can be visited with no limit. That means a die can visit free cells with any number on the top face and any times. Then, definition of blocked cells is simple. No matter what happens, the die cannot visit blocked cells.

Given a board, we need to roll the die on the board to complete some missions. See Figure 1.1. It is a puzzle that the die must visit all the labeled cells. One solution of Figure 1.1 is ESENEESSWWWWN, where E,S,W,N mean East, South, West, and North, respectively (Figure 2.4). In other cases, we may be given a initial state and a final state of the die (Figure 1.3). We consider the case when the initial state and the final state of the die are determined.

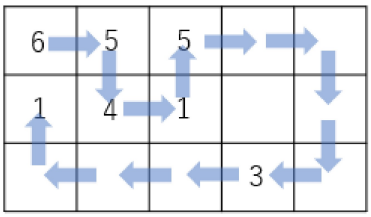

Figure 2.4: One solution of Figure 1.1[3]

2.1.2 State of a Die

We defined the state of a die by a label UBL, where U is the label on the top of the die, B the label on the back face and L the label on the left face. For example, in Figure 2.5, the state of this die is 356 (2 is the label on the front face).

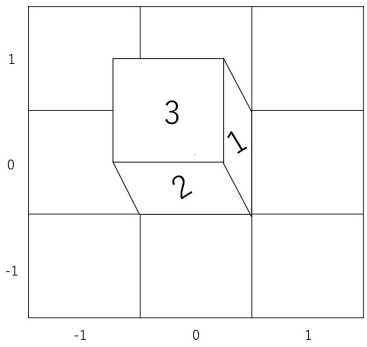

Figure 2.5: State of a die

2.1.3 Four Moving Directions of a Die, East, South, West, North

At last, we define that the four moving directions of a die is East, South, West and North in the coordinate system, for the cell(x, y) the die lied on is given. After an East move, the cell the die lied on will change from (x, y) to $(x+1, y)$. After a South move, the cell the die lied on will change from (x, y) to $(x, y-1)$. After a West move, the cell the die lied on will change from (x, y) to $(x-1, y)$. After a North move, the cell the die lied on will change from (x, y) to $(x, y+1)$.

The state of a die also change after a move which is dependent by the kind of the die.

2.2 Parity Property

Generally, all vertices of the state graph can be divided into two sets of the same size with no edges between the two vertices in the same set. By symmetry, we only need to discuss two of the four orientations if the initial state is given.

In other word, assume we color the corners of the die alternatively with white and black (Figure 2.6(a)), and roll this die over all cells of a board. All nodes of the board will be dyed in these two colors like it shows in Figure 2.6(b). Only two orientations are allowed in a labeled cell because it is not possible to turn the die 90 degrees in the horizontal direction.

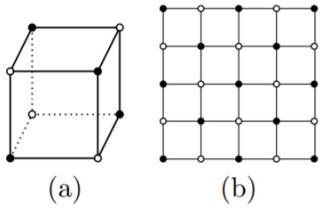

Figure 2.6: A die and a board with 2-colored corners[1]

Now, we have a piece of checker board pattern of alternating black and white squares like Figure 2.7. Then, we define that K is a set of the all states of a die at a white cell in Figure 2.7. K' is a set of the all states of a die at a black cell.

For example, if a die starts with (362) at board A. Then K is {(362), (315), (124), (153), (241), (236), (465), (412), (546), (531), (654), (623)} and K' is {(132), (145), (213), (264), (321), (356), (451), (426), (514), (563), (635), (642)}. The cardinality of K and K' are 12.

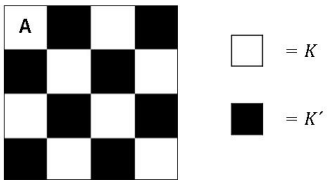

Figure 2.7: Checker board pattern of alternating black and white squares

2.3 Complexity of the Puzzle with only one Block

Rolling block mazes puzzle with only one block can be solved using linear time and space[4]. Kevin Buchin and Maike Buchin consider that if one of the position results from the previous position of it, the size of the state graph is determined by the size of the board, independent of the size of the block. Then, the puzzle can be solved by searching the state graph for a path from the starting to the ending position. Thus, if using breadth first search to solve this, it can be done in time linear in the number of edges.

Chapter 3

Analysis of the Rolling Cube Puzzle

In this chapter, we reveal some properties drawn quickly of the case. In Section 3.1, we define some notions and notations that we will use later. In Section 3.2, we discuss a simple case that we know a starting position and an ending position of the die. Further, we analyze the case rolling a die on a board without any limits in Section 3.3.

3.1 Some Definitions of Rolling Cube Puzzle

3.1.1 The Definition of $D_m((x, y), (x', y'))$

First of all, we define that $D_m((x, y), (x', y'))$ is $|x - x'| + |y–y'|$, when the coordinate of the start point is (x, y), and coordinate of the end point is (x', y'). D_m is "manhattan distance". For example, $D_m((0, 0), (3, 2)) = 3 + 2 = 5$.

3.1.2 The Definition of Path P and $L_p(P)$

We define that path P is the route from (x_0, y_0) to (x_n, y_n) in which a die is rolling. Path P is a sequence of all the cells which the die has been visited. Path $P = ((x_0, y_0), (x_1, y_1), \ldots , (x_n, y_n))$, where $D_m((x_i, y_i), (x_{i+1}, y_{i+1})) = 1$. The length $L_p(P)$ of a path P is $|n - 1|$, where n is the number of points in the path P.

If we roll a die from A to B in Figure 3.1, the path α (A, B) is $((0, 0), (1, 0))$. In the case, $L_p(\alpha)$ is 1. While we roll a die from A to C via B, the path β (A, C) will be $((0, 0), (1, 0), (2, 0))$, $L_p(\beta)$ is 2.

In Figure 3.2, it shows us two ways to roll a die from A to B. If we roll a die along the blue arrow, the path a (A, B) is $((0,0),(1,0))$, and $L_p(a)$ is 1. If we roll a die along the yellow arrow, the path b (A, B) is $((0,0), (0,1), (0,2), (1,2), (1,1), (1,0))$, and $L_p(b)$ is 5.

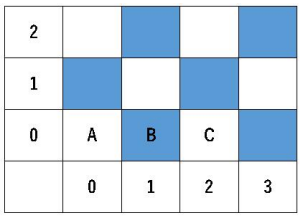

Figure 3.1: A broad

3.1.3 The Definition of $L_d(P)$

We define that $L_d(P)$ is $|L_p(P) - D_m((x_0, y_0), (x_n, y_n))|$. L_d means the length of a detour. In Figure 3.2, D_m from A to B is 1. If we roll a die along the yellow arrow, $L_p > D_m$. That means we take a detour from A to B. For example, in this case, L_d of rolling along the yellow arrow is 4.

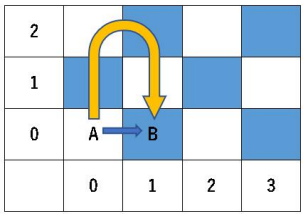

Figure 3.2: Two ways from A to B

3.1.4 The Definition of $k(P, UBL)$ and $K_{final}(((x, y), UBL), (x', y'))$

Now, we define that $k(P, UBL)$ is the current state after rolling a die along the path P from the initial state UBL and $K_{final}((x, y), UBL, (x', y'))$ is all the current states of one cell after rolling. That means $k(P, UBL) \in K_{final}((x, y), UBL, (x', y'))$. $K_{final}((x, y), UBL, (x', y')) \subseteq K$ *or* K'.

3.1.5 Parity of L_d

Theorem 1 L_d is even.

Proof. According to parity in Section 2.2, when $L_p = 0$, k is the initial state of the die, $k \in K_{final} \subseteq K$. Meanwhile, every time L_p add one, which set of all the current states of one cell after rolling K_{final} is included in, will transform from K to K' or from K' to K.

Assume L_d is odd, then $L_p = D_m + odd$. So, when D_m is odd, L_p is even, $K_{final} \subseteq K$. When D_m is even, L_p is odd, $K_{final} \subseteq K'$.

But, when $L_p = D_m$ is odd, $K_{final} \subseteq K'$. When $L_p = D_m$ is even, $K_{final} \subseteq K$. Whether D_m is odd or even, the results are inconsistent. That means the assumption, L_d is odd, is false. So, L_d is even. □

3.1.6 The Definition of $L_d^u(((x, y), UBL), ((x', y'), U'B'L'))$

Before discussing the results of the experiment, we import a new notion that $L_d^u(((x, y), UBL), ((x', y'), U'B'L'))$ is the upper limit of the minimum of detours for one state of a die on a cell.

Now, there is a problem to roll a die from a initial state (0, 0, 362) to an final state (0, 0, 241). Figure 3.3 shows us two ways to clear this problem. In Figure 3.3, the path of rolling along the blue arrow are ((0,0), (1,0), (1,1), (0,1), (0,0)). The path of rolling along the yellow arrow are ((0,0), (1,0), (2,0), (3,0), (2,0), (1,0), (1,1), (0,1), (0,0)). L_d of rolling along the blue arrow is 4 and L_d of rolling along the yellow arrow is 8. By experiment, rolling along the blue arrow is the minimum detour we need for rolling a die from (0, 0, 362) to (0, 0, 241). That means if we roll a die along the yellow arrow, we take a meaningless detour. So, we can say L_d^u of rolling a die from (0, 0, 362) to (0, 0, 241) is 4.

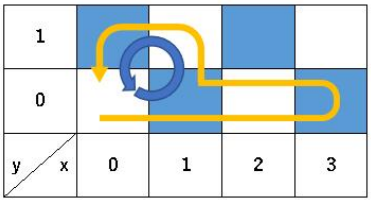

Figure 3.3: Two paths of rolling a die from (0, 0, 362) to (0, 0, 241)

3.2 Rolling a Die along a Path of Length $L_p = D_m$

3.2.1 Preparation

As mentioned above, our objective is to accomplish the case when the initial state and the final state of the die are determined. It is easy if there is no limit of steps to accomplish the case. Thus we devote to finding the minimum steps to finish this case. Now we prepare a board without blocked cells and a left-handed die. We use bearing to named movement of a die like N(north) means moving upwards (Figure 3.4).

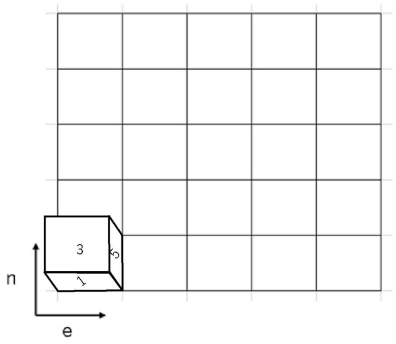

Figure 3.4: Board without blocked cells

3.2.2 Some Simple Properties

Due to our goal, we limit the directions of movement in only North and East. That means we roll a die in such a case $L_p = D_m$. So, the results we found must be the shortest paths. See Figure 3.5 for an example.

In Figure 3.5, all kinds of cell (4,4), cell (3,6) and cell (6,3) is 12 without duplicate results. As we know in Section 2.2, 12 is the maximumimum possible outcomes on one cell. So, if a cell $\alpha(x_1, y_1)$ is farther than cell (4,4), cell (3,6) and cell (6,3), we can find the maximumimum possible outcomes K or K' on the α. x_1 and y_1 subject to the following conditions,

$$\begin{cases} |x_1| \in [4, +\infty) \wedge |y_1| \in [4, +\infty) \\ |x_1| \in [3, +\infty) \wedge |y_1| \in [6, +\infty) \\ |x_1| \in [6, +\infty) \wedge |y_1| \in [3, +\infty) \end{cases}$$

As we know in the case of Figure 1.3, it is not always can find a result in the condition whose $L_p = D_m$. Go a step further, if we cannot find all the

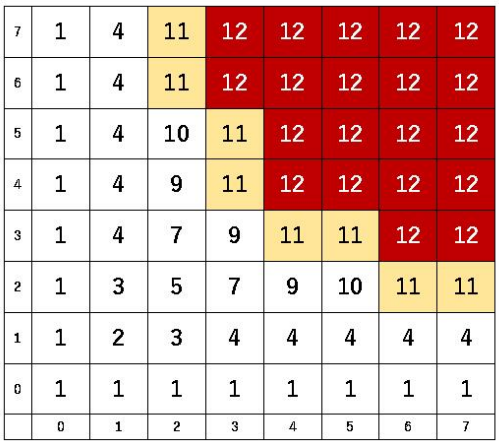

Figure 3.5: Number of results without duplicated of only two directions

possible outcomes on a cell. That means we have to make a detour to find the remained results on this cell.

For example, in Figure 3.5, the red cells are the cell α which have discussed above. That means we can find the maximumimum possible outcomes K or K' in the red cells by D_m. The yellow cells are a part of the cells which we cannot find the maximumimum possible outcomes by D_m. If we roll a die from a red cell, which is next to a yellow cell, to the yellow cell. Then, we can get all the possible outcomes in this yellow cell by taking a detour.

3.2.3 2 Parts of a Path to a Red Cell

Theorem 2 When (x, y) meet each following condition,

$$\begin{cases} |x| \in [4, +\infty) \wedge |y| \in [4, +\infty) \\ |x| \in [3, +\infty) \wedge |y| \in [6, +\infty) \\ |x| \in [6, +\infty) \wedge |y| \in [3, +\infty) \end{cases}$$

we can find the maximumimum possible outcomes K or K' on (x, y), where $L_d = 0$.

Proof. According to mathematical induction of the results of only two directions (Figure 3.5), we can say following. For any given a shortest path from $(0, 0)$ to (x, y), we can find a way to transform the path into two parts. The first part is a path A from $(0, 0)$ to $(\pm 4, \pm 4)$, $(\pm 3, \pm 6)$ or $(\pm 6, \pm 3)$. The second part is a path B from the final cell of the path A to (x, y). If we connect A and B, we also can reach the final state of the die from the initial state and keep L_p the same.

Thus, Theorem 2 can be proved. □

3.3 Minimum Number of Rolling a Die on an Unlimited Board

3.3.1 Preparation

In this step, we hope to know the range of detours. As mentioned above, we only decide the state on the start position in the case we do in this section. There is no any more limit of this case. Now we prepare a board without blocked cells and a left-handed die. Then we roll a die from (0,0) under the state of the die is 362.

3.3.2 The Results on an Unlimited Board

Next, we talk about the results of our experiment. In this experiment, we have explored all the rolling paths in a area. In this area, the coordinate of the all cells must meet the conditions that $x \in [-9, 9] \wedge y \in [-9, 9]$. By this experiment, we can know all L_d^u of getting the maximumimum possible outcomes K or K' on a cell . Figure 3.6 is a path of our results. In Figure 3.6 we know L_d^u of blue cell is 6. L_d^u of orange cell is 4. L_d^u of green cell is 2. L_d^u of yellow cell is 0.

y/x	0	1	2	3	4	5	6	7	8	9
9	4	2	2	0	0	0	0	0	0	0
8	4	2	2	0	0	0	0	0	0	0
7	4	2	2	0	0	0	0	0	0	0
6	4	2	2	0	0	0	0	0	0	0
5	4	2	2	2	0	0	0	0	0	0
4	4	2	2	2	0	0	0	0	0	0
3	4	4	2	2	2	2	0	0	0	0
2	4	4	4	2	2	2	2	2	2	2
1	4	4	4	4	2	2	2	2	2	2
0	6	4	4	4	4	4	4	4	4	4

Figure 3.6: L_d^u to get the maximum possible outcomes K or K' on a cell

3.3.3 The Cell whose L_d^u is 6

According to the result above, cell(0,0) is one and the only one cell whose L_d^u is 6. If we roll a die starting from (0,0), we want to finish this rolling at (0,0). We can take 3 kinds of rolling in Figure 3.7. (a) is rolling a die along a clockwise path of length 4. For example, a path which is ((0,0), (1,0), (1,-1), (0,-1), (0,0)) is a clockwise path. By experiment, we can know that if we roll a die through (a) kind of rolling, we can find 4 kinds of outcomes {(236), (153), (546), (654)} on (0,0). Through (b) kind of rolling which is rolling a die along an anti-clockwise path of length 4, we can get the other 4 kinds of outcomes {(241), (124), (531), (623)} on (0,0). So, if we want to get the last 3 kinds outcomes {(315), (465), (412)} on (0,0), we should roll a die through (c) kind of rolling whose L_d is 6. That means we can find the maximumimum possible outcomes K on (0,0) through rolling a die along a path P, where $L_p(P) \leq 6$.

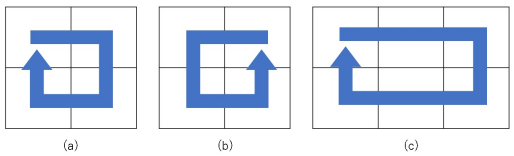

Figure 3.7: 3 kinds of rolling from (0,0) to (0,0)

3.3.4 A General Transformation

Theorem 3 For any given (x, y), (x', y'), initial state k and final state k', we can find a way to transform from the initial state k to the final state k' by rolling along a path P, where $L_p(P) \leq D_m((x, y), (x', y')) + 6$.

Proof. Because we know that we can find the maximumimum possible outcomes K on the initial cell (x, y) through rolling a die along a path P, where $L_p(P) \leq 6$. We can find a way to transform from the initial state k to the corresponding state k_c we need on the cell(x, y). Then, we transform from the corresponding state k_c to the final state k' on the cell(x', y') by rolling along a path P', where $L_p(P') = D_m((x, y), (x', y'))$. Thus, Theorem 3 can be proved. □

3.3.5 Efficient algorithm for reachability problem

Theorem 4 For any given (x, y), (x', y'), initial state $k \in K$, we have the following.

Case 1 $D_m((x,y),(x',y'))$ is odd.

- For any path P between (x,y), (x',y'), the final state k' by P belongs to K'.
- For any $k' \in K'$, we have some path P such that the final state by P is k'.

Case 2 $D_m((x,y),(x',y'))$ is even.

- For any path P between (x,y), (x',y'), the final state k' by P belongs to K.
- For any $k' \in K$, we have some path P such that the final state by P is k'.

Proof. According to parity in Section 2.2 and Theorem 3, Theorem 4 can be proved by mathematical induction. □

3.3.6 Cells whose L_d^u is 4

According to the results in Figure 3.6, L_d^u of $\beta(x_2,y_2)$ is 4, where x_2 and y_2 subject to the following conditions,

$$\begin{cases} |x_2| \in (0,+\infty) \wedge y_2 = 0 \\ |y_2| \in (0,+\infty) \wedge x_2 = 0 \\ |x_2| + |y_2| \in (0,4] \end{cases}$$

.

As we know in the last subsection, (0,0) is the only cell whose L_d^u is 6. Now, we discuss the upper limit of the area of the orange cells. In Figure 3.8, the cells on the line a satisfy to the conditions that $|x_2| + |y_2| = 5$. The least that can be concluded from the result above is that we can find the maximumimum possible outcomes K' at cell (4,1), (3,2), (2,3), (1,4) when $L_p = 7$. We cannot find the maximumimum possible outcomes K' at cell (5,0), (0,5) when $L_p = 7$. Due to parity property, we will never find the maximumimum possible outcomes K' at cell (5,0), (0,5) when $L_p = 8$. According to the results of our experiment, we can find the maximumimum possible outcomes K' at cell (5,0), (0,5) when $L_p = 9$. In the same way, we can find the maximumimum possible outcomes K at cell (5,1), (4,2), (3,3), (2,4), (1,5) when $L_p = 8$. We can find the maximumimum possible outcomes K at cell (6,0), (0,6) when $L_p = 10$. In this fashion, we go on discussing the cell whose $D_m = 7,8,9$. We always get the same conclusion that $L_p(x = 0 \vee y = 0 \wedge D_m = 7,8,9) > L_p(x \neq 0 \wedge y \neq 0 \wedge D_m = 7,8,9)$.

It is unable to roll a die to the cell(x, y), which subjects to the conditions that $|x| \in (0, +\infty) \wedge y = 0$ or the conditions that $|y| \in (0, +\infty) \wedge x = 0$, by a shorter path whose $L_d < 4$. So, we can say the upper limit of the area of the orange cells is $|x| + |y| = 4 \vee |x| \geq 5(\text{when } y = 0) \vee |y| \geq 5(\text{when } x = 0)$.

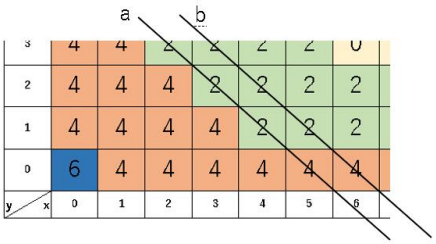

Figure 3.8: A part of Figure 3.6

3.3.7 A Transformation when $L_d = 4$

Theorem 5 For any given (x, y), (x', y'), initial state k, final state k' and a path P from (x, y) to (x', y') satisfying the condition $L_d(P) = 4$, there exists a way that we can transform the goal to find the corresponding state k_c on the cell next to the initial cell(x, y) and keep L_p the same.

Proof. By experiment, we know that we can find the all possible outcomes K' on the cell (x_n, y) or the cell (x, y_n) though rolling a die along a path P from the initial cell(x, y) to (x_n, y) or (x, y_n), where $|x_n - x| = |y_n - y| = 1$ and $L_p(P) = 5$.

Therefore, Theorem 5 can be proved by mathematical induction. □

3.3.8 Cells whose L_d^u is 2

According to the results in Figure 3.6, L_d^u of $\gamma(x_3, y_3)$ is 2, where x_3 and y_3 subject to the following conditions,

$$\begin{cases} |x_3| \in [1, 2] \wedge |y_3| \in (4, +\infty) \\ |y_3| \in [1, 2] \wedge |x_3| \in (4, +\infty) \\ |x_3| + |y_3| \in [5, 8] \wedge (x_3, y_3) \neq (4, 4) \wedge |x_3| \in (0, +\infty) \wedge |y_3| \in (0, +\infty) \end{cases}$$

.

In the same way, we can discuss the upper limit of the area of the green cells.

In the same fashion we used in the last subsection, we can know, when $L_p = 8$, we can find the maximumimum possible outcomes K at cell (4,4).

We cannot find the maximumimum possible outcomes K at cell (7,1), (6,2), (5,3), (3,5), (2,6), (1,7) when $L_p = 8$ we can find K at these cells when $L_p = 10$. Also, when $L_p = 9$, we can find the maximumimum possible outcomes K' at cell (6,3), (3,6). We can find K' at cell (8,1), (7,2), (2,7), (1,8) when $L_p = 11$.

It is unable to roll a die to the cell$(\pm 4, \pm 3)$, $(\pm 5, \pm 3)$, $(\pm 3, \pm 4)$, $(\pm 3, \pm 5)$, (x, y), which subjects to the conditions that $|x| \in [6, +\infty) \wedge y = 2$ or the conditions that $|y| \in [6, +\infty) \wedge x = 2$, by a shorter path whose $L_d < 2$. Thus, we can say the upper limit of the area of the green cells is $|x| \geq 6$(when $|y| = 2) \vee |y| \geq 6$(when $|x| = 2) \vee (x, y) \in \{(\pm 4, \pm 3), (\pm 5, \pm 3), (\pm 3, \pm 4), (\pm 3, \pm 5)\}$.

Chapter 4

Implementation and Analysis of the Algorithm

4.1 Efficient algorithm for reachability

Theorem 6 For any given (x, y), (x', y'), initial state k and final state k', we can find a shortest path from k to k' in $O((D_m)^2 \times (D_m + L_d))$ time.

We will prove Theorem Theorem 6 in the next subsections.

4.2 Function IFK()

According to Theorem 4, we can define a function IFK() to decide whether the goal can be achieved, when we input the state of starting point and the state of ending point. The pseudo-code of IFK() is Algorithm 1.

As we know in Section 2.2, we can get the set of the maximumimum possible outcomes K or K' in all cells, when we know the state of starting point. We can calculate D_m from starting point to ending point, and decide whether the possible outcomes at the ending point is belonged K or K'. Now, we know the state of starting point is (0, 0, 362), and three states of ending point($U_k B_k L_k$). Because of parity property, if D_m is odd, the possible outcomes($U_p B_p L_p$) at the ending point will belong to the set K'. If D_m is even, ($U_p B_p L_p$) will belong to the set K. Thus, when D_m is an odd, if ($U_k B_k L_k$) is belong to K'. Then, we can say this case can be cleared. Otherwise, this case can't be cleared. Similarly, when D_m is an even, if ($U_k B_k L_k$) is belong to K. Then, we can say the input is feasible. Otherwise, this input is not feasible.

Algorithm 1 Function IFK($(x_1, y_1, U_1B_1L_1), (x_2, y_2, U_2B_2L_2)$)

input: the state of starting point, the state of ending point
output: **true** or **false**

```
1: if D_m is odd then
2:    if (U_2B_2L_2) ∈ K' then
3:       return true
4:    else
5:       return false
6:    end if
7: else if D_m is even then
8:    if (U_2B_2L_2) ∈ K then
9:       return true
10:   else
11:      return false
12:   end if
13: end if
```

Algorithm 2 Function IFYC(x_2, y_2)

input: the coordinate of ending point
output: **true** or **false**

```
1: if |x_2| ∈ [4, +∞) ∧ |y_2| ∈ [4, +∞) then
2:    return true
3: else if |x_2| ∈ [3, +∞) ∧ |y_2| ∈ [6, +∞) then
4:    return true
5: else if |x_2| ∈ [6, +∞) ∧ |y_2| ∈ [3, +∞) then
6:    return true
7: else
8:    return false
9: end if
```

4.3 Function IFYC()

IFYC() decides whether the cell of our goal is a cell of the yellow cells we knew in Chapter 3 or not, when we input the coordinate of ending point. IFYC() also transforms the goal to find the corresponding state on the cell $(4, 4)$, $(3, 6)$, $(6, 3)$. The pseudo-code of IFYC() is Algorithm 2.

Algorithm 3 Function East($x_1, y_1, U_1B_1L_1, path_1$)

input: the state of starting point, the path to initial state
output: the state of ending point, the path to the final state

1: $x_2 \leftarrow x_1 + 1$
2: $y_2 \leftarrow y_1$
3: $U_2 \leftarrow L_1$
4: $B_2 \leftarrow B_1$
5: $L_2 \leftarrow 7 - U_1$
6: $path_2 \leftarrow path_1.append(x_2, y_2)$
7: **return** $(x_2, y_2, U_2B_2L_2, path_2)$

4.4 4 Functions East(), South(), West(), North()

East(), South(), West(), North() are four functions used to indicate the change of a die in four directions, when $L_p = 1$, if we input the state of starting point. For example, the pseudo-code of East() is Algorithm 3.

4.5 Function EXP()

EXP() is a function used to decide whether the goal can be achieved by $D_m + L_d$, and give a path to achieve this goal. According to BFS[5], an example of the pseudo-code of EXP() is Algorithm 4.

4.6 Complexity of Function EXP()

The complexity of this part will take $O(4^{D_m+L_d})$ time in a naive implementation, because we need to explore the four moving directions of the die. However, we know the upper bound of the maximumimum moves of a detour to find all the possible states on a cell. That means we only need to explore the area along the paths of the length of the Manhattan distance with the maximumimum detour. Furthermore, if the algorithm records all the states of the die after it has rolled and the cell of these states, the running time is drastically reduced in the manner of dynamic programming[5]. Meanwhile, we verify that the new state does not duplicate in the table.

Figure 4.1 is an example of the application of dynamic programming. We can blacken the cell when a state on the cell is appeared. When we reach

Algorithm 4 Function EXP($(x_1, y_1, U_1B_1L_1), (x_2, y_2, U_2B_2L_2), L_d$)

input: the state of starting point, the state of ending point, L_d
output: a path to the state of ending point, or **false**

$p \leftarrow (x_1, y_1)$
$S \leftarrow (U_1B_1L_1)$
$p_f \leftarrow (x_2, y_2)$
$S_f \leftarrow (U_2B_2L_2)$
$Q.push(p, S, path, D_m + L_d)$
while Q is not an empty list **do**
  $p, S, path, n \leftarrow Q.pop()$
  if $n \leq 0$ **then**
    return false,
  end if
  $p_1, S_1, path_1 \leftarrow East(p, S, path)$
  $p_2, S_2, path_2 \leftarrow South(p, S, path)$
  $p_3, S_3, path_3 \leftarrow West(p, S, path)$
  $p_4, S_4, path_4 \leftarrow North(p, S, path)$
  for $i = 1; i \leq 4; i++$ **do**
    if $(p_i, S_i) == (p_f, S_f)$ **then**
      return $path_i$
    end if
  end for
  $Q.push(p_1, S_1, path_1, n - 1)$
  $Q.push(p_2, S_2, path_2, n - 1)$
  $Q.push(p_3, S_3, path_3, n - 1)$
  $Q.push(p_4, S_4, path_4, n - 1)$
end while

this cell again, we verify if the new state duplicates the states are appeared in the graph. The number of different vertices in the state graph is $D_m + L_d$.

After these optimizations, the time complexity of this part will be reduced to $O((D_m)^2 \times (D_m + L_d))$ time.

4.7 Function TRTSC()

According to Theorem 2 and Theorem 5, we give a function TRTSC() to transform the goal to find the corresponding state on some specific cells. To

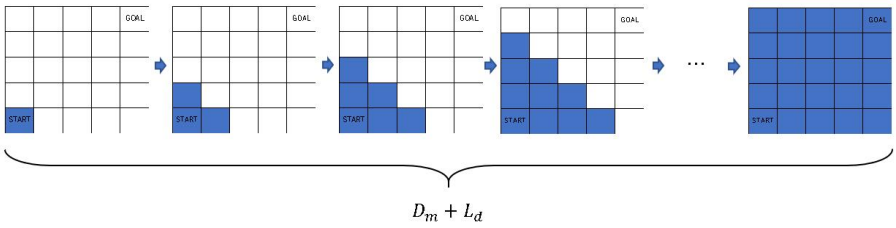

Figure 4.1: An example of the application of dynamic programming

make TRTSC() easier to understand, we assume the initial cell (x, y) is (0,0) and the goal cell (x_g, y_g) with $x_g \geq 0$ and $y_g \geq 0$ by symmetry, and we let the final state be $(U_g B_g L_g)$. We also define a function REVERSE(path A) to reverse the path A and two functions TRS() and TRW() to find the corresponding states$((x_c, y_c), (U_c B_c L_c))$ from the final states$((x_g, y_g), (U_g B_g L_g))$.

4.8 Function RCP()

RCP() is the main function to solve our problem. Without loss of generality, we assume the initial cell (x, y) is (0,0) and the goal cell $x_2, y_2)$ with $x_2 \geq 0$ and $y_2 \geq 0$. An example of the pseudo-code of RCP() is Algorithm 5.

4.9 Proof of Theorem 6

Proof. Based on the functions above, the complexity of IFK() is $O(1)$ time, the complexity of IFYC() is $O(1)$ time, the complexity of TRTSC() is $O(D_m)$ time and the complexity of EXP() is $O((D_m)^2 \times (D_m + L_d))$ time. Then, we can say that the complexity of RCP() is $O((D_m)^2 \times (D_m + L_d))$ time.

So, Theorem 6 can be proven. □

Algorithm 5 Function RCP$((x_1, y_1, U_1B_1L_1), (x_2, y_2, U_2B_2L_2))$

input: the state of starting point, the state of ending point
output: a path to the state of ending point or **false**

1: **if** IFK$((x_1, y_1), (x_2, y_2))$ **then**
2: **if** IFYC(x_2, y_2) **then**
3: TRTSC$((x_2, y_2), (U_2B_2L_2))$
4: EXP$((x_1, y_1, U_1B_1L_1), (x_c, y_c, U_cB_cL_c), 0)$
5: **return** $path.append(path_c)$
6: **else**
7: EXP$((x_1, y_1, U_1B_1L_1), (x_2, y_2, U_2B_2L_2), 2)$
8: **if** EXP$((x_1, y_1, U_1B_1L_1), (x_2, y_2, U_2B_2L_2), 2) \neq$ **false then**
9: **return** $path$
10: **else**
11: TRTSC$((x_2, y_2), (U_2B_2L_2))$
12: EXP$((x_1, y_1, U_1B_1L_1), (x_c, y_c, U_cB_cL_c), 4)$
13: **return** $path.append(path_c)$
14: **end if**
15: **end if**
16: **else**
17: **return false**
18: **end if**

Chapter 5

Conclusion and Future Work

5.1 Conclusion

For the problem, that is, the problem to find the minimum moves of solving rolling cube puzzles when the initial state and the final state of the die are given, we showed some qualities about this problem.

At last, we gave an efficient algorithm RCP() for finding a general solution for the problem and analysis the complexity. After considering efficient implementation, the time complexity of RCP() is reduced to $O((D_m)^2 \times (D_m + L_d))$ time.

5.2 Future Work

There are some future work. For example, the following two problems are possible future work.

- Rolling cube puzzles on a board with blocked cells when the initial state and the final state of the die are given.
- Rolling cube puzzles used 2 dice or more when the initial states and the final states of the dice are given.

Bibliography

[1] Kevin Buchin, Maike Buchin, Martin L. Demaine, Erik D. Demaine, Dania El-Khechen, Sándor P Fekete, Christian Knauer, André Schulz and Perouz Taslakian: On Rolling Cube Puzzles, *Canadian Conference on Computational Geometry*, pp.141–144 (Aug. 2007).

[2] 上嶋章宏, 岡田貴裕 : 8 面 , 20 面ダイスを用いた Rolling Dice Puzzle の NP 完全性, 電子情報通信学会論文誌 *A*, 94(8), pp.621–628 (2011).

[3] Erik D. Demaine and Joseph O ' Rourke: Open problems from CCCG 2005, *Proc. 18th Canadian Conference on Computational Geometry (CCCG ' 06)*, pp.75–80 (2006).

[4] Kevin Buchin, Maike Buchin: Rolling Block Mazes are PSPACE-complete, *Journal of Information Processing*, Vol.20, No.3, pp.719–722 (July 2012).

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein: *Introduction to Algorithms*, pp.359–414&594–603 (1990), MIT Press.

[6] José Pereda: Customizing Javafx 3d Box or Rotating Stackpane, https://stackoverflow.com/questions/31274295/customizing-javafx-3d-box-or-rotating-stackpane, 2020-01-23.