ルービックキューブの解法の比較

5年Ｂ組　中谷　太紀
指導教員　川口　慎二

１．要約

昨年の研究ではルービックキューブの特定の状態について、複数の解法の比較を行った。今年度は、あらゆる状態のルービックキューブについて、コンピュータを用いることで、複数の解法の効率を検証した。

キーワード　ルービックキューブ、Python

２．研究の背景

ルービックキューブが取りうる多くの状態においては、それぞれ多くの解き方が存在する。この複数の解法を、コンピュータの力を借りて検証する。

３．研究内容

３－１　仮説

いくつかの例について検証を行った昨年の研究結果から、平均してT2法の方がOld Pochmann法よりもかかる手数が20手ほど短いのではないかと推定した。

３－２　プログラムの作成

ルービックキューブを複数の解法で解くプログラムを作成し、解法ごとにかかる手数を出力する。今回は Python を用いて、T2法と Old Pochmann 法という2種類の解法でルービックキューブを解くプログラムを作成した。

３－２－１　キューブの定義

ルービックキューブの各面を図１のようにナンバリングし、Python の配列として定義する。

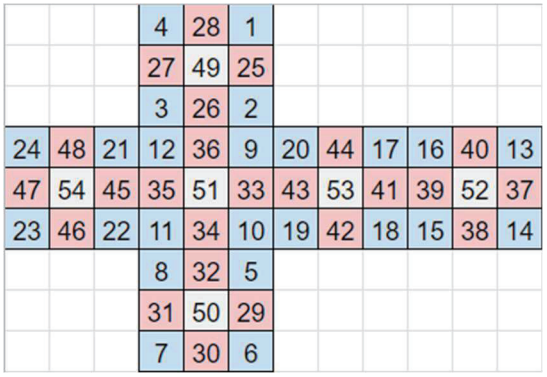

図１　Python における各面の配列

また、「シングマスター記法」という表記方法を利用して回転を記録する。ルービックキューブを置いたときに、上の面（U面）が白、下の面（D 面）が黄、手前の面（F 面）が緑、奥の面（B 面）が青、右の面（R 面）が赤、左の面（L 面）が橙になるようにする。このとき、各面を時計回りに90°回転する操作をそれぞれ U・D・F・B・R・L と表記することにする。また、各面反時計回りに90°回転させる操作を U′ のようにクォーテーションをつけて表し、各面を180°回転させる操作を U2のように、後ろに数字の2をつけて表記する。さらに、各面を2列同時に回す操作を Uw のように

w をつけて表記する。また、中央が F 面、上部が U 面になるように設定する。ここでは、リストの数値の置換によって回転を定義する。

例えば、U を行うと、(1 2 3 4)を交換、(25 26 27 28)を交換、(9 21 13 17)を交換、(12 24 16 20)を交換、(36 48 40 44)を交換することになる。

各面のステッカーの表記は、エッジパーツの場合、[ステッカーのある面、ステッカーのあるエッジパーツの他のステッカーがある面]というようにアルファベット2文字で表記する。コーナーパーツの場合、[ステッカーのある面、ステッカーのあるコーナーパーツの他のステッカー のある面1，ステッカーのあるコーナーパーツの他のステッカーのある面2]というようにアルファベット3文字で表記する。図 1 において、パーツの位置と名称 UFR ならば、前面と上面と右面を含むコーナーパーツの上の面のステッカーを、UF ならば、前面と上面を含むエッジパーツの上面のステッカーをそれぞれ表している。

３－２－２　スクランブルの生成

WCA（世界ルービックキューブ協会）の競技規則で十分な長さであるとされている、長さ25のランダムなスクランブルを生成するプログラムを作成し、回転の定義に則って実行することで、スクランブルどおりに配列を交換する（図 2 参照）。また、実際のルービックキューブと同様にスクランブルされているかどうかを確認するために、ユーザーインターフェースを実装し、表示されたスクランブルとキューブの状態が一致するかを確認した。ここで状態が一致したため、配列が正常に扱えていることが確認できた（図 3 参照）。

図 2　実行結果

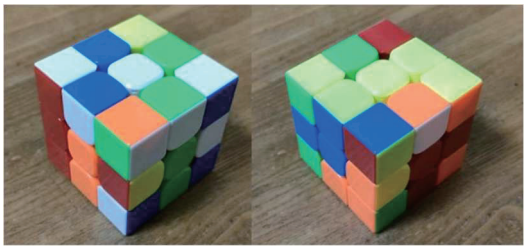

図 3　状態の一致の確認

３－２－３　ソルブ

スクランブルされたルービックキューブの配列を完成状態の配列と比較し、対応する手順を回すことでルービックキューブを解くプログラムを作成した。

＜キューブを解くプログラムの流れ＞

①揃っていないパーツのリストを作成する。

②揃っていないパーツの中から、対応する手順の実行手数（図 4 ）が少ないパーツを優先して選択し、パーツの位置に対応した、パーツ交換の手順を実行することで、配列を完成状態に近づける。

③揃っていないパーツのリストから、揃えたパーツの項目を削除する。

③かかった手数を加える。

④①～③を繰り返し、最終的に配列が完成状態と一致したとき、かかった手数を計

算する。

また図４のように、手数順に分類し、手順が最短になるように工夫した

T2法	
	エッジ
1	UL
2	FL,DL,BL
3	RF,RB,FR,BR,LF,LB,DF,DB,DR
4	UF,UB,LU,LD,FU,BU,FD,BD
5	DR
	コーナー
1	UFR
2	DFL,RDF,LUF
3	DFR,DBR,DBL,FDL,BDR,LDB
4	UFL,UBL,FUR,FUL,BDL,RUF,LDF
5	FDR,BUL,RDB,LUB
Old Pochmann法	
	エッジ
1	UL
2	FL,DL,BL
3	RF,RB,FR,BR,LF,LB,DF,DB,DR
4	UF,UB,LU,LD,FU,BU,FD,BD
5	DR
	コーナー
1	RDF
2	RUF,RUB,RDB,FDL,BDR,LDB,UFR,DFL,LUF
3	FUR,FUL,FDR,BUR,BDR,LDF,DBR,DBL, UBR,UFL,UFL,DFR

図４　パーツ優先順位

さらに、プログラムを実行した際に、プログラム内部でキューブが解けていることを確認するため、手順を表示し実際に手順を回してみることで、キューブが揃えられることを確認した。

```
M' D' L2 T-Perm L2 D M
38
M' D L2 T-Perm L2 D' M
40
Dw2 L' T-Perm L Dw2
39
Dw2 L T-Perm L' Dw2
33
L T-Perm L'
37
M D' L2 T-Perm L2 D M'
36
M2 D L2 T-Perm L2 D' M2
28
```

図５　出力された手順（一部抜粋）

３－３　比較

プログラムを1000回実行し、それぞれのケースについて、かかった手数をテキストファイルに記録し、その後分析した。かかった手数は、平均してそれぞれ約370手、365手であり、標準偏差はそれぞれ4.7，3.7であった。Old Pochmann 法のほうがわずかにかかる手数が短いことが確認できた。（図5および図６参照）。

	T2法	Old Pochmann法
平均値	371.4	365.0
標準偏差	4.7	3.7

図５　比較の結果

なお、図６は平均手数を表すグラフであり、横軸が試行回数、縦軸が平均手数を示している。

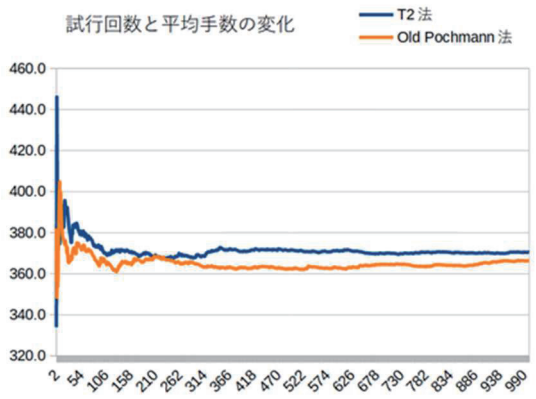

図６　平均手数のグラフ

４．考察

結果として、予想よりもかかる手数に大きな差はなかった。しかし、手数の安定性という視点から見ると、Old Pochmann 法が安定していると考えられる。昨年度の研究からサンプル数を増やすことで、より精密なデータを得ることができた。

５．参考文献・サイト

[1] 「Python Rubik's Cube Scrambler」, https://github.com/BenGotts/Python-Rubiks-Cube-Scrambler

[2] 「Python: 3次元配列のイメージ」, https://qiita.com/ken_yoshi/items/4cbe3abb7d46c5252fdd

[3] 「Python リスト内包表記の使い方」, https://note.nkmk.me/python-list-comprehension/

６．謝辞

本研究では、担当教諭である川口先生にご指導をいただきました。ありがとうございました。