CubeSEの開発
-Cube Solver and Explorer-

東京都立南多摩中等教育学校　5年　寺内駿

研究概要

ルービックキューブを人が揃えるための手順はコンピュータによって複数生成され，人が指で回しやすいと感じるものが選択されルービックキューブのコミュニティに広まってきた．しかし今もルービックキューブを揃える上で回しにくいと感じる手順が多い．そこでこの問題を解決するデスクトップアプリケーション開発した．また，初心者がルービックキューブを揃えることをサポートする機能を実装することにより，初心者から熟練者まで幅広いユーザーの確保を図った．その結果，Herbert Kociemba氏が開発したルービックキューブを揃える手順を提示するソフトウェアであるCube Explorerよりも多機能のものを開発することに成功した．作成したWindows専用アプリケーションはMITライセンスでGit Hubに公開した．

主な機能の説明

1．可能な限り短い解を探索する

※以降赤字はCube Explorerにはない独自の機能

① 指定できる項目
・混ぜ方　・解の長さの上限
・探索開始深さ　・探索秒数
② Start/Stopボタン
下の欄に実行結果が表示される
③ 探索する軸の選択
3つの軸から選択できる
全て並列処理することも可能である
④ ルービックキューブの状態の描画
混ぜた後のルービックキューブの状態を展開図で表示する
下のボタンで展開図を操作できる

図１ 可能な限り短い解を探索する

2．展開図に色塗って解かせる

① ルービックキューブの状態の指定
展開図をクリックして色を塗る
色を塗り間違えているとエラーを表示する
② Reset/Solved/Startボタン
展開図の色の初期化，探索の開始などができる
③ 回転記号の図解
ルービックキューブの回転を表す回転記号の意味を画像で確認できる
非表示にすることも可能である

図２ 展開図に色塗って解かせる

3．様々な手順を探索する

I.　PLLの手順を探索する

① 指定できる項目
・PLLの種類　・探索を始める深さ
・求める解の数　・Generator*の制限
・半回転の可否　・探索を始める向き
② Start/Stopボタン
下の欄に実行結果を表示する
③ 選択したPLLの表示
選択したPLLの画像を表示する

*Generatorとは手順を構成する回転の種類のこと

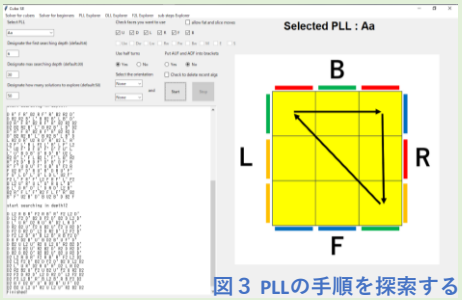


図３ PLLの手順を探索する

II.　OLLの手順を探索する

① 指定できる項目
・探索を始める深さ・求める解の数
・Generatorの制限　・半回転の可否
・探索を始める向き
② Start/Stopボタン
下の欄に実行結果を表示する
③ OLLの種類の指定領域
展開図をクリックして色を塗る
色を塗り間違えているとエラーを表示する

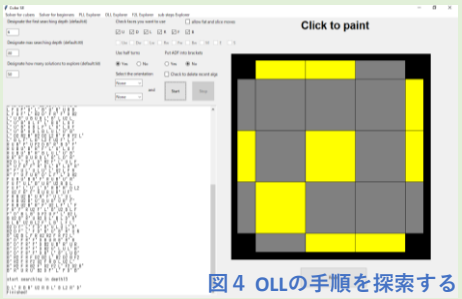


図４ OLLの手順を探索する

III.　F2Lの手順を探索する

① 指定できる項目
・探索を始める深さ・求める解の数
・Generatorの制限　・半回転の可否
・探索を始める向き
② Start/Stopボタン
下の欄に実行結果を表示する
③ F2Lの種類の指定領域
展開図をクリックして色を塗る
色を塗り間違えているとエラーを表示する
既に揃っている部分の指定もできる

図５ F2Lの手順を探索する

IV.　substepの手順を探索する

① 指定できる項目
・探索を始める深さ・求める解の数
・Generatorの制限　・半回転の可否
・探索を始める向き
② Start/Stopボタン
下の欄に実行結果を表示する
③ substepの種類の指定領域
探索したいsubstepの完成状態を選択
展開図をクリックして色を塗る
色を塗り間違えているとエラーを表示する

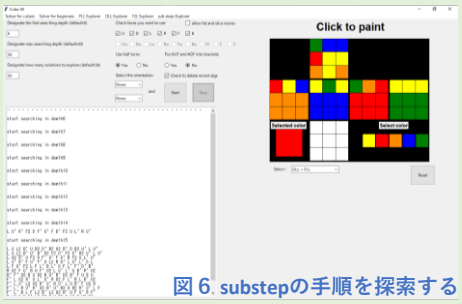


図６ substepの手順を探索する

高速化の研究

1．IDA*の言語による実行速度

CubeSEはルービックキューブを解くためにTwo-Phase AlgorithmやIDA*を用いている．当初GUIだけでなく，この探索部分もPythonで実装していたが高速化のために異なる言語で実装しその実行速度を比較した．

混ぜ方：F2 L2 U R2 F2 D2 U' F2 D2 F2 L2 U2 F' L U2 R D U B' D2 F
探索を始める深さ：7　　19手の解が見つかるまでの時間を測定

言語	Python	Cython	C++
20回平均(秒)	14.93652	15.43959	0.5266667

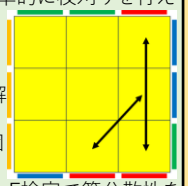
結果からC++が高速に動作することが分かったので探索部分の実装にはC++を用いた．

2．IDA*の枝刈りの効率化

IDA*の枝刈りには，ルービックキューブを７個の要素に分解しその中から選んだ任意の２個または３個の要素のみを揃えるには最短で何手かかるかという情報を格納した表(以降**枝刈り表**と呼ぶ)を用いている．組み合わせる要素によって枝刈り表から無作為に値を取り出したときの平均値(以降**枝刈り表の期待値**と呼ぶ)が変化するためすべての要素の組み合わせで枝刈り表を作成し，その期待値を求めた．PLLの一部の手順を探索するときに使える枝刈り表の期待値を降順に５つ抜粋して示す．

組み合わせる要素	格納される値の数	μ(期待値)
CO_CP	88179840	8.764133684071
CP_MEP	479001600	6.994375941541
CP_EEP	479001600	6.994375941541
CP_SEP	479001600	6.994375941541
CO_EO_CENTER	107495424	6.724048895327

枝刈り表の期待値は枝刈り表の性能を評価するための指標の１つであるので，期待値が高い枝刈り表をどのように組み合わせて使用すれば最も効率的に枝刈りを行えるかを，実際に手順を探索させてその最終的な評価を決定した．



実験方法
1．抜粋した５個の枝刈り表を全て使用した状態で，Jbという PLL(右図)について，探索を始める深さを７とし，2000個の解が見つかるまでの実行時間を５回計測し平均をとる
2．５個の枝刈り表から１個使用しないものを選び，枝刈り表４個での実行時間の５回の平均をとる
3．枝刈り表を減らす前後で実行時間に有意差があるかについて，F検定で等分散性を確認してから有意水準をP<0.05としてT検定で評価する．高速化していた場合は，最も高速化したときに使用しなかった枝刈り表を1個削除し再びその状態での実行時間を計測しその平均をとる．
4．枝刈り表を１個ずつ削除しながら２と３を高速化しなくなるまで繰り返す．

実験結果
枝刈り表５個の時の結果　　※５分経過しても探索が終了しない場合DNFと記した

取り除く表	実行時間(秒)	訪問ノード数	NPS
なし	2.3217	50283550	21683470

高速化ができなくなった枝刈り表２個のときの結果

取り除く表	実行時間(ms)	訪問ノード数	NPS	P値
CO_CP	DNF	DNF	DNF	DNF
CP_MEP	2348.1	132940097	56866800	1.39E-06
CO_EO_CENTER	5487.8	298248118	54360169	2.44E-17

実験結果より使用する枝刈り表をCO_CP，CP_MEP，CO_EO_CENTERの３個とした．

3．枝刈り表の読み込みの高速化

枝刈り表は外部ファイルに保存している．そこで保存方式を変えたときの読み込み速度の変化を比較した．比較には最もファイルサイズの大きい枝刈り表を用いた．

表の名前	要素数	読込時間 JSON	読込時間 CSV	読込時間 バイナリ	読込時間 圧縮バイナリ
CP_MEP	479001600	56701.7[ms]	2066.3[ms]	102[ms]	56[ms]

圧縮バイナリとは，格納される値がすべて４bitであることを利用して１byteに２つ値を格納したものである．結果からこの方式が最も高速に読み込めることが分かったので，枝刈り表の保存には圧縮バイナリ形式を用いた．

今後の展望

知名度が低くまだ多くのフィードバックが得られていないので，今後は知名度を向上させていきたい．また，アプリケーションサイズを性能を落とさず小さくすること，クロスプラットフォームにすること，３×３×３以外のルービックキューブにも対応できるようにすることを次の課題として考えている．更なる機能の拡充を図りたい．

謝辞

本研究を行うにあたり，筑波大学理工学群工学システム学類山名琢翔様にご指導いただきました．ここに深謝の意を表します．またCubeSEのロゴを作成していただいた山本仰様，フィードバックを寄せてくださった全ての方々に深く感謝申し上げます．

参考文献

7y2n．ルービックキューブを解くプログラムを書いてみよう(後編:状態のindex化，Two-Phase-Algorithm)．Qiita．2021-10-17．
https://qiita.com/7y2n/items/a840e44dba77b1859352．(参照 2021-04-22)
Herbert Kociemba．Cube Explorer．Version 5.14，2020-08-19．
https://github.com/hkociemba/CubeExplorer．(参照 2021-04-22)