群論の可視化教材

大阪府立佐野工科高等学校　松川　信彦
Nobuhiko Matsukawa, Osaka Prefectural Sano Technology High School

1　はじめに

著者は GAP や magma など既存の計算群論ソフトを利用せず, JavaScript のみのフルスクラッチで計算群論の基礎的なアルゴリズム (Schreier-Sims, coset enumeration など) を実装し, 群論の実例計算や, その応用として rubik's cube やスライドパズルのソルバーなどの可視化アプリケーション (Three.js による) を自身のサイトにおいて公開している. 取り組み始めたのは 2018 年 8 月頃からであり, 当初は Schreier-Sims アルゴリズムによる, 置換群の BSGS の導出を目指して開始し, Jerrum's filter による効率化と, Minkwitz アルゴリズムによる SGS を最初に与えられた生成系たちの積へ分解する実装までを, 2018 年以内に完了した. 例えば, Mathieu 群 M_{24} に適用し, 家庭用 PC (Intel(R) Core(TM) i3-3220 CPU @ 3.30GHz) で計算してみたところ, 0.8 秒で BSGS と, その Minkwitz による分解までを返すところまでは達しているが, $3\times3\times3$ の rubik's cube 群のそれを計算するには 4 分かかり, $4\times4\times4$ については 16 時間もかかることから, BSGS を一度導出すれば, web ブラウザ上での再利用が可能となるように, web storage に BSGS および Minkwitz 分解の結果を自動的に保存する実装である. 更に, 計算が大規模になることが前提であるため Web Workers (ブラウザのバックグラウンドで JavaScript に計算させる機能) を利用している.

群論計算の有用性を顕在化させる強力な方法の 1 つに置換パズルへの応用がある. JavaScript ライブラリである Three.js を用いて, rubik's cube や 24 puzzle の 3D モデルと, キューブの表面をクリックすることにより意図した方向へ回転したり, スライドしたりする操作を実装し, web page [12] [14] において公開している. 特に立体を回転させる直感的な操作の実装は難しく, web page [15] が参考になった. web page [15] では多くの cube パズルを Three.js によって実装し, ツクダ式など, 有名な解法を実装しているが, 著者の目指しているところは, 純粋に置換群論的な手法のみによる解法である. シャッフルされたパズルに対し, それを元の状態に戻す操作は, ランダムに与えられた群の元に対して先に述べた手法によって得られた Minkwitz 分解を利用すれば得られるが, そのアニメーションまでも実装した.

Minkwitz の論文 [8] では, BSGS を元の生成系の積へ分解するための実装を目指しているにも関わらず, relator に関するアルゴリズムである coset enumeration や項書換えに関する Knuth-Bendix アルゴリズムへの言及が一切なされていない. Minkwitz アルゴリズムの理解をより深めるためでもあるが, 著者は coset enumeration の JavaScript への実装を 2019 年の春頃より開始した.

coset enumeration とは 1936 年に Todd と Coxeter が考案し, 別名 Todd-Coxeter アルゴリズムとも呼ばれている. 1953 年に Haselgrove により初めてコンピュータで実

装されたアルゴリズムであり, 既に計算群論の古典と呼んでも良いと思われる. coset enumeration を含む計算群論の歴史について日本語の文献 [4] で詳細に知ることができる. 特に, 80 年代前半から 90 年代前半にかけての専門書 [6] [13] には, 当時はまだ高性能な計算機が希少であったこともあり, JavaScript の実装のベンチマークに適しているが, 自明ではない実例が豊富に含まれているように著者には思われた. 著者は coset enumeration アルゴリズムのうち, HLT method を実装し, 実例計算アプリを web 上で公開している.

2 基礎的な事項と準備

2.1 BSGS

置換群を実装する上で必要となる基礎的で有名な事実を紹介していく. $\Omega = \{0, 1, \cdots, n-1\}$ 上の対称群を $\mathfrak{S}_\Omega$ と表し, G は $X \subseteq \mathfrak{S}_\Omega$ により生成される部分群と仮定する. Ω は作用域とも呼ばれる. $\alpha \in \Omega$ への $g \in G$ の作用は右からとし, α^g と表すことにする. $\alpha_0, \alpha_1, \cdots, \alpha_{l-1} \in \Omega$ に対し, その固定部分群を $G_{\alpha_0,\alpha_1,\cdots,\alpha_{l-1}}$ と表す. すなわち

$$G_{\alpha_0,\alpha_1,\cdots,\alpha_{l-1}} = \{g \in G \mid {\alpha_i}^g = \alpha_i \ \forall i \in \{0, 1, \cdots, l-1\}\}$$

とおく. $B = (\beta_0, \beta_1, \cdots, \beta_{k-1}) \in \Omega^{\times k}$ が base であるとは, $G_{\beta_0,\beta_1,\cdots\beta_{k-1}} = \{e\}$ を満たすものと定義する. 特に $(0, 1, \cdots, n-1)$ も base である. base B に対し, $G^{(i)} := G_{\beta_0,\beta_1\cdots,\beta_{i-1}}$ とおくとき,

$$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(k)} = \{e\}$$

を stabilizer chain であるという. base B が重複なしであるとは, $G^{(i)} \neq G^{(i+1)}$ が全ての $i \in \{0, 1, \cdots, k-1\}$ について成り立つことをいう. $G^{(i+1)}$ の $G^{(i)}$ における右剰余類の完全代表系を $U^{(i)} (\subseteq G^{(i)})$ とおく. $U^{(i)}$ の取り方は同一の base B に対し色々あるが, 軌道 $\Delta^{(i)} := {\beta_i}^{G^{(i)}} \simeq G^{(i+1)} \backslash G^{(i)}$ は base B に対し一意に取れる. 著者の実装は全単射 $u_i : \Delta^{(i)} \to U^{(i)}$ $(i \in \{0, 1, \cdots, k-1\})$ で任意の $\beta \in \Delta^{(i)}$ に対し, $\beta = {\beta_i}^{u_i(\beta)}$ が成り立つものを何か一組構成するアルゴリズムとなっている. このとき, base と $u^{(i)}$ たちを利用して, 任意の $x \in \mathfrak{S}_n$ に対し, x が G の元であるかどうかを判定し, もしそうであるならば $U^{(i)}$ の元の積に一意的に表示することも可能である. 実際, $G^{(1)}x = G^{(1)}s_0$ となるような $s_0 \in U^{(0)}$ を取り, $x{s_0}^{-1} \in G^{(1)}$ となるので, $G^{(2)}x{s_0}^{-1} = G^{(2)}s_1$ となる $s_2 \in U^{(2)}$ が取れる. 以下同様の操作で $x{s_0}^{-1}{s_1}^{-1} \cdots {s_{l-1}}^{-1} \in G^{(l)}$ であるが, $x{s_0}^{-1}{s_1}^{-1} \cdots {s_{l-1}}^{-1} \notin G^{(l+1)}$ ならば x は G の元ではなく, $l = k$ ならば, $x{s_0}^{-1}{s_1}^{-1} \cdots {s_{k-1}}^{-1} = e$ だから, $x = s_{k-1} \cdots s_1 s_0$ と表される. 特に, 集合として

$$G^{(i)} = U^{(k-1)} \times \cdots \times U^{(i)} \quad (\forall i \in \{0, 1, \cdots k-1\})$$

である. $G = U^{(k-1)} \times \cdots \times U^{(1)} \times U^{(0)} \simeq \Delta^{(k-1)} \times \cdots \times \Delta^{(1)} \times \Delta^{(0)}$ なので, G の位数が $\prod_{i=0}^{k-1} |\Delta^{(i)}|$ となる.

$B \subseteq G$ が strong generating set (以下 SGS) であるとは,

$$\langle S \cap G^{(i)} \rangle = G^{(i)} \quad (\forall i \in \{0, 1, \cdots k-1\})$$

が成り立つことをいう. base B とその SGS S との 組 (B, S) を BSGS と呼ぶ. BSGS (B, S) が構成されることは, 或る全単射 $u_i : \Delta^{(i)} \to U^{(i)}$ で $\beta = \beta_i^{u_i(\beta)}$ $(\forall \beta \in \Delta^{(i)})$ を満たすものが構成でき, $S \subseteq \bigsqcup_{i=0}^{k-1} U^{(i)}$ となり, $S^{(i)} = S \cap (U^{(i)} \sqcup \cdots \sqcup U^{(k-1)})$ とおくとき,

$$G^{(i)} = \langle S^{(i)} \rangle \quad (\forall i \in \{0, 1, \cdots, k-1\})$$

が成り立つことと同値である.

2.2 Jerrum's filter

現時点では実際に扱う例は $|\Omega|$ が 100 を超えることも少ないため, 各右剰余類の完全代表系 $U^{(i)}$ の大きさも小さい. そのため, SGS として

$$S = U^{(0)} \sqcup \cdots \sqcup U^{(k-1)}$$

を求める実装にしている. また, $B = (0, 1, \cdots, n-1)$ を取って計算することが多い. X が対称群全体を生成しなければ, 途中で $G^{(k)} = \{e\}$ となったり, $G^{(i)} = G^{(i+1)}$ ($\Leftrightarrow$ $\Delta^{(i)} = \{i\}$) となることがあるのだが, このような i は計算終了後に排除すればよい. base B としては, 増加列 $0 \leq \beta_0 < \beta_1 < \cdots < \beta_{k-1} \leq n-1$ で $G = G^{(0)} > G^{(1)} > \cdots > G^{(k-1)} = \{e\}$ となる.

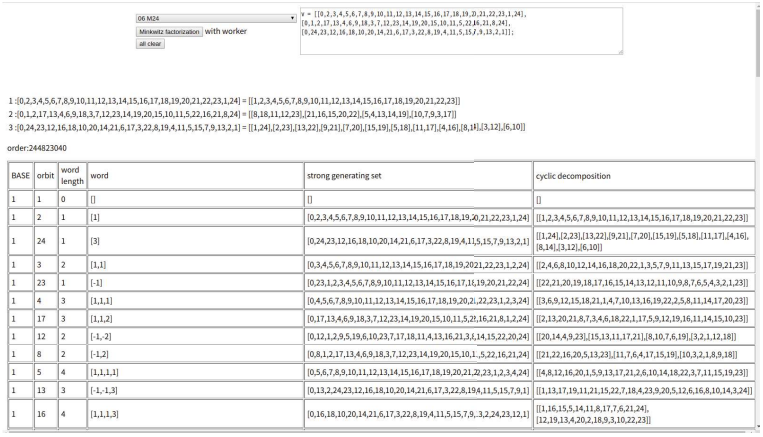
具体的な実装は次のような $i = 0, 1, \cdots, k-1$ に関するループである. base $B = (\beta_0, \beta_1, \cdots, \beta_{k-1})$ に対し, $G^{(i)} = G_{\beta_0, \cdots, \beta_{i-1}}$ の生成系 $S^{(i)}$ が構成されたと仮定する ($i = 0$ のときは最初の生成系). $U^{(i)}$ は, β_i に $S^{(i)}$ の元を次々に作用させ, 軌道 Δ_i を張るとき, それと同時に作用させた元たちを掛けていったものを記録していけば構成できる. 従って, 全単射 $u_i : \Delta^{(i)} \to U^{(i)}$ が構成されたことになる. 次に $G^{(i+1)}$ の構成の仕方について述べる. よく知られた Schreier の補題を使って $G^{(i+1)}$ の生成系は

$$\{u_i(\beta) x u_i(\beta^x)^{-1} \mid \beta \in \Delta^i,\ x \in X^{(i)}\}$$

となることが分かる. 単純にこのまま実装すれば, Mathieu 群 M_{12} のような小さな群ならば機能するが, i が増加するに従って, この生成系の大きさは爆発的に大きくなり, JavaScript では Mathieu 群 M_{24} でもオーバーフローを起こしてしまい使い物にならない. 従って, この大きさを $|\Omega| - 1$ 以下に抑えるためのアルゴリズムである Jerrum's filter が必要となる. Cameron [3], 花木 [5] を参考に実装した.

3 Minkwitz 分解アルゴリズム

Minkwitz の論文 [8] を参考に実装した. Minkwitz の論文は 7 ページと短いが, 著者にとって難解であったため web page [7] を参考に試行錯誤を重ね, とりあえず動くものができたというのが現実である. 著者のサイト [9] において, 下の図のように Mathieu 群 (M_{11}, M_{12}, M_{22}, M_{23}, M_{24}) などの置換群に対して試すことができる.



BASE	orbit	word length	word	strong generating set	cyclic decomposition
1	1	0	[]	[]	[]
1	2	1	[1]	[0,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,1,24]	[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]]
1	24	1	[3]	[0,24,23,12,16,18,10,20,14,21,6,17,3,22,8,19,4,11,5,15,7,9,13,2,1]	[[1,24],[2,23],[13,22],[9,21],[7,20],[15,19],[5,18],[11,17],[4,16],[8,14],[3,12],[6,10]]
1	3	2	[1,1]	[0,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,2021,22,23,1,2,24]	[[2,4,6,8,10,12,14,16,18,20,22,1,3,5,7,9,11,13,15,17,19,21,23]]
1	23	1	[-1]	[0,23,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,1t,19,20,21,22,24]	[[22,21,20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,23]]
1	4	3	[1,1,1]	[0,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,2(,22,23,1,2,3,24]	[[3,6,9,12,15,18,21,1,4,7,10,13,16,19,22,2,5,8,11,14,17,20,23]]
1	17	3	[1,1,2]	[0,17,13,4,6,9,18,3,7,12,23,14,19,20,15,10,11,5,2!,16,21,8,1,2,24]	[[2,13,20,21,8,7,3,4,6,18,22,1,17,5,9,12,19,16,11,14,15,10,23]]
1	12	2	[-1,-2]	[0,12,1,2,9,5,19,6,10,23,7,17,18,11,4,13,16,21,3,{,14,15,22,20,24]	[[20,14,4,9,23],[15,13,11,17,21],[8,10,7,6,19],[3,2,1,12,18]]
1	8	2	[-1,2]	[0,8,1,2,17,13,4,6,9,18,3,7,12,23,14,19,20,15,10,1_,5,22,16,21,24]	[[21,22,16,20,5,13,23],[11,7,6,4,17,15,19],[10,3,2,1,8,9,18]]
1	5	4	[1,1,1,1]	[0,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,2(,23,1,2,3,4,24]	[[4,8,12,16,20,1,5,9,13,17,21,2,6,10,14,18,22,3,7,11,15,19,23]]
1	13	3	[-1,-1,3]	[0,13,2,24,23,12,16,18,10,20,14,21,6,17,3,22,8,19{,11,5,15,7,9,1]	[[1,13,17,19,11,21,15,22,7,18,4,23,9,20,5,12,6,16,8,10,14,3,24]]
1	16	4	[1,1,1,3]	[0,16,18,10,20,14,21,6,17,3,22,8,19,4,11,5,15,7,9,_3,2,24,23,12,1]	[[1,16,15,5,14,11,8,17,7,6,21,24], [12,19,13,4,20,2,18,9,3,10,22,23]]

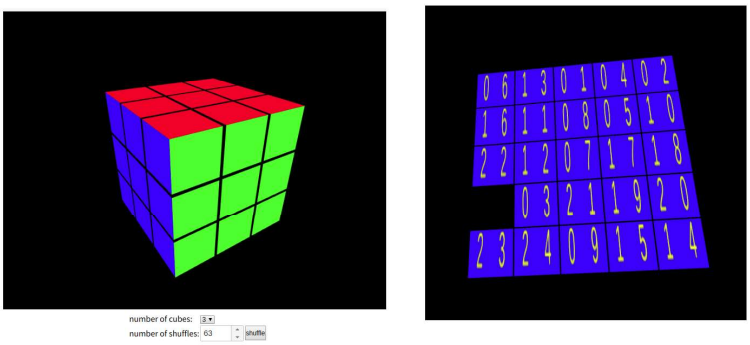
[https://noblegarden-math.jp/math/notes/computer_algebra/MinkwitzFactorization/]

置換パズルの群論的な solver は予め計算された BSGS とその Minkwitz 分解を利用して実行する. 任意の元を与えられた SGS の積で表示する実装は容易である上, JavaScript であっても非常に高速であり低コストである. それに対し BSGS を導出するコストは高く, 著者の JavaScript での実装において, $3 \times 3 \times 3$ の rubik's cube 群の BSGS を導出するには 4 分もかかる. 従って GAP や magma などで計算した BSGS とその Minkwitz 分解を配列データとして JavaScript に書いて利用する方が, より効率的で洗練された解法を提示することが可能であるが, 著者の目指しているのは, BSGS の導出を始めとした群論アルゴリズムの理解を深めることと, それを利用し様々な具体的な群を調べることであるから, その道を取らなかった. せっかく作成した BSGS が実際にどのような挙動を取るのかを可視化したい思いもあった.

著者の実装では $3 \times 3 \times 3$ の rubik's cube 群で, 各面の中心を固定し, 90° 回転させる計 6 個の変換から生成されるものについて, その位数が 43252003274489856000 と正しく計算でき, Minkwitz 分解の最大長が 194 という結果であった. これは [7] の 184 や Minkwitz の 144 に比べてかなり劣る.

4 Three.js での実装

和島茂氏のサイト「バーチャル 3D パズル」[15] において, 多種多様な cube puzzle とその solver が実装されている. 著者も自力で置換パズルの 3D モデルを実装したい情熱に駆られ, 特定非営利活動法人 natural science のサイト [11] にあるサンプルコードを改変することから開始し, rubik's cube や スライドパズルの描画を実装した. rubik's cube 群や 24 puzzle 群などをマウスクリックによって, 直感的に回転やスライドを自在に制御できる実装である. また, 与えられた操作の語に対して, その挙動をアニメーションで表現できる. 24 puzzle 群の生成系には 4 つの loop generator を使っており, それらのみで実際にパズルが解かれる様子を観察することができる.



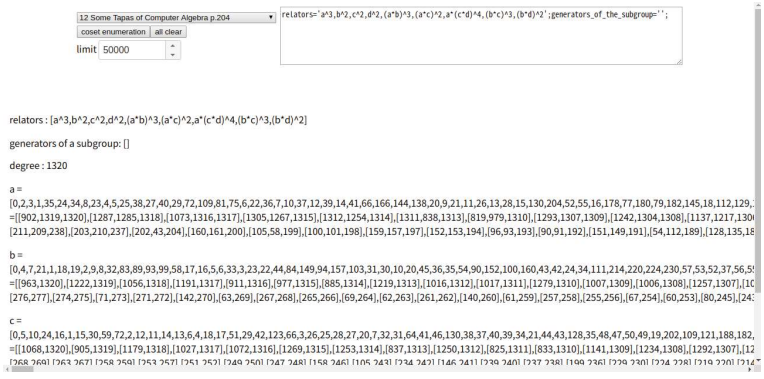
[https://noblegarden-math.jp/math/three_rubiks.html] [https://noblegarden-math.jp/math/three_24puzzle.html]

置換の操作は固定される部分が多ければ多いほど価値があることがわかる. 特に, $4 \times 4 \times 4$ の rubik's cube において, 6 面全て揃っているように見えても, ある面の中央の 4 個のピースに置換が発生する操作があるなど, 面白い操作をいくつか確認することができる.

5 Coset Enumaration

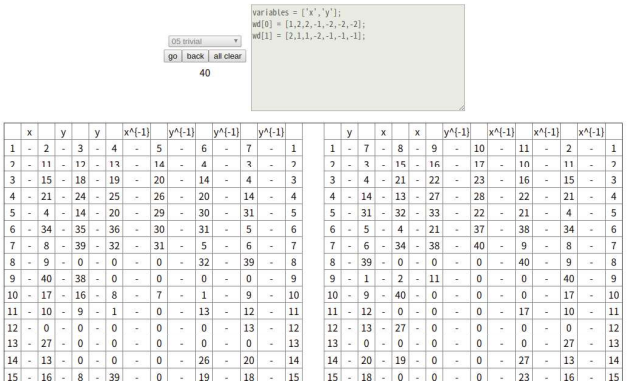
coset enumeration とは, 群の表示及びその部分群の生成系 relator table と coset table を構成するアルゴリズムであり, これら table たちに空きが無くなった時点で終了する. 空きがあれば新しい数を埋めるのであるが, それが relator table であるとき HLT method と呼び, coset table であるとき Felsch method と呼ぶ. 正確には, 群 G が表示を持つとは, 集合 X と X から生成される自由群 $F(X)$ の (有限) 部分集合 R, $F(X)$ における R の正規化部分群 $N(= N_{F(X)}(R))$ によって $G = F(X)/N$ と表されることであり, この状況を $G = \langle X|R\rangle$ と表す. coset enumeration とは, 表示を持つ群 $G = \langle X|R\rangle$ とその部分群 H について, $H = \langle Y\rangle + N/N$ (for some $Y \subseteq F(X)$) となるとき, R と Y を使って準同型 $G \to \mathfrak{S}(H\backslash G)$ を構成するアルゴリズムのことである. ここで, $\mathfrak{S}(H\backslash G)$ とは $H\backslash G$ 上の置換全体の成す群のことである.

筆者は HLT method を JavaScript で実装し, 可視化教材を web page において公開している.



[https://noblegarden-math.jp/math/notes/computer_algebra/CosetEnumeration/]

具体的にどのように relator table と coset table が成長していくのか, step by step で可視化するアプリも公開している.



[https://noblegarden-math.jp/math/notes/computer_algebra/coset_enumeration_step_by_step/]

6 まとめ, 次の目標

- Jerrum's filter によって, $G^{(i)}$ の生成系の個数を抑えるアルゴリズムは BSGS を導出する上で最も高コストであり, CPU と時間を多大に消費する. 著者の実装を chrome browser の console 画面で観察すると, 生成系の個数がたった 800 個であっても数十秒はかかっている. JavaScript だから遅いのか, 著者の実装が非効率なのか, 未だに良く理解していない. Jerrum's filter の効率化・高速化とその限界について, 更に理解を深めたい.

- 有限体上の行列群など, 作用域が巨大な置換群について, その BSGS を求める実装に取り組みたい. 作用域が巨大な場合, deterministic な方法は非常に高コストで時間がかかるため, 確率論的な方法を用いなくてはならない.

- BSGS の計算で Todd-Coxeter Schreier-Sims アルゴリズムというものが Murray の解説 [10] にあり, これは coset enumeration を取り入れた方法とのことで, そちらへの応用にも取り組んでみたい.

- coset enumeration の高速化. HLT method における coset table 構成中に同じ箱の中に異なる数が入ることがあり, これを修正するアルゴリズムの実装が複雑なので, より簡潔・効率化したい.

- 既に古典となった Reidemeister-Schreier アルゴリズムによる部分群表示導出の実装

参考文献

[1] Gregory Butler, Fundamental Algorithms for Permutation Groups (Lecture Notes in Computer Science), Springer, 1991.

[2] ca.js,

https://noblegarden-math.jp/math/notes/computer_algebra/01/

[3] P. J. Cameron, Permutation groups (London Mathematical Society Student Texts, vol. 45, Cambridge University Press), Cambridge, 1999.

[4] 藤本光史, Computational Group Theory への招待, 数理解析研究所講究録 941 巻, 1996 年, 73-83.

[5] 花木章秀, 群論と対称性,

http://math.shinshu-u.ac.jp/~hanaki/edu/symmetry/perm.pdf

[6] D. L. Johnson, Topics in the Theory of Group Presentations, (London Mathematical Society Lecture Note Series, vol. 42, Cambridge University Press), Cambridge, 1980.

[7] Mathematics_and_Such,

https://mathstrek.blog/2018/06/21/solving-permutation-based-puzzles/

[8] T. Minkwitz, An Algorithm for Solving the Factorization Problem in Permutation Groups, J. Symbolic Computation (1998) 26, 8995

[9] Minkwitz 分解の実装,

https://noblegarden-math.jp/math/notes/computer_algebra/MinkwitzFactorization/

[10] Scott H. Murray, The Schreier-Sims algorithm, 2003,

http://www.maths.usyd.edu.au/u/murray/research/essay.pdf

[11] 特定非営利活動法人 natural science, コンピュータ・シミュレーション講座 Tips 集,

http://www.natural-science.or.jp/laboratory/computer_tips.php

[12] rubik's cube solever,

https://noblegarden-math.jp/math/three_rubiks.html

[13] C.C.Sims, Computation with finitely presented groups, Cambridge University Press, 1994.

[14] 24 puzzle solver,

https://noblegarden-math.jp/math/three_24puzzle.html

[15] 和島茂, バーチャル 3D パズル,

http://aows.jp/jspuzzle/