

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、 $3 \times 3 \times 3$ の rubik's cube 群のそれを計算するには 4 分かかり、 $4 \times 4 \times 4$ 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。 $\Omega = \{0, 1, \dots, n-1\}$ 上の対称群を \mathfrak{S}_Ω 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。 $\alpha_0, \alpha_1, \dots, \alpha_{l-1} \in \Omega$ に対し、その固定部分群を $G_{\alpha_0, \alpha_1, \dots, \alpha_{l-1}}$ と表す。すなわち

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

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

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

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 $i \in \{0, 1, \dots, k-1\}$ 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 $u_i : \Delta^{(i)} \rightarrow U^{(i)}$ ($i \in \{0, 1, \dots, k-1\}$) 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、 $xs_0^{-1} \in G^{(1)}$ となるので、 $G^{(2)}xs_0^{-1} = G^{(2)}s_1$ となる $s_2 \in U^{(2)}$ が取れる。以下同様の操作で $xs_0^{-1}s_1^{-1} \dots s_{l-1}^{-1} \in G^{(l)}$ であるが、 $xs_0^{-1}s_1^{-1} \dots s_{l-1}^{-1} \notin G^{(l+1)}$ ならば x は G の元ではなく、 $l = k$ ならば、 $xs_0^{-1}s_1^{-1} \dots s_{k-1}^{-1} = e$ だから、 $x = s_{k-1} \dots s_1 s_0$ と表される。特に、集合として

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

である。 $G = U^{(k-1)} \times \dots \times U^{(1)} \times U^{(0)} \simeq \Delta^{(k-1)} \times \dots \times \Delta^{(1)} \times \Delta^{(0)}$ 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、

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

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 $u_i : \Delta^{(i)} \rightarrow U^{(i)}$ 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, $S^{(i)} = S \cap (U^{(i)} \sqcup \dots \sqcup U^{(k-1)})$ とおくと,

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

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

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

を求める実装にしている. また, $B = (0, 1, \dots, n-1)$ 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 $0 \leq \beta_0 < \beta_1 < \dots < \beta_{k-1} \leq n-1$ で $G = G^{(0)} > G^{(1)} > \dots > G^{(k-1)} = \{e\}$ となる.

具体的な実装は次のような $i = 0, 1, \dots, k-1$ に関するループである. base $B = (\beta_0, \beta_1, \dots, \beta_{k-1})$ に対し, $G^{(i)} = G_{\beta_0, \dots, \beta_{i-1}}$ 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 $u_i : \Delta^{(i)} \rightarrow U^{(i)}$ 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

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

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, 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.

- [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] 花本章秀, 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 solver,
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/>