JavaScript で CGT
Using JavaScript for CGT

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

JavaScript のみを用い, 置換群の BSGS とその minkwitz 分解を構成するアルゴリズムを実装した。その応用として rubik's cube や 24 puzzle の solver の構成と Three.js を用いた可視化を行った.

Abstract

Using only JavaScript, we implemented an algorithm to construct a BSGS of a permutation group and its minkwitz decomposition. As its application, we constructed a solver of rubik's cube (and 24 puzzle) and visualized it using Three.js.

1 概要

著者は GAP や magma など既存の計算群論ソフトを利用せず, JavaScript のみで, 計算群論 の基礎的なアルゴリズムを実装し, 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を Three.js によって可視化したアプリケーションたちを, 自身のサイトにおいて公開するまでを目的とし, 2018 年 8 月頃から取り組み始めた. そのため, とりあえず問題なく動く実装であり, アルゴリズムの練り直しによる, 効率化や高速化は次の課題としている. 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 Schreier-Sims アルゴリズムの効率化と, Minkwitz アルゴリズムによる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.

さらに 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 cite [9] [10] において公開している. 特に立体を回転させる操作の実装は難しく, web cite [11] の実装が参考になった. web cite [11] 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.

*noblegarden@gmail.com

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

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

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 [2], 花木 [4] を参考に実装した. 証明がそのまま実装に適用できるので, その概略を述べる.

命題 1
Jerrum's filter によって, 対称群 $\mathfrak{S}_n=\mathfrak{S}_{0,1,\cdots n-1}$ の部分群の生成系は個数を $n-1$ 以下にすることができる.

対称群 $\mathfrak{S}_n=\mathfrak{S}_{0,1,\cdots,n-1}$ の部分群 $G=\langle X\rangle$ について考える. 任意の $g\in X$ に対して, $i^g=i$ ($i=0,1,\cdots s-1$), $s^g=t>s$ となるような数の組 $e(g):=\{s,t\}$ が取れる. また, $i(g):=s$ と定義する.

$$\Gamma(X):=\{e(g)\mid g\in X\})$$

は X の元によってラベル付けられた無向グラフを成す. $\Gamma(X)$ には重複辺が存在することがある. 即ち $g\neq h$ であり $e(g)=e(h)$ となることがあるが, X と $X':=(X\setminus\{g\})\cup\{gh^{-1}\}$ は同じ群を生成し, $\Gamma(X')=\Gamma(X)\setminus\{e(g)\}\cup\{e(gh^{-1})\}$ なので, 先の重複辺が消去されたものとなっている. この操作を繰り返して, X と同じ群を生成するが, 重複辺を持たないものを構成できる. この操作を関数 remove_multiple_edge として実装し, $\Gamma(X)$ には重複辺が含まれない場合に帰着する. グラフ理論の基本的事実により, $\Gamma(X)$ には全域木が存在し, それを $\Gamma(Y)=\{e(g)\mid g\in Y\}$ とする. 任意に取った $g\in X\setminus Y$ に対し $\Gamma(Y\cup\{g\})$ には唯一つのサイクルが存在する. このサイクルを点集合で $c=(v_0,v_1,\cdots,v_{k-1},v_0)$ と表す. ただし, v_0 は c の中で最小とする. $e(g_i)=\{v_i,v_{i+1}\}$ となるような $g_i\in X$ が存在する. このとき $\varepsilon_i\in\{-1,1\}$ を, $v_i<v_{i+1}$ ならば $\varepsilon_i=1$, $v_i>v_{i+1}$ ならば $\varepsilon_i=-1$ と定義して. $g(c):={g_0}^{\varepsilon_0}\cdots{g_{k-1}}^{\varepsilon_{k-1}}$ とおく. $v_0<i(c(g))$ となる. $g(c)$ が単位元と異なるとき, $Z:=(Y\cup\{g\})\setminus\{g_0\}\cup\{g(c)\}$ とおき, $g(c)$ が単位元であるならば, $Z:=(Y\cup\{g\})\setminus\{g_0\}$ とおくと, Z と $Y\cup\{g\}$ は同一の群を生成する. $\Gamma(Z)$ から重複辺を取り除いた結果, サイクルが発生すれば先と同様の操作を施す. この操作の繰り返しが終結することは, Cameron の著書 [2] において証明されている. $\Gamma(Z)$ にはサイクルが存在しないが, Z と $Y\cup\{g\}$ は同一の群を生成する. これを $X\setminus Y$ の元全てについて繰り返せば, $\Gamma(Z)$ にはサイクルが存在しないが, X と同一の群を生成するような Z が構成できる. $\Gamma(Z)$ は全域木なので, $|Z|<n$ となる.

2.3 Minkwitz 分解

置換群 G とそのオリジナルの生成系 $X=\{x_1,\cdots,x_n\}$, 更にその base $B=(\beta_0,\cdots,\beta_{k-1})$ と軌道 $\Delta^{(i)}={\beta_i}^{G^{(i)}}=\{\beta_{i,0},\cdots,\beta_{i,r_i-1}\}$ が与えられているとする. これは, 先に述べた Schreier-Sims アルゴリ

ズムによって得られる. Minkwitz のアルゴリズムは SGS の元を, 最初の生成系 X の元たちの積に表すアルゴリズムのことで, この分解のことを Minkwitz 分解と呼ぶことにする. 著者は web 上にある Minkwitz の原論文 [6] と cite [5] の解説を参照して実装した. Minkwitz の原論文 [6] の方法は heuristic であり, 著者にとって理解し切ることは難しく, 大まかなアイデアと, 著者なりの工夫と実験により得られた実装の紹介となる. 集合 $X = \{x_1, \cdots, x_n\}$ に対し自由群 $F(X)$ を

$$F(X) = \{[a_1, \cdots, a_m] \mid m \geq 0,\ a_1, \cdots, a_m \in \{\pm 1, \cdots, \pm n\}\}$$

と表す. 但し, $F(X)$ は $a_i + a_{i+1} = 0$ のとき,

$$[a_1, \cdots, a_m] = [a_1, \cdots, a_{i-1}, a_{i+2}, \cdots, a_m]$$

が成り立つように同値関係割った商集合であり, 演算は

$$[a_1, \cdots, a_m][b_1, \cdots, b_k] = [a_1, \cdots, a_m, b_1, \cdots, b_k]$$

と定義する. 単位元は [ ] であり, $w = [a_1, \cdots, a_m] \in F(X)$ の逆元は $w^{-1} = [-a_m, \cdots, -a_1]$ である. 自由群の元 $w = [a_1, \cdots, a_m]$ に対し, このような m のうち最小のものを w の長さといい, $l(w)$ と表す. $m = l(w)$ のとき, w は簡約であるという. $w = [a_1, \cdots, a_m]$ に対して,

$$\pi(w) := x_{|a_1|}^{\varepsilon_1} \cdots x_{|a_m|}^{\varepsilon} \in G \text{ ただし } \varepsilon_i = \frac{a_i}{|a_i|}$$

と定義するとき, $\pi : F(X) \to G$ は群の全射準同型である. $F(X)$ の元で長さが m であるもの全体を $F(X)_m$ で表す. $F(X)_1 = \{[-n], \cdots, [-1], [1], \cdots, [n]\}$ である. 著者の ca.js では自由群のクラスを実装しており, Minkwitz 分解アルゴリズムにおいて利用する. $X = \{x_1, \cdots, x_n\}$ なので, $|F(X)_m| = 2n(2n-1)^{m-1}$ となり, その列挙は著者の実装 [1] において試すことができる.

次に具体的な実装について考察する. 著者は予め BSGS が求まっている状態で, 関数 minkwitz を引数

$$\text{生成系：} X, \quad \text{軌道の属：} \{\Delta^{(i)}\}_{i=0}^{k-1}$$

に対し, $w_{i,j} \in F(X)$ $(i = 0, \cdots, l-1,\ j = 0, \cdots, r_i - 1)$ で,

$$\pi(w_{i,j}) \in G^{(i)},\ {\beta_i}^{\pi(w_{i,j})} = \beta_{i,j} \tag{1}$$

を満たすようなものを戻り値となるよう実装した. まず, 軌道の集合は配列 $\text{orbits}[i][j] = \beta_{i,j}$ とする. $\text{orbits}[i][0] = \beta_i$ は base である. 次に, 多重配列 $\text{box}[i][j] = [\,]$ $(i = 0, \cdots, l-1,\ j = 0, \cdots, r_i - 1)$ を定義し, 条件 (1) を満たすような $w_{i,j}$ を構成できたら $\text{box}[i][j] = w_{i,j}$ とし, これを全ての i, j について完成させ, なおかつ各語 $\text{box}[i][j]$ の長さを可能な限り小さくすることが目的である. $\text{box}[i][j]$ に語 $w_{i,j}$ を格納すると同時に, 多重配列 $\text{sgs}[i][j]$ に置換 $\pi(w_{i,j})$ を格納していく. 素朴な考えとして $m = 1$ から順に $w \in F(X)_m$ が条件 (1) を満たすかどうか判別していき, box を完成させることを思いつくが, $|F(X)_m| = 2n(2n-1)^{m-1}$ なので m が大きくなればなるほど, 非効率であることが分かる. 著者の実装では, 最初に $x_a \in X$ の位数が $2m$ のとき, $a, \cdots, a^m, -a, \cdots, (-a)^{m-1}$ を, $x_a \in X$ の位数が $2m+1$ のとき, $a, \cdots, a^m, -a, \cdots, (-a)^m$ について条件 (1) が成り立てば $\text{box}[i][j]$ に格納する. 次に $m \geq 2$ について, $F(X)_m$ の各元 w に対し, $\pi(w) \in G^{(i)}$ かつ $\pi(w) \notin G^{i+1}$ のとき, ${\beta_i}^{\pi(w)} = \beta_{i,j}$ ならば,

- $\text{box}[i][j] = [\,]$ または, $l(\text{box}[i][j]) > l(w)$ のとき, $\text{box}[i][j] = w$
- $0 < l(\text{box}[i][j]) < l(w)$ のとき, $w_0 := w\ \text{box}[i][j]^{-1}$ とおく.

これを i に関する for ループで繰り返す関数を add_word と定義する. これを m に関する for ループで繰り返し, box が全て埋まり次第終了する. 語の長さを短くする工夫として, add_word 中に reduce_row$_1$ength という関数を挿入する. $\mathrm{sgs}[i][j][\mathrm{orbits}[i][0]] = \mathrm{orbits}[i][j]$ であるが, $\mathrm{sgs}[i][k][\mathrm{orbits}[i][j]] = \mathrm{orbits}[i][0]$ となるような k が存在すれば, $l(\mathrm{box}[i][j]) > l(\mathrm{box}[i][k])$ ならば, $\mathrm{box}[i][j]$ に $\mathrm{box}[i][k]^{-1}$ を格納し, $\mathrm{sgs}[i][j]$ に $\mathrm{sgs}[i][k]^{-1}$ を格納する. 以下に関数 minkwitz の JavaScript コードを ca.js [1] から抜粋して紹介する.

```
var minkwitz = function(v,orbs){//v は生成系, orbs は軌道
  var n = v.length, m = orbs.length;
  var word = free.first(n);//長さ 1の自由群の元全体
  var box = [], sgs = [];
  for(var i=0;i<m;i++){
    box[i] = [], sgs[i] = [];
    var ol = orbs[i].length;
    for(var j=0;j<ol;j++){
      box[i][j] = [], sgs[i][j] = [];
    }
  }
  var word_to_perm = function(w){//自由群の元w に生成系の積を返す
    var l = w.length;
    for(var i=0;i<l;i++){
      var a = w[i];
      a>0 ? var t = v[a-1] : var t = inv(v[-a-1]);
      i==0 ? var y = t : y = composition(y,t);
    }
    return y;
  }

  function add_power(){//生成元のべきに対応する語全部を返す
    var w = [], s=0;
    for(var i=0;i<n;i++){
      var d = order(v[i]), sg = d%2, hd = parseInt((d-sg)/2), r=i+1;
      for(var k=1;k<=hd-(1-sg);k++){
        var ua = free.power([-r],k), ub = free.power([r],k);
        w.push(ua), w.push(ub);
      }
      if(sg==0) w.push(free.power([r],hd));
    }
    return w;
  }
  var add_word = function(wx){
    var x = word_to_perm(wx);
    for(var j=0;j<m;j++){
      var orb = orbs[j], a = orb[0], d = x[a], e = inv(x)[a];
      if(d != a){
        reduce_row_length(j);
        var k = orb.indexOf(d), l = orb.indexOf(e);
        if(box[j][k].length == 0 || wx.length < box[j][k].length){
          box[j][k] = wx, sgs[j][k] = x, break;
        }else{
          var sigi = false;
```

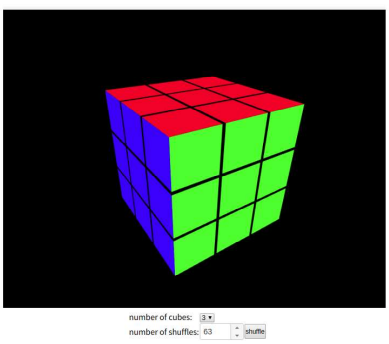
```
45            if(box[j][l].length != 0 && wx.length > box[j][l].length){
46              var wxi = free.composition(box[j][l],wx), xi = composition(sgs[j][l],x);
47              sigi = true;
48            }
49            wx = free.composition(wx,free.inv(box[j][k]));
50            x = composition(x,inv(sgs[j][k]));
51            if(sigi && wxi.length < wx.length){
52              wx = wxi.slice(), x = xi.slice();
53            }
54          }
55        }
56      }
57    }
58    var reduce_row_length = function(j){
59      var orb = orbs[j], a = orb[0];
60      for(var k=1;k<orb.length;k++){
61        var wx = box[j][k];
62        if(wx.length != 0){
63          var b = orb[k], x = sgs[j][k], wxl = wx.length, c = inv(x)[a];
64          var o = orb.indexOf(c), wz = box[j][o], z = sgs[j][o], wzl = wz.length;
65          if(wzl != 0 && wzl > wxl){
66            box[j][o] = free.inv(wx), sgs[j][o] = inv(x);
67          }
68        }
69      }
70    }
71    var pw = add_power(), pl = pw.length;
72    for(var i=0;i<pl;i++){
73      add_word(pw[i]);
74    }
75    var count=0;
76    while(true){
77      word = free.add_elements(word);//自由群の長さを 1つ増やす
78      //word[i] は長さ i+1 の自由群の元全体
79      var wl = word.length;
80      for(var i=1;i<wl;i++){
81        var wd = word[i], wdl = wd.length;
82        for(var s=0;s<wdl;s++){
83          var wx = wd[s];
84          add_word(wx);
85        }
86      }
87      var sig = true;
88      for(var i=0;i<m;i++){
89        var bo = box[i], ol = orbs[i].length;
90        for(var j=1;j<ol;j++){
91          if(bo[j].length == 0) sig = false, break;
92        }
93        if(!sig) break;
94      }
```

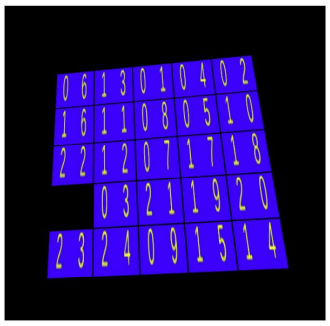
```
95        if(sig) break;
96        count++;
97      }
98      return {
99        box:box, sgs:sgs, count:count
100     };
101   }
```

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 [5] の 184 や Minkwitz の 144 に比べてかなり劣る.

3 Three.js での実装

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. 和島茂氏のサイト「バーチャル 3D パズル」[11] 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 [8] 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る回転やスライドを実装して動かしたり, 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するなどを行った. 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_24puzzle.html
https://noblegarden-math.jp/math/three_rubiks.html

4 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.

- Minkwitz 分解アルゴリズムは応用上重要で有るにも関わらず, 英文であってもその効率化に関する文献が非常に少ない. より洗練されたアルゴリズムを実装したい.

- 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.

- coset enumeration は計算群論の教科書では定番で, 基本的であるにも関わらず, 手計算で実行するには非常に時間がかかる上, 計算ミスも起こしやすい. 著者はこれを JavaScript で実装し, cite [1] にて公開している. 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 [7] 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.

参 考 文 献

[1] ca.js, https://noblegarden-math.jp/math/notes/computer_algebra/01/

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

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

[4] 花木章秀, 群論と対称性, http://math.shinshu-u.ac.jp/~hanaki/edu/symmetry/perm.pdf

[5] Mathematics_and_Such, https://mathstrek.blog/2018/06/21/solving-permutation-based-puzzles/

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

[7] Scott H. Murray, The Schreier-Sims algorithm, 2003, http://www.maths.usyd.edu.au/u/murray/research/essay.pdf

[8] 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

[9] rubik's cube solever, https://noblegarden-math.jp/math/three_rubiks.html

[10] 24 puzzle solver, https://noblegarden-math.jp/math/three_24puzzle.html

[11] 和島茂, バーチャル 3D パズル, http://aows.jp/jspuzzle/