The Schreier-Sims Algorithm – A (very) Naive Implementation

Stefan Hoffmann

October 2021

1 Introduction

The Schreier-Sims algorithm [3] is based on Schreier's lemma (see Lemma 1.1) and provides a convenient data structure to represent a permutation group that yields (polynomial time) algorithms to compute the order of a given group or to perform membership queries, i.e., asking if a given permutation is in the permutation group. Additionally, it allows for an easy enumeration of the elements of the permutation group.

This document accompanies a very naive implementation that only serves educational purposes and does not use any (necessary for polynomial time) improvements like reducing the number of generators, see for example [2].

A *permutation group on a set* Ω is a subgroup of the set of all permutations, i.e., a bijective mapping, on Ω.

If G is a permutation group, $\alpha \in \Omega$ and $g \in G$, we write α^g (which is a common notation in permutation group theory) for the image of α under g.

Let G be a group with subgroup $H \leq G$. A subset $R \subseteq G$ is called a *right transversal (for H in G)* if for every $g \in G$ there exists precisely one $r \in R$ such that $gr^{-1} \in H$. This is equivalent to the fact that R contains precisely one element from each right coset of H in G.

First, let use state (and prove) Schreier's lemma.

Lemma 1.1 (Schreier's lemma)**.** *Let G be a group generated by $S \subseteq G$. Let $H \leq G$ be a subgroup and R a right transversal for H in G with $1 \in R$. For every $g \in G$ denote by $\overline{g} \in R$ the unique element from R with $g\overline{g}^{-1} \in H$. Then H is generated by the set*

$$T = \{rs(\overline{rs})^{-1} \mid r \in R, s \in S\}.$$

Proof. Clearly $T \subseteq H$. Let $r \in R$ and $s \in S$ and set $g = rs(\overline{rs})^{-1} \in H$. Then $g^{-1} = \overline{rs}s^{-1}r^{-1} \in H$, which implies $\overline{\overline{rs}s^{-1}} = r$, as $r \in R$. So $g^{-1} = r's^{-1}\overline{r's^{-1}}^{-1}$ with $r' = \overline{rs} \in R$ and we find $T^{-1} \subseteq \{rs^{-1}(\overline{rs^{-1}})^{-1} \mid r \in R, s \in S\}$. Conversely, if $g = rs^{-1}(\overline{rs^{-1}})^{-1}$. Then, $g^{-1} = \overline{rs^{-1}}sr^{-1} \in H$, which implies $\overline{\overline{rs^{-1}}s} = r$. Set $r' = \overline{rs^{-1}} \in R$. So we can write $g = \overline{r's}(r's)^{-1} \in T^{-1}$. So, in summary, we have shown $T^{-1} = \{rs^{-1}(\overline{rs^{-1}})^{-1} \mid r \in R, s \in S\}$.

Now, let $h \in H$. Then $h = s_1 s_2 \cdots s_n$ with $s_i \in S \cup S^{-1}$. We have $h = r_1 s_1 s_2 \cdots s_n$ with $r_1 = 1 \in R$. Now, suppose $h = u_1 u_2 \cdots u_j r_j s_j s_{j+1} \cdots s_n$ for $j \in \{1, 2, \ldots, n\}$ with $r_j \in R$ and $u_1, u_2, \ldots, u_j \in T \cup T^{-1}$. Then

$$\begin{aligned} h &= u_1 u_2 \cdots u_j r_j s_j s_{j+1} \cdots s_n \\ &= u_1 u_2 \cdots u_j r_j s_j \overline{r_j s_j}^{-1} \overline{r_j s_j} s_{j+1} \cdots s_n \\ &= u_1 u_2 \cdots u_j u_{j+1} r_{j+1} s_{j+1} \cdots s_n \end{aligned}$$

with $u_{j+1} = \overline{r_j s_j}^{-1} \overline{r_j s_j} \in T \cup T^{-1}$ and $r_{j+1} = \overline{r_j s_j}$. So, inductively, we have

$$h = u_1 u_2 \cdots u_n r_n$$

with $u_i \in T \cup T^{-1}$ and $r_n \in R$. Finally, as $u_1 u_2 \cdots u_n \in H$ we can deduce $r_n = 1$ and the statement is shown. □

2 The Schreier-Sims Algorithm

For $\alpha \in \Omega$, the *point stabilizer (of α in G)* is $G_\alpha = \{g \in G \mid \alpha^g = \alpha\}$.

Suppose $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. Set $G_0 = G$ and

$$G_i = G_{\alpha_1} \cap G_{\alpha_2} \cap \ldots \cap G_{\alpha_n}$$

for $i \in \{1, 2, \ldots, n\}$. Then $G_n = \{1\}$. Such a sequence of subgroups is called a *stabilizer chain (for G)*.

As $h \in G_\alpha g$ iff $\alpha^{hg^{-1}} \in G_\alpha$, we find:

$$G_\alpha g = \{h \in G \mid \alpha^h = \alpha^g\}. \tag{1}$$

In fact, Equation (1) implies $\alpha^h = \alpha^g$ iff h and g are contained in the same right coset for G_α and so the map $g \mapsto \alpha^g$ for $g \in G$ is a bijection between the orbit of α and the set of right coset of G_α. This is the statement of the *orbit-stabilizer theorem*.

A stabilizer chain together with a right transversals R_i for G_i in G_{i-1} is a convenient way to represent a given permutation group G. By the orbit-stabilizer theorem, we have $|G_{i-1} : G_i| = |R_i|$. By storing the elements from R_i in concordance with this correspondence given by Equation (1) and the orbit-stabilizer theorem, the following computational problems can then be solved.

1. Computing the order of G: By Lagrange's theorem, we have
$$|G| = \prod_{i=1}^{n} |G_{i-1} : G_i| = \prod_{i=1}^{n} |R_i|.$$
2. Membership testing, i.e. given a permutation g on Ω, testing if $g \in G$: We have $g \in G_{i-1}$ iff $gr^{-1} \in G_i$ for some $r \in R_i$. Hence $g \in G$ iff there exist $r_1 \in R_1, r_2 \in R_2, \ldots, r_n \in R_n$ such that
$$g r_1^{-1} r_2^{-1} \cdots r_n^{-1} = 1.$$
In total, we have to perform at most n^2 such tests here.

3. Enumerating the elements of G: By the arguments given for the membership testing algorithm, we have

$$G = \{r_1 r_2 \cdots r_n \mid r_1 \in R_1, r_2 \in R_2, \ldots, r_n \in R_n\}.$$

3 Computation of the Stabilizer Chain

Here, we assume the permutation group is given as a list of generators and we will outline a procedure to compute a stabilizer chain together with a right transversal as assumed in the description of the algorithm in Section 2.

But first, let us reformulate Schreier's lemma for point stabilizers with the help of Equation (1).

Lemma 3.1. *Let G be a transitive permutation group on $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ generated by $S \subseteq G$. Let $\alpha \in \Omega$ and choose $g_i \in G$, $i \in \{1, 2, \ldots, n\}$, such that $\alpha^{g_i} = \alpha_i$. Then G_α is generated by the set*

$$\{g_i s g_j^{-1} \mid i, j \in \{1, 2, \ldots, n\}, s \in S : \alpha_i^s = \alpha_j\}.$$

Proof. By Equation (1), the elements $g_1, g_2, \ldots, g_n$ are a right transversal and, with the notation from Lemma 1.1 and Equation (1),

$$\overline{g_i s} = g_j$$

for the unique $j \in \{1, 2, \ldots, n\}$ with $\alpha_i^s = \alpha^{g_i s} = \alpha^{g_j} = \alpha$. □

So, in particular, a point stabilizer has a generating set of size $|S||\Omega|$.

Recall $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and the definitions of the G_i from Section 2. Now, given G in terms of the generators S, we can compute a right transversal for G_1 by computing the image of α_1 for each generator. Then, we continue this recursively, but stop as soon as we find an image that was already considered in the process. Throughout this process, we keep track of an element from G for each recursive call by multiplying out the generators we selected in each call, starting with the identity element. Furthermore, as soon as a new point is discovered, the tracked element along the recursive calls is one element from the right transversal. In particular, for α_1, we put the identity element into the right transversal. Additionally, we also associate the so computed element with the point, for example, by storing them both together in a list. This whole procedure could be made to run into $|\Omega| + |S||\Omega|$ many steps. In fact, it could be viewed as a depth-first search on a labelled graph with vertex set Ω and edge set $\{(\alpha, s, \beta) \mid \alpha, \beta \in \Omega, s \in S\}$.

Now, by the above procedure, we can compute a right transversal for G_1 and store the image of the point α_1 for each element from this transversal.

Then, by applying Lemma 3.1, we can compute a set of generators for G_1 with these data.

Furthermore, we can continue this process for G_2, considered as a subgroup of G_1 given by generators, and continuing this way for G_3, G_4 up to G_1.

Note that in each step, the number of generators increases by $n = |\Omega|$ times the number of generators from the previous step. Hence, in total, this counts up to n^n times the number of generators of G after the n-th step. However, this could be made to run more efficient by using Jerrum's filter [2], which runs in polynomial time and reduces the number of generators to $n - 1$ in each step.

Furthermore, as desribed in [1], Schreier vectors can be used for storing both a right transversal and the resulting points from above (which are the orbits of the points α_i for the groups G_{i-1}). In the implementation presented here, we do not use Schreier vectors, but associate them by using an array.

4 The Program

I tried to keep the program simple and more or less self-explanatory. It consists of four classes:

- `permutation`,
- `orbit_rep`,
- `point_stabilizer` and
- `stabilizer_chain`.

A right transversal is stored together with an orbit (as outlined in Section 3). More specifically, the variable `m_rep` is a double pointer, which is meant to hold an array of pointers to permutations, which encompass the right transversal, and the index in this array represents the point that is the image of another point (stored in `m_point`) under the stored element of the right transversal. All indices that correspond to point not in the orbit are set to `NULL`. This is a naive way to implement the storage of a right transversal and a corresponding orbit, see the discussion in [1].

I do not claim that this is an elegant or efficient implementation, and it also does not follow the object oriented paradigm in full rigor (for example, a member variable, which is a pointer to heap memory, is public in `orbit_rep`). I wrote it only for educational purposes.

A sample run with the dihedral group on twelve points.

```
Please enter the number of points, n = 12
Generators are entered by giving a list of integers, where the
value at position i determines the image of i.
Example, enter '1 2 0' for the permutation that maps 0 to 1, 1 to 2
and 2 to 0 on three points.
Note that the points are numbered from 0 to n - 1.

Please enter generator number 1 by a list of 12 integers.
Enter a negative value if you do not want to input further generators.
Input: 1 2 3 4 5 6 7 8 9 10 11 0
```

```
Please enter generator number 2 by a list of 12 integers.
Enter a negative value if you do not want to input further generators.
Input: 0 11 10 9 8 7 6 5 4 3 2 1

Please enter generator number 3 by a list of 12 integers.
Enter a negative value if you do not want to input further generators.
Input: -1
Options:
        (1) Compute order of the group.
        (2) Enumerate the elements.
        (3) Do a membership query.
        (4) Exit.

1
Order of group: 24
Options:
        (1) Compute order of the group.
        (2) Enumerate the elements.
        (3) Do a membership query.
        (4) Exit.

2
Enumerate group elements:
1:       0  1  2  3  4  5  6  7  8  9 10 11
2:       0 11 10  9  8  7  6  5  4  3  2  1
3:      11  0  1  2  3  4  5  6  7  8  9 10
4:       1  0 11 10  9  8  7  6  5  4  3  2
5:      10 11  0  1  2  3  4  5  6  7  8  9
6:       2  1  0 11 10  9  8  7  6  5  4  3
7:       9 10 11  0  1  2  3  4  5  6  7  8
8:       3  2  1  0 11 10  9  8  7  6  5  4
9:       8  9 10 11  0  1  2  3  4  5  6  7
10:      4  3  2  1  0 11 10  9  8  7  6  5
11:      7  8  9 10 11  0  1  2  3  4  5  6
12:      5  4  3  2  1  0 11 10  9  8  7  6
13:      6  7  8  9 10 11  0  1  2  3  4  5
14:      6  5  4  3  2  1  0 11 10  9  8  7
15:      5  6  7  8  9 10 11  0  1  2  3  4
16:      7  6  5  4  3  2  1  0 11 10  9  8
17:      4  5  6  7  8  9 10 11  0  1  2  3
18:      8  7  6  5  4  3  2  1  0 11 10  9
19:      3  4  5  6  7  8  9 10 11  0  1  2
20:      9  8  7  6  5  4  3  2  1  0 11 10
21:      2  3  4  5  6  7  8  9 10 11  0  1
22:     10  9  8  7  6  5  4  3  2  1  0 11
```

```
23:      1  2  3  4  5  6  7  8  9 10 11  0
24:     11 10  9  8  7  6  5  4  3  2  1  0
Options:
        (1) Compute order of the group.
        (2) Enumerate the elements.
        (3) Do a membership query.
        (4) Exit.

3
Enter permutation on 12 points: 1 0 2 3 4 5 6 7 8 9 10 11
Result of membership query: 0
```

References

[1] Alexander Hulpke. Notes on Computational Group Theory. `https://www.math.colostate.edu/~hulpke/CGT/cgtnotes.pdf`, 2010. Lecture Notes.

[2] Mark Jerrum. A compact representation for permutation groups. In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 126–133. IEEE Computer Society, 1982.

[3] Charles Coffin Sims. Computational methods in the study of permutation groups. In JOHN LEECH, editor, *Computational Problems in Abstract Algebra*, pages 169–183. Pergamon, Oxford, 1970. Proceedings of a Conference Held at Oxford Under the Auspices of the Science Research Council Atlas Computer Laboratory, 29th August to 2nd September 1967.