Factorization in finite Groups

Alexander Hulpke

Department of Mathematics

Colorado State University

Fort Collins, CO, 80523

hulpke@math.colostate.edu

http://www.math.colostate.edu/~hulpke

Factorization

Given a finite (permutation) group $G = \langle X \rangle$, express $g \in G$ as a word in $X \cup X^{-1}$.

Applications

- Puzzles
- Homomorphisms
- Finding presentations

Desiderata and Options

- fast
- find shortest word
- might be willing to choose particular (nice) generators

The Orbit Algorithm

For $G = \langle X \rangle$, acting on a set Ω find the orbit and stabilizer of $\omega \in \Omega$.

For example: Orbit of $1 \in G$ under right multiplication.

$L := [\omega]; \quad p := 1$
while $p \leq |L|$ **do**
  $\alpha := L[p]$
  for $g \in X \cup X^{-1}$ **do**
    $\beta := \alpha^g$
    if $\beta \notin L$ **then**
      Add β to L
    fi;
  od;
  $p := p + 1$
od;

Transversals

As a side effect we get a *transversal* (set of coset representatives) for $\mathrm{Stab}_G(\omega)$ in G:

For each $\eta \in \omega^G$, we have an element $r_\eta \in G$ such that $\omega^{r_\eta} = \eta$.

This element is obtained as product of generators from X. One can remember expression of this element as word.

These words are *shortest* words for elements mapping ω to η.

Schreier's theorem

$$\mathrm{Stab}_G(\omega) = \langle r_\eta x r_{\eta^x}^{-1} \mid \eta \in \omega^G, x \in X \rangle$$

These "Schreier generators" also can be expressed as words in the generators.

Shortest words

If we *must* have guaranteed shortest words the only known algorithm is to calculate the orbit of 1 under right multiplication.

The transversal then yields factorizations.

Memory

Assume we can enumerate group with little memory.

Then store in list at position i:

the (number of the) generator by which group element i was reached

or the length of shortest word to reach element i. (To find word, multiply with generator inverses to find element of shorter length.)

Further reduction

If we do not store full length, but length modulo 3 (COOPERMAN).

(Backtrack if there are several preimage candidates.)

If we only want the diameter of the Cayley graph, single bit would be sufficient.

Implementation fall'03 by N. Rohrbacher.

Feasibility

This approach is clearly limited by available memory.

Works for corners or Rubik's cube ($2 \times 2 \times 2$, order 88179840 – 21 MB) but not for action on edges (order 980995276800 – 228 GB)

Heuristic: Store full element indices and run until memory is filled. Then form products of two words.

Approximations

Take subset $Y \subset X$ and consider $U = \langle Y \rangle$ such that we can enumerate all elements of U and a transversal (action on the cosets of U) for U in G.

Memory requirement is $|U| + [G : U]$. Potentially one can iterate through a chain of subgroups.

Rubik's cube: (THISTLETHWAITE, REID) Let $Y = \{\text{top}, \text{bottom}, \text{right}^2, \text{left}^2, \text{front}^2, \text{back}^2\}$.

Then $|U| = 19508428800$ and $[G : U] = 2217093120$.

Cosets of U can be enumerated with length 12. Diameter of U is 18 (or 30 if we count squares of length 2).

Thus every element of the cube can be reached by 42 rotations.

(Best known lower bound is 24.)

Stabilizer chains

Fundamental idea (SIMS, 1967):

Given a finite permutation group G on domain Ω

- Pick a point $\omega \in \Omega$.
- Using the orbit algorithm, calculate ω^G, transversal, generators for $\mathrm{Stab}_G(\omega)$.
- Iterate recursively for $\mathrm{Stab}_G(\omega)$.

The list of points ω_i chosen for the iterative stabilizers is called a *base*, the union of the stabilizer generators is called a set of *strong generators*.

The resulting data structure (base, strong generators, transversals) is called a *stabilizer chain*

Example:

$G = S_3 = \langle (1,2,3), (1,2) \rangle$. $\omega_1 = 1$

η	1	2	3
r_η	$()$	$(1,2,3)$	$(1,2,3)^2$

Some Schreier generators:

$$\begin{aligned} r_1 \cdot (1,2) \cdot r_2^{-1} &= (2,3) \\ r_3 \cdot (1,2) \cdot r_3^{-1} &= (2,3) \\ r_3 \cdot (1,2,3) \cdot r_1^{-1} &= () \end{aligned}$$

$\omega_2 = 2$, then:

η	2	3
r_η	$()$	$(2,3)$

All Schreier generators are trivial now.

Consequences:

- Group order (product of transversal lengths).
- Element test (sifting): if $\omega^g = \eta$, then $g/r_\eta \in \mathrm{Stab}_G(\omega)$, iterate.
- Yields expression as words, but unhandily large ($10^5 - 10^6$ for Rubik's cube).
- Enumeration of group (consider transversal indices as multi-adic representation).
- "Perfect" random elements.
- Systematic way to run through elements of group (backtrack).

Practicalities:

To save memory, often a *factorized transversal* is stored:

Instead of storing different transversal elements we just store a *pointer* to the last generator in the expression as word.

Transversal elements then can be obtained dividing off apropriate generators.

Generator Numbers

Large number of Schreier generators is unhandy:

Algorithm does not build list of schreier generators but sifts each newly found generator through the stabilizer chain below. If it does not sift trivially, it is added as generator. $\mathcal{O}\left(n^6\right)$.

Randomization

The Random Schreier Sims algorithm uses only some carefully formed (random) Schreier generators. $\mathcal{O}\left(n \log n + \epsilon\right)$.

Problem: Chain is built "too small" (resulting group order).

Verification by either:

- Known group order (`SetSize`)
- Checking properties for certain action(s) (GAP default)
- Computing a composition series, constructive recognition of composition factors, verification by finding presentations for composition factors. Still in the same $\sim \mathcal{O}\left(n \log n\right)$ complexity *if short presentations are known*.

  (Known theoretically for all simple groups except Ree.)

Presentations

Suppose that $\mathrm{Stab}_G(\omega)$ is trivial.

Then the Schreier generators form a set of defining relators for G.

Similarly, if Y is a set of words in generators of G and $U = \langle Y \rangle$, one can express all Schreier generators for U in G as words in Y.

As relators for a finitely presented group, these expressions ensure a subgroup of index $[G : U]$.

One can combine this (CANNON) to compute a presentation for G in the given generators.

Homomorphisms

Suppose $\varphi\colon G \to H$ is homomorphism, given by images of generating set X. Instead of decomposing into words, simulateneously evaluate all transversal elements and schreier generators in images $\varphi(X)$.

To calculate $\varphi(g)$, sift g through stabilizer chain, and multiply together the respective transversal images.

Presentation evaluation tests whether map on generators yields homomorphism.

Presentation test for "inverse mapping" computes kernel.

Words

The same homomorphism approach works for preimages under homomorphism: instead of $X \to Y$ map $Y \to X$ (not a homomorphism).

Map $F \to G$ from free group then gives words for generators. However we get the same problem as above with word lengths.

The reason for this is that the (iterated) Schreier generators very quickly grow in length.

MINKWITZ' idea

Instead of systematically building transversals and Schreier generators

Produce short words in the generators, and insert them at the appropriate place as transversal elements.

Then use the ordinary decomposition algorithm. As the transversal elements are words in generators (and not words in Schreier generators) they produce shorter words overall.

In principle this increases the number of subgroup generators. However these generators are only used within the stabilizer chain.

Base Choice

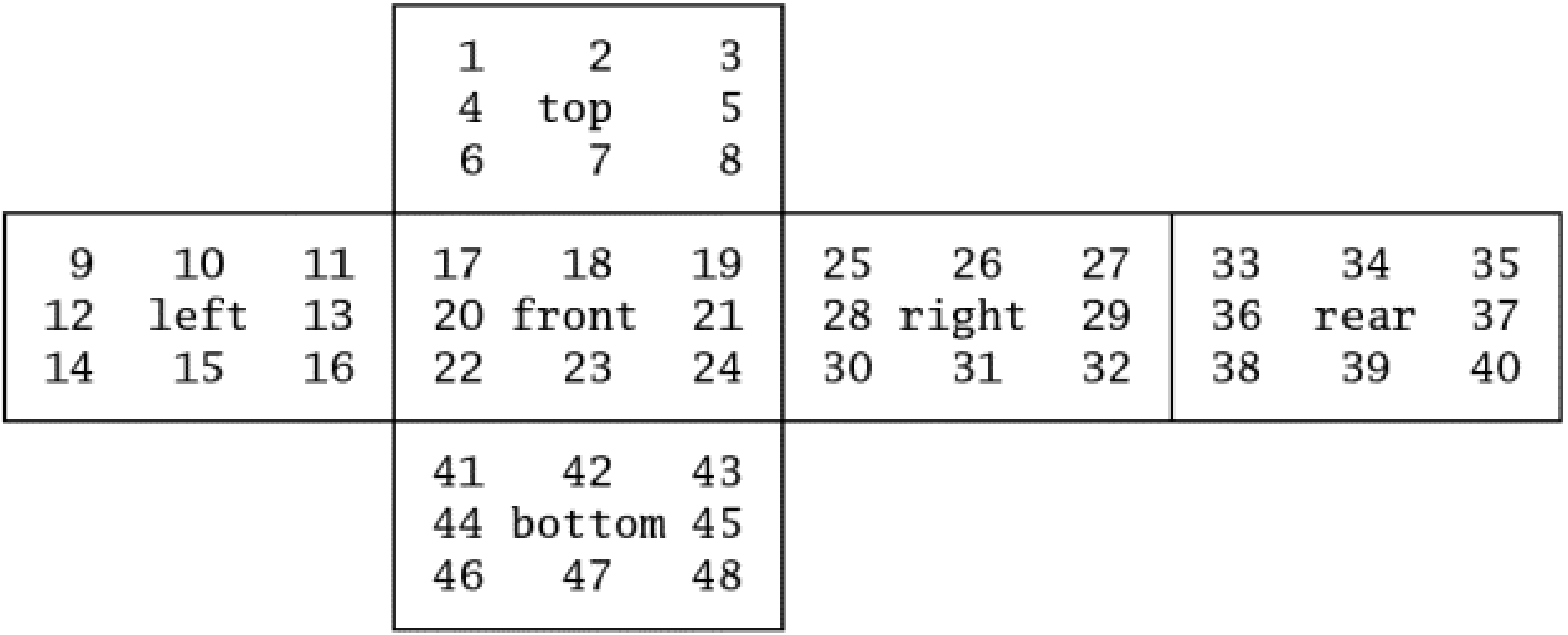
Furthermore, we have a freedom in choosing a base:

Ensure that as many as short words as possible lie in the stabilizers.

In the Rubik's cube example this reduces the length of words for random elements to about 100.

It is unlikely that an approach that does not rely on extensive enumeration would produce much shorter words.

Example




```
gap> cube:=Group((1,3,8,6)(2,5,7,4)(9,48,15,12)(10,47,16,13)(11,46,17,14),
>   (1,12,33,41)(4,20,36,44)(6,27,38,46)(9,11,26,24)(10,19,25,18),
>   (6,15,35,26)(7,22,34,19)(8,30,33,11)(12,14,29,27)(13,21,28,20),
>   (3,43,35,14)(5,45,37,21)(8,48,40,29)(15,17,32,30)(16,23,31,22),
>   (24,27,30,43)(25,28,31,42)(26,29,32,41)(33,35,40,38)(34,37,39,36),
>   (1,24,40,17)(2,18,39,23)(3,9,38,32)(41,43,48,46)(42,45,47,44));
<permutation group with 6 generators>
```

```
gap> Size(cube);
43252003274489856000
```

```
gap> f:=FreeGroup("t","l","f","r","b","z");;
gap> hom:=GroupHomomorphismByImages(f,cube,
> GeneratorsOfGroup(f), GeneratorsOfGroup(cube));
[ t, l, f, r, b, z ] -> [ (1,3,8,6)(2,5,7,4)...]
gap> r:=Random(cube);
(1,29,6,38,15,9,30,12,41,14,46,35,11,24,8)(2,47)..
gap> w:=PreImagesRepresentative(hom,r);
```

$$Lt^2ltfrtRFltlFLf^2TFLtFLTltf^2tltLFlTLfTFT^2ltfTFLTlftF$$

$$TFLfLFLfltZTzFLfRF^2rtlRFR^3BLbLtFtlBZbZR^2zTrFTB^2$$

```
gap> Length(w);
100
```

If we are willing to change generators

Much more efficient way to get reasonably short words in particular generators:

Determine a composition series $G = G_0 > G_1 > \cdots > G_k = \langle 1 \rangle$ of G.

For each factor G_i/G_{i+1} choose representatives for generators: 1 if cyclic, 2 if noncyclic.

Then first decompose in top factor group. Divide off evaluated word in original generators. Result will lie in next subgroup in series. Repeat

Ex: If G is solvable all factors are cyclic, the corresponding set of generators is called a *P*oly*C*yclic*G*enerating*S*ystem.

Each element can be described by *exponent vector* with enties $0 \leq x_i < p_i$.

Constructive Recognition

The most efficient way to treat nonsolvable factors is by constructive recognition:

Find “nice" generators (for which we have for example a short presentation stored).

Reconstruction of “abstract" geometry gives decomposition algorithm.

Much theoretical word over the last decade, very little available implementations so far.