Solving Permutation-Based Puzzles

Posted on June 21, 2018 by limsup

Introduction

In the previous article, we described the Schreier-Sims algorithm. Given a small subset $A \subseteq S_n$ which generates the permutation group *G*, the algorithm constructs a sequence $k_1, k_2, \ldots, k_m \in [n] = \{1, 2, \ldots, n\}$ such that for:

$$G = G^{(0)} \geq G^{(1)} \geq \ldots \geq G^{(m)} = 1,$$
$$G^{(i)} := \{g \in G : g(k_1) = k_1, g(k_2) = k_2 \ldots, g(k_m) = k_m\}.$$

we have a small generating set $A^{(i)}$ for each $G^{(i)}$. Specifically, via the Sims filter, we can restrict $|A^{(i)}| \leq \frac{n(n-1)}{2}$ for each *i*.

In this article, we will answer the following question.

> ***Factorization Problem***.
>
> *How do we represent an arbitrary representation $g \in G$ as a product of elements of A and their inverses?*

If we can solve this problem in general, we would be able to solve a rather large class of puzzles based on permutations, e.g. Rubik's cube and the Topspin puzzle. First, let's look at the straightforward approach from the Schreier-Sims method.

Attempt

Recall that the Schreier-Sims method starts with $A^{(0)} := A$. At each step it picks a base element $k_i \in [n]$, computes a generating set for $G^{(i)}$ from $A^{(i-1)}$ then pares down this set with the Sims filter so that $|A^{(i)}| \leq \frac{n(n-1)}{2}$. Thus one can, in theory, keep track of the elements of $A^{(i)}$ by expressing them as words in *A*.

The problem with this approach is that since $A^{(i)}$ is obtained from $A^{(i-1)}$, the length of words for $A^{(i)}$ would be a constant factor of those for $A^{(i-1)}$. Thus by the time we reach $A^{(m)}$, their lengths would be exponential in *m*. 😟

Minkwitz's Algorithm

As far as we know, the first paper to solve the factorization problem is by T. Minkwitz in "*An Algorithm for Solving the Factorization Problem in Permutation Groups*" (1998). The idea is elegant. First, we replace the generating sets $A^{(i)}$ with the following tables.

> ***Main Goal***.
>
> *For each $0 \leq i \leq m-1$, let O_i be the orbit $G^{(i)} \cdot k_{i+1}$. We wish to obtain a set $B_i \subseteq G^{(i)}$, indexed by $x \in O_i$, such that*

- $B_i(x) \in G^{(i)}$ *takes the base element* $k_{i+1} \mapsto x$.

In other words, the element $g = B_i(x)$ is any element of *G* satisfying:

$$g(k_1) = k_1, g(k_2) = k_2, \qquad , g(k_i) = k_i, g(k_{i+1}) = x.$$

Minkwitz's method replaces the sequence of sets $A_0, \ldots, A_{m-1}$ with $B_0, \ldots, B_{m-1}$; furthermore, the elements of the latter sets are stored as *words* in $A \cup A^{-1}$ instead of mere permutations.

To begin, we initialize $B_i(k_{i+1})$ to be the empty word for *i* = 0, ..., *m*-1.

Next we run through words in $A \cup A^{-1}$, starting from those of length 1, then those of length 2, etc. For each word *w*, compute the corresponding element $g \in G$ and do the following:

1. Start with *i* = 0.
2. Let $x = g(k_{i+1}) \in O_i$. Do we have an entry for $B_i(x)$?
   - If not, we let $B_i(x)$ be the word *w* and quit.
   - If yes, and *w* is shorter than the current word in $B_i(x)$, we replace it with *w* and quit.
   - Otherwise, let $w' = B_i(x)$. Replace *w* with $w'^{-1}w$ and increment *i* then repeat step 2.

Example

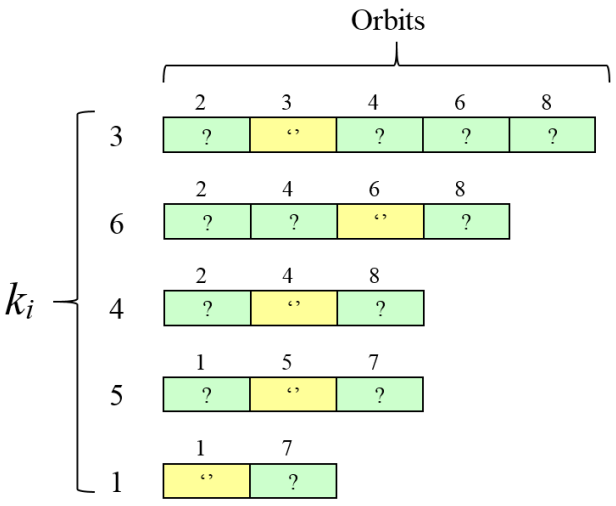
Let us take the example from the previous article, with $G \leq S_8$ generated by:

$$a = (1, 5, 7)(2, 6, 8), \qquad b = (1, 5)(3, 4, 8, 2).$$

Applying the Schreier-Sims algorithm, we obtain the following base and orbits:

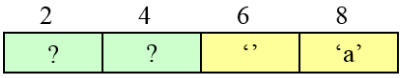
$$\begin{aligned} k_1 = 3 &\implies \text{orbit } O_0 = \{2, 3, 4, 6, 8\}, \\ k_2 = 6 &\implies \text{orbit } O_1 = \{2, 4, 6, 8\}, \\ k_3 = 4 &\implies \text{orbit } O_2 = \{2, 4, 8\}, \\ k_4 = 5 &\implies \text{orbit } O_3 = \{1, 5, 7\}, \\ k_5 = 1 &\implies \text{orbit } O_4 = \{1, 7\}. \end{aligned}$$

From this, we deduce that the order of *G* is 5 × 4 × 3 × 3 × 2 = 360. Applying Minkwitz's algorithm, we first initialize the table as follows:

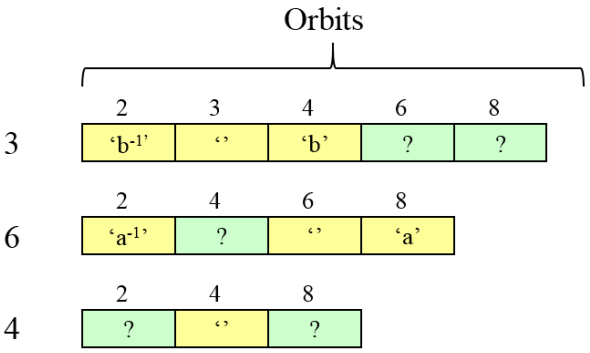


Now consider the word 'a', which corresponds to *g* = (1, 5, 7)(2, 6, 8). This has no effect on $k_1 = 3$. On the

other hand, it takes $k_2 = 6 \mapsto 8$. Thus, we write the word 'a' into the last entry of the second row:



After running through all words of length 1, we arrive at the following table:

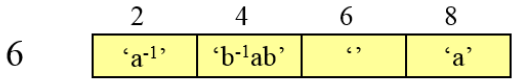


The first word of length 2 is 'aa', which corresponds to (1, 7, 5)(2, 8, 6), but this is just a^{-1}, and we have already processed it.

The next word of length 2 is 'ab', which gives $g = ab = (1,7)(2,3,4)(6,8)$. This takes $k_1 = 3 \mapsto 4$. However, the corresponding entry is already filled with 'b'. Since this new word does not improve upon the old one, we replace the word with $b^{-1}ab$. Now we have

$$g = b^{-1}ab = (1,7,5)(4,8,6).$$

and proceed on to k_2. This element takes $k_2 = 6 \mapsto 4$ so we fill in the corresponding entry on the second row:



Further Improvements

The above method is quite effective for most random groups. However, the last few entries of the table may take a really long time to fill up. E.g. on the set:

$$A = \{(1,2),(2,3),(3,4),\ldots,(n-1,n)\}, \quad \langle A \rangle = S_n$$

the shortest word which takes 1 to *n* is given by $(n-1,n)(n-2,n-1)\ldots(1,2)$. Intuitively, the table fills up slowly because the elements {1, ..., *n*} diffuse slowly via the generators.

One possible way out of this rut is to fill in entries of the table by looking at the group elements below the current row. To be specific, suppose the row for k_{i+1} has quite a few empty entries. We look at the filled entries on that row, say $x_1, x_2, \ldots$ and consider all words *w* found below the row. Their group elements $g \in G$ are guaranteed to lie in $G^{(i+1)} \leq G^{(i)}$. If the entry for $g(x_j)$ is currently empty, we fill it in with the concatenation of *w* and the corresponding word for $k_{i+1} \mapsto x_j$. [Of course we need to perform reduction after concatenating the two words.]

Optimization

To further optimize, note that sometimes a table entry gets replaced with a nice shorter word – it would be desirable to use this short word to update the other table entries.

Hence, we perform the following. For each row *i*, consider any two words *w'*, *w"* on that row. We take their concatenation *w* := *w'w"* and use it to update the entire table from row *i* onwards (by update, we mean: compute $x = w(k_i)$ and check if *w* is shorter than the table entry at *x*. If it is, we update; otherwise we let w_1 be this entry and replace *w* with $w_1^{-1}w$ and repeat the whole process).

As this process is rather slow, we only update the table if either *w'* or *w"* are of shorter length, compared to the last time we did this optimization.

Summary

Minkwitz's paper suggests the following iterative procedure.

- Set a maximum word length *l*.
- For each word of short length, update the table as described earlier.
  - If, at a certain row, the word length exceeds *l*, we bail out.
- For every *s* steps, do the following.
  - Perform the above two enhancements: filling up and optimizing.
  - Increase *l*, say, to (5/4)*l*.

Setting the word length is optional, but can speed things up quite a bit in practice. It prevents the initial *s* steps from filling up the table with overly long words, otherwise optimizing them will be costly. For further details, the diligent reader may refer to Minkwitz's original paper, available freely on the web.

> ***Outcome.***
>
> *We applied this to the study of the Rubik's cube group, and obtained a representation with maximum word length of 184 over all elements of the group. This is slightly worse than the result of 144-165 quoted in Minkwitz's paper.*

Case Study: Topspin Puzzle

The Topspin puzzle is a toy which consists of a loop with 20 labeled counters in it. In the middle of the loop lies a turntable which reverts the order of any four consecutive counters. This is what the puzzle looks like.

[Photo from product page on Amazon.]

Thus the group of permutations of the counters is generated by

a = (1, 2, 3, …, 20), b = (1, 4)(2, 3)

From the Schreier-Sims algorithm, we see that these two elements generate the full symmetric group S_{20} so Topspin achieves all possible permutations of the 20 counters. Minkwitz's algorithm gives us a practical way to solve the puzzle given any initial configuration. Our program gave us a full table with a maximum word length of 824.

To search for short words representing a given $g \in S_{20}$,

- let w be a short word in $\{a, b, a^{-1}, b^{-1}\}$;
- let $h \in S_{20}$ be the permutation for w;
- find the word w' for $h^{-1}g$;
- thus ww' is a word representing g.

We iterate over about 1000 instances of w and pick the shortest representation among all choices.

Example

Consider the transposition (1, 2). We obtain the following word:

$$a^{-1}a^{-1}b^{-1}a^{-1}b^{-1}ab^{-1}a^{-1}b^{-1}abab^{-1}a^{-1}a^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a^{-1}b$$

of length 43 (to be applied *from right to left*). This is a remarkably short word – for most random configurations we tried, the length of the word is >200.

Related

Casual Introduction to Group Theory (1)
September 21, 2012
In "Notes"

Schreier-Sims Algorithm
June 12, 2018
In "group actions"

Burnside's Lemma and Polya Enumeration Theorem (2)
January 15, 2013
In "Notes"

This entry was posted in Uncategorized and tagged group actions, group theory, permutations, rubik's cube, schreier-sims, symmetric group. Bookmark the permalink.