AM 106/206: Applied Algebra **Madhu Sudan**

Lecture Notes 9

October 3, 2016

References: Based on text by Akos Seress on Permutation Group Algorithms. Algorithm due to Sims.

1 Algorithms for Permutation Groups

Many basic tasks associated with a permutation group $G \leq S_n$ can be solved in time poly(n).

Describing G: First note that order of G can be as large as $n!$ and so exponential in n. Still one does not have to specify G by giving its multiplication table. Intead we can specify G by describing a set $S \subseteq G$ such that G equals the set generated by S. Hopefully such a generating set is much smaller than $n!$, and indeed this is true, of every *minimal* generating set.

Definition 1 ((Minimal) Generating Set:) *For set $S \subseteq G$, let $\langle S \rangle \triangleq \{\sigma_1 \cdots \sigma_k | \sigma_1, \ldots, \sigma_k \in S \cup S^{-1}\}$ denote the subgroup of generated by S. (Here $S^{-1} = \{\sigma^{-1} | \sigma \in S\}$. Note that for finite groups, we can replace $S \cup S^{-1}$ by just S. Why?) We use the convention that $\langle \emptyset \rangle = \{e\}$. We say S generates G if $\langle S \rangle = G$. We say S is a minimal generating set if $G \neq \langle T \rangle$ for every proper subset $T \subset S$.*

Proposition 2 *If S is a minimal generating set of a finite group G, then $|S| \leq \log_2 |G|$.*

Proof: Let $S = \{\sigma_1, \ldots, \sigma_k\}$. For $0 \leq i \leq k$, let $T_i = \{\sigma_1, \ldots, \sigma_i\}$. We claim that $\langle T_{i-1} \rangle \lneq \langle T_i \rangle$: By definition $\langle T_{i-1} \rangle \leq \langle T_i \rangle$ and if they are equal then $\sigma_i \in \langle T_{i-1} \rangle$ and so $\sigma_i \in \langle S - \{\sigma_i\} \rangle$ implying $G = \langle S - \{\sigma_i\} \rangle$ contradicting the minimality of S.

But now we are done, since now we must have $|\langle T_i \rangle| \geq 2 \cdot |\langle T_{i-1} \rangle|$ (the order of every subgroup divides the order of the group). So by induction we have $|\langle T_i \rangle| \geq 2^i$ and so $|G| = |\langle T_k \rangle| \geq 2^k = 2^{|S|}$. ■

We conclude from the above that if S is a minimal generating set of a subgroup $G \leq S_n$ then $|S| \leq \log(n!) = O(n \log n)$. Thus G can be specified with poly(n) bits. Interestingly enough, many algorithmic questions can also be decided in time polynomial in n. Two central problems are:

Membership Problem: Given $S \subseteq S_n$ and $\pi \in S_n$, determine if $\pi \in G \triangleq \langle S \rangle$.

Order Problem: Given $S \subseteq S_n$, determine the order of $G \triangleq \langle S \rangle$.

Both can be solved in time polynomial in n. Today we will describe an algorithm for the former problem.

2 Overview of rest of the lecture

1. We will start by defining a notion of a *Strong Generating Set* (SGS).
2. We will show that if we have an SGS for a group G then membership testing is straightforward.
3. We will finally give an algorithm for computing an SGS of G. Its run time will be obviously bounded by a polynomial in n, but correctness will not be obvious.
4. We will analyze the correctness of the SGS finding algoriothm.

3 Stabilizers, Types, and Strong Generating Sets

In order to motivate the definitions of this section, the idea we have in mind is that to generate the permutation π in G, we will find a "rich" enough generating set T (different that the set S given to us) that will allow use to generate π by first providing an element of T that corrects the first coordinate of π, and then operating on π so that this coordinate remains fixed in all future iterations. So at the beginning of iteration i we would have found $\tau_1, \ldots, \tau_{i-1} \in T$ such that $(\tau_{i-1} \circ \cdots \circ \tau_1)-1(j) = \pi^{-1}(j)$ for every $1 \leq j \leq i-1$ and in the jth iteration we will find τ_i such that this property extends to $j = i$ also. The needed condition on τ_i is that $\tau_i(j) = j$ for $j < i$ and $\tau_i^{-1}(i) = \pi^{-1}(i)$. A set T that allows us to find such elements τ_i is called a Strong Generating Set and we formalize all this below.

The idea of looking at subgroups of a group that fix some set of elements and only affect the others is a central one. The concept of a stabilizer is associated with this study. Even though we don't use this terminology much below, let us introduce it anyway. Given $X \subseteq [n]$, let $\text{stab}_X(G) = \{\sigma \in G | \sigma(i) = i \forall i \in X\}$. (For example in a Rubik's cube, if G corresponds to all the legal moves, G_X might be used to determine all the moves that do not disturb the top face — for an appropriate choice of X). For our purpose we will be happy enough with the subsets X of the form $\{1, \ldots, i\}$. For $i \in \{0, \ldots, n\}$ we let $G_i = \text{stab}_{\{1,\ldots,i\}}(G)$, i.e., the subgroup that fixes the elements 1 to i.

Definition 3 ((Extended) Types) *For $\sigma \in S_n$, we let the* type *of σ, denoted* $\text{type}(\sigma)$*, be the smallest integer i which is not fixed by σ, i.e., i is the integer such that $\sigma(1) = 1, \ldots, \sigma(i-1) = i-1$ and $\sigma(i) \neq i$. (For $\sigma \in G$, $\sigma \in G_{\text{type}(\sigma)-1}$ but $\sigma \notin G_{\text{type}(\sigma)}$.) We also define the extended type of σ, denoted $e\text{type}(\sigma)$, to be the pair (i, j) where $i = \text{type}(\sigma)$ and $j = \sigma^{-1}(i)$. Note $j > i$.*

Finally we define a strong generating set.

Definition 4 ((sub) Strong Generating Set (subSGS)) *A set $T \subseteq G$ is said to be a Strong Generating Set (SGS) of G if for every (i, j) such that there exists $\pi \in G$ with $e\text{type}(\pi) = (i, j)$, there exists a unique element $\tau \in T$ with $e\text{type}(\tau) = (i, j)$. A set T is a subSGS for G if it is a subset of an SGS for G. Equivalently*[1]*, T contains at most one element of every type (i, j).*

[1] This equivalence is not syntactic. You should really check that this holds!

4 Membership Algorithm using an SGS

We now formalize the natural algorithm which we hoped to use when defining the notion of an SGS. This algorithm will not only help decide membership, but also help us note that the SGS is indeed a generating set of G.

REDUCE(π, T) /* Returns e iff $\pi \in G$, if T is a generating set of G */
If $\pi = e$ return (e).
Else if there exists $\tau \in T$ with $e\text{type}(\tau) = e\text{type}(\pi)$ Return (REDUCE$(\pi \circ \tau^{-1}, T)$)
Else Return(π).

We now analyze the algorithm above:

Correctness: We now claim that the algorithm returns e if and only if $\pi \in G$. This is so by induction on the number of recursive calls of the algorithm (which we assume to be finite, and prove in the next para). If $\pi = e$ then obviously $\pi \in G$ and the algorithm returns e. Now if the algorithm does not find $\tau \in T$ with $e\text{type}(\tau) = e\text{type}(\pi)$ then $\pi \notin G$ since otherwise an SGS should contain a representative of the type of π. Finally we consider the case that the algorithm makes a recursive call. In this case we have that $\tau \in T \subseteq G$ and so $\pi \in G \Leftrightarrow \pi \circ \tau^{-1} \in G$. By induction the algorithm returns e on input $\pi \circ \tau^{-1}$ if and only if $\pi \circ \tau^{-1} \in G$ which holds if and only if $\pi \in G$, concluding the claim.

Running time: Next we claim that the algorithm makes at most n recursive calls. In particular we claim that $\text{type}(\pi) < \text{type}(\pi \circ \tau^{-1})$. This is easily verified. Since $\text{type}(\pi) = \text{type}(\tau) = i$, we have for $k < i$, $\pi(\tau^{-1}(k)) = \pi(k) = k$. Furthermore, we also have $\pi^{-1}(i) = \tau^{-1}(i) = j$ (since $e\text{type}(\pi) = e\text{type}(\pi) = (i, j)$). So $\pi(\tau^{-1}(i)) = \pi(j) = i$. Thus $\text{type}(\pi \circ \tau^{-1}) > i = \text{type}(\pi)$. Thus after each recursive call the type increases. Since $1 \leq \text{type}(\pi) \leq n$, we have that the algorithm makes at most n recursive calls.

We conclude that there is a polynomial time algorithm to decide if $\pi \in G$ given a strong generating set T for G.

Lemma 5 *Algorithm* REDUCE(π, T) *runs in time* poly(n) *and returns* e *if and only if* $\pi \in G$.

Some additional facts that the argument describes above will be useful in what follows. Suppose T is a subSGS for G and suppose we run REDUCE on $\pi \in G$. Let the algorithm run for k iterations, discovering $\tau_1, \ldots, \tau_k \in T$ before reporting σ. We have the following:

1. $\pi = \sigma \circ \tau_k \circ \cdots \circ \tau_1$. In particular if T is an SGS then $\sigma = e$ and $\pi = \tau_k \circ \cdots \circ \tau_1 \in \langle T \rangle$.
2. $\sigma \in G$.
3. $\text{type}(\tau_i) > \text{type}(\tau_{i-1})$ for every i.
4. $\text{type}(\sigma) \neq \text{type}(\tau)$ for every $\tau \in T$ and so $T \cup \{\sigma\}$ is also a subSGS for G.

We summarize the observation from Item (1) above as a proposition, before turning to the task of building a strong generating set.

Proposition 6 *Every SGS is also a generating set.*

5 Constructing an SGS

The hope for constructing an SGS is simple. We will start with an empty subSGS T and iteratively add elements to T till we are done. Since the numer of distinct types is at most $\binom{n}{2}$ this takes at most $\binom{n}{2} = O(n^2)$ iterations. The main question is how to find elements to add to T. We can start with the elements of S and REDUCE them with the current T and if the output is not e, we can add the resulting element to T. (See Item (4) above.) But what do we do once we are done with elements of S? We could now try products of pairs of elements in S. Then triples, and so on. But when do we stop?

A somewhat better strategy turns out to be to try products of elements in $S \cup T$. Certainly this is legitimate too. Somewhat remarkably, this suffices! We assume this to describe the algorithm and analyze its run time, and prove correctness in the next section.

BUILD-SGS(S) /* Returns SGS T for $G = \langle S \rangle$ */
  Initialize $T \leftarrow \emptyset$.
  While $\exists\ \sigma, \rho \in S \cup T \cup \{e\}$ such that $\tau \triangleq$REDUCE$(\sigma \circ \rho, T) \neq e$
    $T \leftarrow T \cup \{\tau\}$
    endwhile
  Return(T)

Runtime analysis: Let $|S| = \ell$. Note that the input size is $\ell \cdot n$. The number of iterations of the while loop is at most $O(n^2)$. Each iteration requires trying out at most $(\ell + n^2)^2$ possible pairs σ, ρ and running REDUCE on them (which in turn takes $O(n)$ iterations each of which takes $O(n)$ time). Putting all this together we get (not-so-respectable-but-nevertheless) polynomial running time in $\ell + n$.

However the correctness is not so obvious. It is clear that at all times we have a subSGS. But do we have an SGS at the end? We prove this in the next section.

6 Correctness Analysis

Our goal now is to show that when the algorithm stops it returns an SGS. We start with a weaker claim.

Lemma 7 *If* BUILD-SGS(S) *returns T, then $\langle S \rangle = \langle T \rangle$.*

Proof: Since T is always (through every iteration) a subset of $\langle S \rangle$, it suffices to show that at the end of all iterations, $S \subseteq \langle T \rangle$. By the termination condition, taking $\rho = e$, we know that for every $\sigma \in S$, REDUCE$(\sigma, T) = e$. By Observation (1), we have that $\sigma = \tau_1 \circ \cdots \circ \tau_k$ for some $\tau_1, \ldots, \tau_k \in T$, and so $\sigma \in \langle T \rangle$. We conclude ■

So it suffices to prove the following fact which we promote to a "theorem" to conclude the correctness of the algorithm above.

Theorem 8 *If $T \subseteq S_n$ satisfies for every $\sigma, \rho \in T \cup \{e\}$,* REDUCE$(\sigma \circ \rho, T) = e$*, then T is an SGS for $\langle T \rangle$.*

Let us start with some notation that will prove useful. For a subSGS T, let

$$|T\rangle \triangleq \{\tau_k \circ \cdots \circ \tau_1 | \tau_1, \ldots, \tau_k \in T, \text{type}(\tau_1) < \text{type}(\tau_2) \cdots < \text{type}(\tau_k)\}.$$

Note that $|T\rangle$ is exactly the set of elements π such that $\textsc{Reduce}(\pi, T) = e$. In terms of this notation, we wish to show that if $\sigma \circ \rho \in |T\rangle$ for every $\sigma, \rho \in T \cup \{e\}$ then $|T\rangle = \langle T\rangle$.

Before rushing into the proof, lets think of the potential strategy. We will use a careful induction to keep the proof tame. Suppose we wish to show $\pi = \tau_1 \circ \cdots \circ \tau_m \in |T\rangle$. We can use induction on m to assume $\pi' = \tau_1 \circ \cdots \circ \tau_{m-1} \in |T\rangle$ and so $\pi' = \tau'_k \cdots \tau'_1$ for some $\tau'_1, \ldots, \tau'_k \in T$ with $\text{type}(\tau'_i) < \text{type}(\tau'_{i+1})$. So $\pi = \tau'_k \cdots \tau'_1 \circ \tau_m$. If $k < m-1$ we would be done by the length of the sequence, but we have no way of assuming this. If $\text{type}(\tau'_1) > \text{type}(\tau_m)$ then also we would be done since this would prove $\pi \in |T\rangle$ by definition of $|T\rangle$. We need to deal with the case where $\text{type}(\tau'_1) \leq \text{type}(\tau_m)$. This would probably be a good time to look at the assumption about T! It tells that for any pair of elements in T their product is in $|T\rangle$. Let us apply this to τ'_1 and τ_m. We get that we can write $\tau'_1 \circ \tau_m = \alpha_\ell \cdots \alpha_1$ with α_i's having increasing types as we increase i. We now stare at the result $\pi = \tau'_k \cdots \tau'_2 \circ \alpha_\ell \cdots \alpha_1$ and wonder what progress we've made? [DRAW PICTURE HERE] It turns out that we managed to push the element of the least type all the way out to the end, and this is actually progress. The reason being if we could assume by induction that $\pi'' = \tau'_k \cdots \tau'_2 \circ \alpha_\ell \cdots \alpha_2$, by virtue of having higher type than π, is also in $|T\rangle$ and equals $\beta_t \cdots \beta_1$ then $\pi\beta_t \cdots \beta_1 \circ \alpha_1$ would be a product of decreasing types and also live in $|T\rangle$. A few things need to be checked to make sure all the above worked. Is it really the case that π'' has a higher type than π? Is the final product really one of decreasing types. And is that induction really legit? We had both the length of the sequence m and the type of π as induction variables. How do we blend the two? Well ... we do it with care.

Define the type of a sequence $\tau_1, \ldots, \tau_m$ to be the minimum of the types of the elements in the sequence. We prove that for every finite sequence $\tau_1, \ldots, \tau_m$, the product $\tau_1 \cdots \tau_m \in |T\rangle$ first by reverse induction on the type of the sequence (i.e., we assume it is true for all types in $\{i+1, \ldots, n\}$ and then prove it for sequences of type i) and then by induction on the length of the sequence (so we can assume $\tau_1 \cdots \tau m - 1 \in |T\rangle$ because it has type no smaller than $\tau_1, \ldots, \tau_m$, and furthermore it is of smaller length.

It turns out the above can now all be carried out as a formal proof - but we skip the details since it is repetitive and we are tired.