Some Algorithmic questions in Finite Group Theory

V Arvind
Institute of Mathematical Sciences, Chennai
India
email arvind@imsc.res.in

June 30, 2015

Plan of Talk

- Permutation groups: background and some basic algorithms.

Plan of Talk

- Permutation groups: background and some basic algorithms.
- Fixed point-free elements of a permutation group. Existence and computation.

Plan of Talk

- Permutation groups: background and some basic algorithms.
- Fixed point-free elements of a permutation group. Existence and computation.
- Computing small bases for permutation groups.

Plan of Talk

- Permutation groups: background and some basic algorithms.
- Fixed point-free elements of a permutation group. Existence and computation.
- Computing small bases for permutation groups.
- Random subproducts in groups.

What is in this talk for me?

- The group theory content is easy/elementary and is from a first course in algebra.

What is in this talk for me?

• The group theory content is easy/elementary and is from a first course in algebra.

• The algorithms are straightforward based on simple techniques taught in a first algorithms course.

What is in this talk for me?

• The group theory content is easy/elementary and is from a first course in algebra.

• The algorithms are straightforward based on simple techniques taught in a first algorithms course.

• What's new is probably the cocktail of group theory + algorithms.

What is in this talk for me?

• The group theory content is easy/elementary and is from a first course in algebra.

• The algorithms are straightforward based on simple techniques taught in a first algorithms course.

• What's new is probably the cocktail of group theory + algorithms.

• Teachers: Experimenting with similar cocktails in other courses like linear algebra can be interesting.

Permutation Groups: Definitions

- Let S_n denote the group of all permutations on n elements, say $\{1, 2, \ldots, n\}$. It is a group under permutation composition and has $n!$ many elements.

Permutation Groups: Definitions

- Let S_n denote the group of all permutations on n elements, say $\{1, 2, \ldots, n\}$. It is a group under permutation composition and has $n!$ many elements.

- A subgroup G of S_n, denoted $G \leq S_n$, is a *permutation group*.

Permutation Groups: Definitions

• Let S_n denote the group of all permutations on n elements, say $\{1, 2, \ldots, n\}$. It is a group under permutation composition and has $n!$ many elements.

• A subgroup G of S_n, denoted $G \leq S_n$, is a *permutation group*.

• We can "describe" a permutation group G by listing down all its elements. A more compact description is to give a generating set for it.

Permutation Groups: Definitions

- Let S_n denote the group of all permutations on n elements, say $\{1, 2, \ldots, n\}$. It is a group under permutation composition and has $n!$ many elements.

- A subgroup G of S_n, denoted $G \leq S_n$, is a *permutation group*.

- We can "describe" a permutation group G by listing down all its elements. A more compact description is to give a generating set for it.

- The permutation group $\langle S \rangle$, *generated* by a subset $S \subseteq S_n$ of permutations, is the smallest subgroup of S_n containing S.

Permutation Groups: Definitions Contd.

• Every finite group G has a generating set of size $\log_2 |G|$.
Because

$$\langle g_1 \rangle < \langle g_1, g_2 \rangle < \ldots < \langle g_1, g_2, \ldots, g_k \rangle = G.$$

Permutation Groups: Definitions Contd.

• Every finite group G has a generating set of size $\log_2 |G|$.
Because

$$\langle g_1 \rangle < \langle g_1, g_2 \rangle < \ldots < \langle g_1, g_2, \ldots, g_k \rangle = G.$$

Each new generating element at least doubles the group size by Lagrange's theorem. Thus, $k \leq \log_2 |G|$.

Permutation Groups: Definitions Contd.

• Every finite group G has a generating set of size $\log_2 |G|$. Because

$$\langle g_1 \rangle < \langle g_1, g_2 \rangle < \ldots < \langle g_1, g_2, \ldots, g_k \rangle = G.$$

Each new generating element at least doubles the group size by Lagrange's theorem. Thus, $k \leq \log_2 |G|$.

So, giving a generating set for G is a succinct representation as it as algorithmic input.

Permutation Groups: Definitions Contd.

- For a permutation $\pi \in S_n$, the image of a point $i \in [n]$ is denoted by i^π.

Permutation Groups: Definitions Contd.

- For a permutation $\pi \in S_n$, the image of a point $i \in [n]$ is denoted by i^π.

- For a permutation $\pi \in S_n$, a point $i \in [n]$ is a *fixed point* if $i^\pi = i$. Let $\mathrm{fix}(\pi)$ denote the number of points fixed by π.

Permutation Groups: Definitions Contd.

• For a permutation $\pi \in S_n$, the image of a point $i \in [n]$ is denoted by i^π.

• For a permutation $\pi \in S_n$, a point $i \in [n]$ is a *fixed point* if $i^\pi = i$. Let $\mathrm{fix}(\pi)$ denote the number of points fixed by π.

• A permutation group $G \leq S_n$ partitions the domain $[n]$ into *orbits*: i and j are in the same orbit precisely when $i^g = j$ for some $g \in G$.

Permutation Groups: Definitions Contd.

• For a permutation $\pi \in S_n$, the image of a point $i \in [n]$ is denoted by i^π.

• For a permutation $\pi \in S_n$, a point $i \in [n]$ is a *fixed point* if $i^\pi = i$. Let $\mathrm{fix}(\pi)$ denote the number of points fixed by π.

• A permutation group $G \leq S_n$ partitions the domain $[n]$ into *orbits*: i and j are in the same orbit precisely when $i^g = j$ for some $g \in G$.

• The group G is called *transitive* if there is exactly one orbit.

Permutation Group Algorithmics

- Each permutation π in S_n can be represented as an n-tuple $(1^\pi, 2^\pi, \ldots, n^\pi)$ (or an array of n integers).

Permutation Group Algorithmics

• Each permutation π in S_n can be represented as an n-tuple $(1^\pi, 2^\pi, \ldots, n^\pi)$ (or an array of n integers).

• Given $\pi \in S_n$ and a point $i \in [n]$ we can "compute" i^π in "one step" by looking up the i^{th} entry of the array representing π. We can consider this a "unit cost" operation.

Permutation Group Algorithmics

• Each permutation π in S_n can be represented as an n-tuple $(1^\pi, 2^\pi, \ldots, n^\pi)$ (or an array of n integers).

• Given $\pi \in S_n$ and a point $i \in [n]$ we can "compute" i^π in "one step" by looking up the i^{th} entry of the array representing π. We can consider this a "unit cost" operation.

• Given two permutations $\pi, \psi \in S_n$ we can compute their product $\pi\psi$ by computing $(i^\pi)^\psi$ for each i. This operation costs n.

Permutation Group Algorithmics

• Each permutation π in S_n can be represented as an n-tuple $(1^\pi, 2^\pi, \ldots, n^\pi)$ (or an array of n integers).

• Given $\pi \in S_n$ and a point $i \in [n]$ we can "compute" i^π in "one step" by looking up the i^{th} entry of the array representing π. We can consider this a "unit cost" operation.

• Given two permutations $\pi, \psi \in S_n$ we can compute their product $\pi\psi$ by computing $(i^\pi)^\psi$ for each i. This operation costs n.

• What is an efficient algorithm on permutation groups?

Permutation Group Algorithmics Contd.

- Elements and subgroups of S_n require encoding size n and $n^2 \log n$ respectively.

Permutation Group Algorithmics Contd.

- Elements and subgroups of S_n require encoding size n and $n^2 \log n$ respectively.

- Roughly speaking, for algorithm dealing with permutation groups in S_n:

Permutation Group Algorithmics Contd.

• Elements and subgroups of S_n require encoding size n and $n^2 \log n$ respectively.

• Roughly speaking, for algorithm dealing with permutation groups in S_n:

Polynomial in n many operations = Efficient.

Permutation Group Algorithmics Contd.

• Elements and subgroups of S_n require encoding size n and $n^2 \log n$ respectively.

• Roughly speaking, for algorithm dealing with permutation groups in S_n:

Polynomial in n many operations = Efficient.

Exponential in n operations = Inefficient.

Computing Orbits Efficiently

- Given $G \leq S_n$ by a generating set S, we can compute the orbit of any point i in $(n|S|)^{O(1)}$ time.

Computing Orbits Efficiently

• Given $G \leq S_n$ by a generating set S, we can compute the orbit of any point i in $(n|S|)^{O(1)}$ time.

Input: $S = \{g_1, g_2, \ldots, g_k\}$ generators for G;
$O := \{i\}$;
while O changes **do**
$O := O \cup \{i^{g_j} \mid i \in S, 1 \leq j \leq k\}$;
endwhile

Computing Orbits Efficiently

• Given $G \leq S_n$ by a generating set S, we can compute the orbit of any point i in $(n|S|)^{O(1)}$ time.

Input: $S = \{g_1, g_2, \ldots, g_k\}$ generators for G;
$O := \{i\}$;
while O changes **do**
$O := O \cup \{i^{g_j} \mid i \in S, 1 \leq j \leq k\}$;
endwhile

• The loop runs for at most n steps. In the loop the number of operations is bounded by $O(nk)$. Thus, $O(n^2 k)$ operations in all.

The Membership Testing Problem

- Given as input $\pi \in S_n$ and a subgroup $G = \langle S \rangle \leq S_n$ test if π is in G. Express π in terms of the generators.

The Membership Testing Problem

- Given as input $\pi \in S_n$ and a subgroup $G = \langle S \rangle \leq S_n$ test if π is in G. Express π in terms of the generators.

- Writing π as a product of generators may be exponentially long!

The Membership Testing Problem

• Given as input $\pi \in S_n$ and a subgroup $G = \langle S \rangle \leq S_n$ test if π is in G. Express π in terms of the generators.

• Writing π as a product of generators may be exponentially long!

Example Consider the cyclic group $G = \langle g \rangle$, where g is a permutation of order $2^{O(\sqrt{n \log n})}$ (by choosing g to be a product of cycles of prime length for different primes).

The Membership Testing Problem

- Given as input $\pi \in S_n$ and a subgroup $G = \langle S \rangle \leq S_n$ test if π is in G. Express π in terms of the generators.

- Writing π as a product of generators may be exponentially long!

Example Consider the cyclic group $G = \langle g \rangle$, where g is a permutation of order $2^{O(\sqrt{n \log n})}$ (by choosing g to be a product of cycles of prime length for different primes).

We need to have a more compact way of expressing $\pi \in G$ in terms of its generators.

Membership Testing Contd.

Elements of G are g^b where b is $t = O(\sqrt{n \log n})$ bits. We compute g^b by repeated squaring and multiplying the appropriate powers g^{2^i}. Let $b = \sum_{i=0}^{t-1} b_i 2^i$.

Membership Testing Contd.

Elements of G are g^b where b is $t = O(\sqrt{n \log n})$ bits. We compute g^b by repeated squaring and multiplying the appropriate powers g^{2^i}. Let $b = \sum_{i=0}^{t-1} b_i 2^i$.

The following *straight-line program* computes g^b:

Membership Testing Contd.

Elements of G are g^b where b is $t = O(\sqrt{n \log n})$ bits. We compute g^b by repeated squaring and multiplying the appropriate powers g^{2^i}. Let $b = \sum_{i=0}^{t-1} b_i 2^i$.

The following *straight-line program* computes g^b:

$x_0 := g$;
for $i := 1$ to $t-1$ **do**
$x_i := x_{i-1}^2$;
$x_t := 1$;
for $i := 1$ to $t-1$ **do**
$x_{t+i} := x_{t+i-1} \cdot x_i^{b_i}$;

Srtaight-Line Programs

Let $\pi \in G = \langle g_1, g_2, \ldots, g_k \rangle \leq S_n$. A straight-line program for π consists of the following:

Srtaight-Line Programs

Let $\pi \in G = \langle g_1, g_2, \ldots, g_k \rangle \leq S_n$. A straight-line program for π consists of the following:

• The first k lines of the program has $x_i := g_i, 1 \leq i \leq k$ as instructions, where $g_1, g_2, \ldots, g_k$ are the generators of G.

Srtaight-Line Programs

Let $\pi \in G = \langle g_1, g_2, \ldots, g_k \rangle \leq S_n$. A straight-line program for π consists of the following:

• The first k lines of the program has $x_i := g_i, 1 \leq i \leq k$ as instructions, where $g_1, g_2, \ldots, g_k$ are the generators of G.

• Each subsequent line is an instruction of the form:

$$x_i := x_j x_k$$

Where $j < i$ and $k < i$.

Srtaight-Line Programs

Let $\pi \in G = \langle g_1, g_2, \ldots, g_k \rangle \leq S_n$. A straight-line program for π consists of the following:

• The first k lines of the program has $x_i := g_i, 1 \leq i \leq k$ as instructions, where $g_1, g_2, \ldots, g_k$ are the generators of G.

• Each subsequent line is an instruction of the form:

$$x_i := x_j x_k$$

Where $j < i$ and $k < i$.

• **Nice Fact** If $\pi \in \langle g_1, g_2, \ldots, g_k \rangle \leq S_n$ then it has a straight-line program of length polynomial in n and k, and the membership testing algorithm will find in $poly(n)$ time.

Strong Generating Sets

For $G \leq S_n$ let $G_{[i]}$ denote its subgroup that *pointwise stabilizes* $\{1, 2, \ldots, i\}$.

Strong Generating Sets

For $G \leq S_n$ let $G_{[i]}$ denote its subgroup that *pointwise stabilizes* $\{1, 2, \ldots, i\}$.

- Consider the tower of stabilizers subgroups in G:

$$\{id\} = G_{[n-1]} < G_{[n-2]} < \ldots < G_{[1]} < G_{[0]} = G.$$

Strong Generating Sets

For $G \leq S_n$ let $G_{[i]}$ denote its subgroup that *pointwise stabilizes* $\{1, 2, \ldots, i\}$.

- Consider the tower of stabilizers subgroups in G:

$$\{id\} = G_{[n-1]} < G_{[n-2]} < \ldots < G_{[1]} < G_{[0]} = G.$$

Consider the right coset representative sets T_i for $G_{[i]}$ in $G_{[i-1]}, 1 \leq i \leq n-1$. Their union forms a *strong generating set* for G.

Strong Generating Sets

For $G \leq S_n$ let $G_{[i]}$ denote its subgroup that *pointwise stabilizes* $\{1, 2, \ldots, i\}$.

- Consider the tower of stabilizers subgroups in G:

$$\{id\} = G_{[n-1]} < G_{[n-2]} < \ldots < G_{[1]} < G_{[0]} = G.$$

Consider the right coset representative sets T_i for $G_{[i]}$ in $G_{[i-1]}, 1 \leq i \leq n-1$. Their union forms a *strong generating set* for G.

- "Strong" because every $\pi \in G$ can be expressed uniquely as a "short" product $\pi = \pi_{n-1}\pi_{n-2}\ldots\pi_1$, where $\pi_i \in T_i$.

Back to Membership Testing

• Given a strong generating set for $G \leq S_n$, membership testing is easy and efficient. Let $\pi \in S_n$:

$$\pi \in G \iff \pi \in G_{[1]}\pi_1 \text{ for } \pi_1 \in T_1.$$

We can find $\pi_1 \in T_1$ easily and the problem reduces to checking if $\pi\pi_1^{-1}$ is in $G_{[1]}$.

Back to Membership Testing

• Given a strong generating set for $G \leq S_n$, membership testing is easy and efficient. Let $\pi \in S_n$:

$$\pi \in G \iff \pi \in G_{[1]}\pi_1 \text{ for } \pi_1 \in T_1.$$

We can find $\pi_1 \in T_1$ easily and the problem reduces to checking if $\pi\pi_1^{-1}$ is in $G_{[1]}$.

• How do we find a strong generating set for G?

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Finding a Strong Generating Set

- Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Proof
We know $G = HR$.

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Proof

We know $G = HR$.

And we have $RA \subseteq BR$

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Proof

We know $G = HR$.

And we have $RA \subseteq BR$

Which implies $RAA \subseteq BRA \subseteq BBR$.

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Proof

We know $G = HR$.

And we have $RA \subseteq BR$

Which implies $RAA \subseteq BRA \subseteq BBR$.

Repeating, we get $R\langle A \rangle \subseteq \langle B \rangle R$.

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Proof

We know $G = HR$.

And we have $RA \subseteq BR$

Which implies $RAA \subseteq BRA \subseteq BBR$.

Repeating, we get $R\langle A \rangle \subseteq \langle B \rangle R$.

Hence $G = \langle B \rangle R$.

Finding a Strong Generating Set

• Given $G = \langle S \rangle$, finding T_1 is easy. We can compute the orbit O of 1, and for each $j \in O$ keep track of a $\pi_1 \in G$ such that $1^{\pi_1} = j$. How do we find T_2?

Schreier's Lemma Let $G = \langle A \rangle$ and $H \leq G$ of finite index with R as the set of right coset representatives. Then H is generated by the set $B = \{r_1 a r_2^{-1} \in H \mid a \in A, r_1, r_2 \in R\}$.

Proof

We know $G = HR$.

And we have $RA \subseteq BR$

Which implies $RAA \subseteq BRA \subseteq BBR$.

Repeating, we get $R\langle A \rangle \subseteq \langle B \rangle R$.

Hence $G = \langle B \rangle R$.

Since $\langle B \rangle \leq H$ it follows that $\langle B \rangle = H$.

Finding a Strong Generating Set

Difficulty Applying Schreier's lemma, the number of generators for $G_{[1]}$ can be $n|A|$. For $G_{[2]}$ it can grow to $n^2|A|$ and so on...

Finding a Strong Generating Set

Difficulty Applying Schreier's lemma, the number of generators for $G_{[1]}$ can be $n|A|$. For $G_{[2]}$ it can grow to $n^2|A|$ and so on...

Solution: a "reduce" step Given $G = \langle g_1, g_2, \ldots, g_k \rangle$, we can efficiently find a generating set of size $O(n^2)$.

Finding a Strong Generating Set

Difficulty Applying Schreier's lemma, the number of generators for $G_{[1]}$ can be $n|A|$. For $G_{[2]}$ it can grow to $n^2|A|$ and so on...

Solution: a "reduce" step Given $G = \langle g_1, g_2, \ldots, g_k \rangle$, we can efficiently find a generating set of size $O(n^2)$.

for $i = 1$ **to** n **do**

while there are generators x, y fixing $1, 2, \ldots, i-1$
such that $i^x = i^y$ **do**

replace the pair x, y with the pair x, yx^{-1}.

end-while

end-for

How many generators are needed?

• **Exercise** Given $G = \langle S \rangle \leq S_n$ as input we can efficiently compute a generating set of size at most $n-1$.

Hint:

How many generators are needed?

• **Exercise** Given $G = \langle S \rangle \leq S_n$ as input we can efficiently compute a generating set of size at most $n - 1$.

Hint:

For each $g \in S$, let $i_g \in [n]$ be the smallest point moved by g.

How many generators are needed?

• **Exercise** Given $G = \langle S \rangle \leq S_n$ as input we can efficiently compute a generating set of size at most $n-1$.

Hint:

For each $g \in S$, let $i_g \in [n]$ be the smallest point moved by g.

Consider the graph X_S on vertex set $\{1, 2, \ldots, n\}$ and edge set $\{(i_g, i_g^g) \mid g \in S\}$.

How many generators are needed?

- **Exercise** Given $G = \langle S \rangle \leq S_n$ as input we can efficiently compute a generating set of size at most $n-1$.

Hint:

For each $g \in S$, let $i_g \in [n]$ be the smallest point moved by g.

Consider the graph X_S on vertex set $\{1, 2, \ldots, n\}$ and edge set $\{(i_g, i_g^g) \mid g \in S\}$.

As long as X_S has cycles, we can apply a modified reduce step to shrink the size of S.

How many generators are needed?

• **Exercise** Given $G = \langle S \rangle \leq S_n$ as input we can efficiently compute a generating set of size at most $n-1$.

Hint:

For each $g \in S$, let $i_g \in [n]$ be the smallest point moved by g.

Consider the graph X_S on vertex set $\{1, 2, \ldots, n\}$ and edge set $\{(i_g, i_g^g) \mid g \in S\}$.

As long as X_S has cycles, we can apply a modified reduce step to shrink the size of S.

• **McIver-Neumann** Every subgroup of S_n has a generating set of size at most $n/2$. Proof uses CFSG. No efficient algorithm is known for it.

How many generators are needed?

• **Exercise** Given $G = \langle S \rangle \leq S_n$ as input we can efficiently compute a generating set of size at most $n-1$.

Hint:

For each $g \in S$, let $i_g \in [n]$ be the smallest point moved by g.

Consider the graph X_S on vertex set $\{1, 2, \ldots, n\}$ and edge set $\{(i_g, i_g^g) \mid g \in S\}$.

As long as X_S has cycles, we can apply a modified reduce step to shrink the size of S.

• **McIver-Neumann** Every subgroup of S_n has a generating set of size at most $n/2$. Proof uses CFSG. No efficient algorithm is known for it.

Finding a Strong Generating Set

Theorem (Schreier-Sims)

Let $G < S_n$ be input by some generating set. In polynomial time we can compute a strong generating set $\cup T_i$ with the following properties:

1. *Every element $\pi \in G$ can be expressed uniquely as a product $\pi = \pi_1 \pi_2 \dots \pi_{n-1}$ with $\pi_i \in T_i$,*

Finding a Strong Generating Set

Theorem (Schreier-Sims)

Let $G < S_n$ be input by some generating set. In polynomial time we can compute a strong generating set $\cup T_i$ with the following properties:

1. *Every element $\pi \in G$ can be expressed uniquely as a product $\pi = \pi_1 \pi_2 \ldots \pi_{n-1}$ with $\pi_i \in T_i$,*
2. *Membership in G of a given permutation can be tested in polynomial time.*

Finding a Strong Generating Set

Theorem (Schreier-Sims)

Let $G < S_n$ be input by some generating set. In polynomial time we can compute a strong generating set $\cup T_i$ with the following properties:

1. *Every element $\pi \in G$ can be expressed uniquely as a product $\pi = \pi_1 \pi_2 \ldots \pi_{n-1}$ with $\pi_i \in T_i$,*
2. *Membership in G of a given permutation can be tested in polynomial time.*
3. *$|G|$ can be computed in polynomial time.*

Finding a Strong Generating Set

Theorem (Schreier-Sims)

Let $G < S_n$ be input by some generating set. In polynomial time we can compute a strong generating set $\cup T_i$ with the following properties:

1. *Every element $\pi \in G$ can be expressed uniquely as a product $\pi = \pi_1 \pi_2 \dots \pi_{n-1}$ with $\pi_i \in T_i$,*
2. *Membership in G of a given permutation can be tested in polynomial time.*
3. *$|G|$ can be computed in polynomial time.*

Running Time Analysis

Suppose $G = \langle S \rangle$ is the input group.

Running Time Analysis

Suppose $G = \langle S \rangle$ is the input group.

• Initial reduce operation if $|S| > n^2$. For $i = 1, 2, \ldots, n-1$ we compute i^g for $|S|$ many g. After that at most $|S|n$ many replacements of x, y by x, yx^{-1}.

Running Time Analysis

Suppose $G = \langle S \rangle$ is the input group.

• Initial reduce operation if $|S| > n^2$. For $i = 1, 2, \ldots, n-1$ we compute i^g for $|S|$ many g. After that at most $|S|n$ many replacements of x, y by x, yx^{-1}.

• In Schreier's lemma: computing orbit of i takes $n^2|S| \leq n^4$ operations, which also gives transversal R for $G_{[i]}$ in $G_{[i-1]}$.

Running Time Analysis

Suppose $G = \langle S \rangle$ is the input group.

• Initial reduce operation if $|S| > n^2$. For $i = 1, 2, \ldots, n-1$ we compute i^g for $|S|$ many g. After that at most $|S|n$ many replacements of x, y by x, yx^{-1}.

• In Schreier's lemma: computing orbit of i takes $n^2|S| \leq n^4$ operations, which also gives transversal R for $G_{[i]}$ in $G_{[i-1]}$.

• $|R| \cdot |S| = n^3$ operations for computing generating set of $G_{[i]}$. Applying reduce operation costs $|S|n \leq n^4$ operations.

Running Time Analysis

Suppose $G = \langle S \rangle$ is the input group.

• Initial reduce operation if $|S| > n^2$. For $i = 1, 2, \ldots, n-1$ we compute i^g for $|S|$ many g. After that at most $|S|n$ many replacements of x, y by x, yx^{-1}.

• In Schreier's lemma: computing orbit of i takes $n^2|S| \leq n^4$ operations, which also gives transversal R for $G_{[i]}$ in $G_{[i-1]}$.

• $|R| \cdot |S| = n^3$ operations for computing generating set of $G_{[i]}$. Applying reduce operation costs $|S|n \leq n^4$ operations.

Overall costs is $n^4 \cdot n$ plus $O(|S|n)$ operations. Each operation costs $O(n)$. Thus, $O(n^6 + |S|n^2)$ is the time.

Orbit Counting Lemma

$i \in [n]$ is a *fixpoint* of $g \in G$ if $i^g = i$.

Orbit Counting Lemma

$i \in [n]$ is a *fixpoint* of $g \in G$ if $i^g = i$.

$\mathrm{fix}(g)$ = number of fixpoints of g.

Orbit Counting Lemma

$i \in [n]$ is a *fixpoint* of $g \in G$ if $i^g = i$.

$\mathrm{fix}(g) =$ number of fixpoints of g.

$\mathrm{orb}(G) =$ number of orbits of G.

Orbit Counting Lemma

$i \in [n]$ is a *fixpoint* of $g \in G$ if $i^g = i$.

$\mathrm{fix}(g)$ = number of fixpoints of g.

$\mathrm{orb}(G)$ = number of orbits of G.

Lemma (Orbit Counting Lemma)

Let $G \leq S_n$ and $\mathrm{orb}(G)$ denote the number of orbits of G. Then

$$\mathrm{orb}(G) = \frac{1}{|G|} \sum_{g \in G} \mathrm{fix}(g) = \mathbb{E}_{g \in G}[\mathrm{fix}(g)].$$

Orbit Counting Lemma

$i \in [n]$ is a *fixpoint* of $g \in G$ if $i^g = i$.

$\mathrm{fix}(g)$ = number of fixpoints of g.

$\mathrm{orb}(G)$ = number of orbits of G.

Lemma (Orbit Counting Lemma)

Let $G \leq S_n$ and $\mathrm{orb}(G)$ denote the number of orbits of G. Then

$$\mathrm{orb}(G) = \frac{1}{|G|} \sum_{g \in G} \mathrm{fix}(g) = \mathbb{E}_{g \in G}[\mathrm{fix}(g)].$$

Proof Define 0-1 matrix: $M_{g,i} = 1$ iff $i^g = i$. Equate row-wise and column-wise sums using $|O(i)| = |G|/|G_i|$.

Fixpoint free elements in G

Theorem (Jordan's Theorem)

If $G \leq S_n$ is transitive then the group G has fixpoint free elements.

Fixpoint free elements in G

Theorem (Jordan's Theorem)

If $G \leq S_n$ is transitive then the group G has fixpoint free elements.

Proof G has a single orbit and $\mathrm{fix}(g) = n$ for the identity. Since the expectation is 1 there are g such that $\mathrm{fix}(g) = 0$.

Fixpoint free elements in G

Theorem (Jordan's Theorem)

If $G \leq S_n$ is transitive then the group G has fixpoint free elements.

Proof G has a single orbit and $\mathrm{fix}(g) = n$ for the identity. Since the expectation is 1 there are g such that $\mathrm{fix}(g) = 0$.

Problem: Given $G = \langle g_1, g_2, \ldots, g_k \rangle$ transitive, can we find a fixpoint free element in polynomial time?

Cameron-Cohen's Theorem

Theorem (CC92)

If $G \leq S_n$ is transitive then the group G has at least $|G|/n$ many fixpoint elements.

Proof

Cameron-Cohen's Theorem

Theorem (CC92)

If $G \leq S_n$ is transitive then the group G has at least $|G|/n$ many fixpoint elements.

Proof

Let A denote the set of fixpoint free elements in G.

Cameron-Cohen's Theorem

Theorem (CC92)

If $G \leq S_n$ is transitive then the group G has at least $|G|/n$ many fixpoint elements.

Proof

Let A denote the set of fixpoint free elements in G.

$$|G| = \sum_{g \in G} \mathrm{fix}(g) = \sum_{g \in G_1} \mathrm{fix}(g) + \sum_{g \in G \setminus G_1} \mathrm{fix}(g)$$

Cameron-Cohen's Theorem

Theorem (CC92)

If $G \leq S_n$ is transitive then the group G has at least $|G|/n$ many fixpoint elements.

Proof

Let A denote the set of fixpoint free elements in G.

$$
\begin{aligned}
|G| &= \sum_{g \in G} \mathrm{fix}(g) = \sum_{g \in G_1} \mathrm{fix}(g) + \sum_{g \in G \setminus G_1} \mathrm{fix}(g) \\
|G| &\geq |G|/n + |G_1| + |G \setminus (A \cup G_1)|
\end{aligned}
$$

Which yields

$$|A| \geq |G|/n.$$

A Randomized Algorithm

Let $G = \langle S \rangle$ be a permutation group given as input by generating set S.

A Randomized Algorithm

Let $G = \langle S \rangle$ be a permutation group given as input by generating set S.

- We first compute a strong generating set T for G.

A Randomized Algorithm

Let $G = \langle S \rangle$ be a permutation group given as input by generating set S.

- We first compute a strong generating set T for G.

- Using T we can sample $\pi \in G$ uniformly at random, and check if π is fixpoint free.

A Randomized Algorithm

Let $G = \langle S \rangle$ be a permutation group given as input by generating set S.

• We first compute a strong generating set T for G.

• Using T we can sample $\pi \in G$ uniformly at random, and check if π is fixpoint free.

Analysis Probability that we do not find a fixpoint free element in, say, n^2 trials is bounded by $(1 - 1/n)^{n^2} \approx e^{-n}$.

Deterministic Algorithm

- Let $\mathrm{move}(g) = n - \mathrm{fix}(g)$. Orbit counting lemma restated:

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \frac{1}{|G|} \sum_{g \in G} \mathrm{move}(g) = n - \mathrm{orb}(G).$$

Deterministic Algorithm

- Let $\mathrm{move}(g) = n - \mathrm{fix}(g)$. Orbit counting lemma restated:

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \frac{1}{|G|} \sum_{g \in G} \mathrm{move}(g) = n - \mathrm{orb}(G).$$

For G transitive we have $\mathbb{E}_{g \in G}[\mathrm{move}(g)] = n - 1$.

Deterministic Algorithm

- Let $\mathrm{move}(g) = n - \mathrm{fix}(g)$. Orbit counting lemma restated:

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \frac{1}{|G|} \sum_{g \in G} \mathrm{move}(g) = n - \mathrm{orb}(G).$$

For G transitive we have $\mathbb{E}_{g \in G}[\mathrm{move}(g)] = n - 1$.

Since G_1 has at least two orbits,

$$\mathbb{E}_{g \in G_1}[\mathrm{move}(g)] \leq n - 2.$$

Let $G = \biguplus_{\pi \in R} G_1 \pi$.

Deterministic Algorithm

- Let $\mathrm{move}(g) = n - \mathrm{fix}(g)$. Orbit counting lemma restated:

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \frac{1}{|G|} \sum_{g \in G} \mathrm{move}(g) = n - \mathrm{orb}(G).$$

For G transitive we have $\mathbb{E}_{g \in G}[\mathrm{move}(g)] = n - 1$.

Since G_1 has at least two orbits,

$$\mathbb{E}_{g \in G_1}[\mathrm{move}(g)] \leq n - 2.$$

Let $G = \biguplus_{\pi \in R} G_1 \pi$.

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \mathbb{E}_{\pi \in R} \mathbb{E}_{g \in G_1 \pi}[\mathrm{move}(g)].$$

Deterministic Algorithm

- Let $\mathrm{move}(g) = n - \mathrm{fix}(g)$. Orbit counting lemma restated:

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \frac{1}{|G|} \sum_{g \in G} \mathrm{move}(g) = n - \mathrm{orb}(G).$$

For G transitive we have $\mathbb{E}_{g \in G}[\mathrm{move}(g)] = n - 1$.

Since G_1 has at least two orbits,

$$\mathbb{E}_{g \in G_1}[\mathrm{move}(g)] \leq n - 2.$$

Let $G = \biguplus_{\pi \in R} G_1 \pi$.

$$\mathbb{E}_{g \in G}[\mathrm{move}(g)] = \mathbb{E}_{\pi \in R} \mathbb{E}_{g \in G_1 \pi}[\mathrm{move}(g)].$$

- Thus, for some coset $G_1 \pi$ of G_1 in G we must have $\mathbb{E}_{g \in G_1 \pi}[\mathrm{move}(g)] > n - 1$.

Deterministic Algorithm Contd.

• For each coset representative $\pi \in R$ we explain how to efficiently compute $\mathbb{E}_{g \in G_1\pi}[\mathrm{move}(g)]$. Revisit the proof of the orbit counting lemma:
Recall $M_{g\pi,i} = 1$ iff $i^{g\pi} = i$.

Deterministic Algorithm Contd.

• For each coset representative $\pi \in R$ we explain how to efficiently compute $\mathbb{E}_{g \in G_1 \pi}[\mathrm{move}(g)]$. Revisit the proof of the orbit counting lemma:
Recall $M_{g\pi,i} = 1$ iff $i^{g\pi} = i$.

• Number of 1's in the i^{th} column is $|\{g \in G_1 \mid i^{g\pi} = i\}|$.

Deterministic Algorithm Contd.

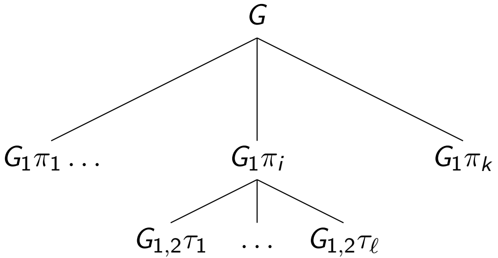
- For each coset representative $\pi \in R$ we explain how to efficiently compute $\mathbb{E}_{g \in G_1 \pi}[\mathrm{move}(g)]$. Revisit the proof of the orbit counting lemma:
Recall $M_{g\pi,i} = 1$ iff $i^{g\pi} = i$.

- Number of 1's in the i^{th} column is $|\{g \in G_1 \mid i^{g\pi} = i\}|$.

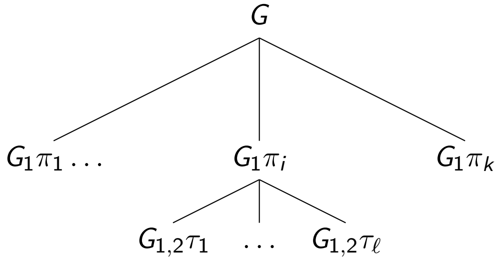
- This number is zero if i and $i^{\pi^{-1}}$ are in different G_1-orbits, and is $|G_{1,i}|$ otherwise.

Thus, using the Schreir-Sims algorithm we can compute all column sums efficiently, and hence also the above expectation in polynomial time.

Method of Conditional Probabilities



Method of Conditional Probabilities



• At the d^{th} level of the tree compute the coset $G_{1,\ldots,d}\sigma$ that maximizes $\mathbb{E}_{g \in G_{1,\ldots,d}\sigma}[\mathrm{move}(g)]$.

Remarks and related questions

• Method of conditional probabilities – Erdös-Selfridge, Spencer, Raghavan.

Remarks and related questions

• Method of conditional probabilities – Erdös-Selfridge, Spencer, Raghavan.

• Fixpoint free element checking in general permutation groups $G = \langle S \rangle \leq S_n$ is NP-complete.

Remarks and related questions

• Method of conditional probabilities – Erdös-Selfridge, Spencer, Raghavan.

• Fixpoint free element checking in general permutation groups $G = \langle S \rangle \leq S_n$ is NP-complete.

• Fein-Kantor-Schacher: Every transitive permutation group $G \leq S_n$ for $n \geq 2$ has a fixpoint free element of prime power order. Is there an efficient algorithm for finding one?

Remarks and related questions

• Method of conditional probabilities – Erdös-Selfridge, Spencer, Raghavan.

• Fixpoint free element checking in general permutation groups $G = \langle S \rangle \leq S_n$ is NP-complete.

• Fein-Kantor-Schacher: Every transitive permutation group $G \leq S_n$ for $n \geq 2$ has a fixpoint free element of prime power order. Is there an efficient algorithm for finding one?

• Isaacs-Kantor-Spaltenstein: For $G \leq S_n$ and prime p dividing $|G|$, there are at least $|G|/n$ many elements whose order is divisible by p. Deterministic algorithm?

Remarks and related questions

• Method of conditional probabilities – Erdös-Selfridge, Spencer, Raghavan.

• Fixpoint free element checking in general permutation groups $G = \langle S \rangle \leq S_n$ is NP-complete.

• Fein-Kantor-Schacher: Every transitive permutation group $G \leq S_n$ for $n \geq 2$ has a fixpoint free element of prime power order. Is there an efficient algorithm for finding one?

• Isaacs-Kantor-Spaltenstein: For $G \leq S_n$ and prime p dividing $|G|$, there are at least $|G|/n$ many elements whose order is divisible by p. Deterministic algorithm?

Bases for Permutation Groups

• Let $G \leq S_n$ be a permutation group. A subset of points $B \subseteq [n]$ is called a *base* for G if the subgroup G_B of G that fixes every point of G is the identity.

Bases for Permutation Groups

• Let $G \leq S_n$ be a permutation group. A subset of points $B \subseteq [n]$ is called a *base* for G if the subgroup G_B of G that fixes every point of G is the identity.

• This generalizes bases for vector spaces and has proven computationally useful. There is a library of nearly linear-time algorithms for small base groups due to Akos Seress and others.

Bases for Permutation Groups

• Let $G \leq S_n$ be a permutation group. A subset of points $B \subseteq [n]$ is called a *base* for G if the subgroup G_B of G that fixes every point of G is the identity.

• This generalizes bases for vector spaces and has proven computationally useful. There is a library of nearly linear-time algorithms for small base groups due to Akos Seress and others.

• Finding minimum bases of permutation groups is NP-hard [Blaha 1992] even for cyclic groups and groups with bounded orbit size.

Bases for Permutation Groups

• Let $G \leq S_n$ be a permutation group. A subset of points $B \subseteq [n]$ is called a *base* for G if the subgroup G_B of G that fixes every point of G is the identity.

• This generalizes bases for vector spaces and has proven computationally useful. There is a library of nearly linear-time algorithms for small base groups due to Akos Seress and others.

• Finding minimum bases of permutation groups is NP-hard [Blaha 1992] even for cyclic groups and groups with bounded orbit size.

Approximating minimum bases

- If $G = \langle S \rangle \leq S_n$ has minimum base size b then $|G| \leq n^b$.

Approximating minimum bases

- If $G = \langle S \rangle \leq S_n$ has minimum base size b then $|G| \leq n^b$.

Proof If $\{1, 2, \ldots, b\}$ is a minimum size base then $|G_{[i-1]}|/|G_{[i]}| \leq n$ and $|G_{[b]}| = 1$.

Approximating minimum bases

• If $G = \langle S \rangle \leq S_n$ has minimum base size b then $|G| \leq n^b$.

Proof If $\{1, 2, \ldots, b\}$ is a minimum size base then $|G_{[i-1]}|/|G_{[i]}| \leq n$ and $|G_{[b]}| = 1$.

• An *irredundant base* $B = \{\alpha_1, \alpha_2, \ldots, \alpha_d\}$ is such that no α_i is fixed by the pointwise stabilizer of all earlier points. Hence $|G| \geq 2^d$.

Approximating minimum bases

• If $G = \langle S \rangle \leq S_n$ has minimum base size b then $|G| \leq n^b$.

Proof If $\{1, 2, \ldots, b\}$ is a minimum size base then $|G_{[i-1]}|/|G_{[i]}| \leq n$ and $|G_{[b]}| = 1$.

• An *irredundant base* $B = \{\alpha_1, \alpha_2, \ldots, \alpha_d\}$ is such that no α_i is fixed by the pointwise stabilizer of all earlier points. Hence $|G| \geq 2^d$.

• Thus, any irredudant base is of size $d \leq b \log n$.

A Better Approximation Algorithm

- Having picked $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$, pick α_{i+1} from an orbit of largest size in the pointwise stabilizer of $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$.

A Better Approximation Algorithm

- Having picked $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$, pick α_{i+1} from an orbit of largest size in the pointwise stabilizer of $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$.

Claim This yields a base of size $(\log \log n + O(1))b$.

A Better Approximation Algorithm

• Having picked $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$, pick α_{i+1} from an orbit of largest size in the pointwise stabilizer of $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$.

Claim This yields a base of size $(\log \log n + O(1))b$.

If $H \leq G$ then H has an orbit of size at least $|H|^{1/b}$. Fixing a point in it makes $|H_\alpha| \leq |H|^{1/b}$.

A Better Approximation Algorithm

• Having picked $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$, pick α_{i+1} from an orbit of largest size in the pointwise stabilizer of $\{\alpha_1, \alpha_2, \ldots, \alpha_i\}$.

Claim This yields a base of size $(\log \log n + O(1))b$.

If $H \leq G$ then H has an orbit of size at least $|H|^{1/b}$. Fixing a point in it makes $|H_\alpha| \leq |H|^{1/b}$.

• Thus, by picking $b \log \log n$ points $|G$ shrinks as $|G|^{(1-1/b)^{b \log \log n}} \approx e^b$. Pick $O(b)$ more points irredundantly.

Other Finite Groups

• Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$, where q is a prime power and $\mathbb{F}_q$ is the finite field of size q.

Other Finite Groups

- Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$, where q is a prime power and $\mathbb{F}_q$ is the finite field of size q.

- Membership testing is likely to be hard.

Other Finite Groups

• Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$, where q is a prime power and $\mathbb{F}_q$ is the finite field of size q.

• Membership testing is likely to be hard.

• Given $a, b \in \mathbb{F}_q^{\times}$, checking if $a^x = b (\mathrm{mod} p)$ is considered a computationally hard problem. Finding x is the so-called *discrete log* problem.

Other Finite Groups

• Let $G \leq \mathrm{GL}_n(\mathbb{F}_q)$, where q is a prime power and $\mathbb{F}_q$ is the finite field of size q.

• Membership testing is likely to be hard.

• Given $a, b \in \mathbb{F}_q^\times$, checking if $a^x = b (\mathrm{mod} p)$ is considered a computationally hard problem. Finding x is the so-called *discrete log* problem.

• We cannot embed $\mathbb{F}_q^\times$ in S_n for small n if $q - 1$ has "large" prime factors.

References/further reading

- Peter J Cameron “Permutation Groups”, LMS Student Texts 45, Cambridge Univ Press.

- Eugene M Luks “Permutation groups and polynomial-time computation”, in Groups and Computation, DIMACS series in Discrete Mathematics and Theoretical Computer Science 11 (1993), 139-175. Available online.

- Laszlo Babai “Local expansion of vertex-transitive graphs and random generation in finite groups”.

Other Finite Groups Contd.

Suppose $G = \langle g_1, g_2, \ldots, g_k \rangle$ where we assume *no structure* about G. What can we compute efficiently? Randomness helps.

- *Random subproducts* of $g_1, g_2, \ldots, g_k$ are elements of the form

$$g_1^{\epsilon_1} g_2^{\epsilon_2} \ldots g_k^{\epsilon_k}, \epsilon_i \in_R \{0, 1\}.$$

Testing Commutativity

Input $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Testing Commutativity

Input $G = \langle g_1, g_2, \ldots, g_k \rangle$.

• Check if $g_i g_j = g_j g_i$ for all pairs i, j. This is an $O(k^2)$ test, and the best possible deterministic test.

Testing Commutativity

Input $G = \langle g_1, g_2, \ldots, g_k \rangle$.

• Check if $g_i g_j = g_j g_i$ for all pairs i, j. This is an $O(k^2)$ test, and the best possible deterministic test.

A randomized test
Let $x = g_1^{\epsilon_1} g_2^{\epsilon_2} \ldots g_k^{\epsilon_k}$ and $y = g_1^{\mu_1} g_2^{\mu_2} \ldots g_k^{\mu_k}$ be two independent subproducts.
Accept iff $xy = yx$.

Testing Commutativity Contd.

- **Claim** If $H < G$ is a proper subgroup then

$$\mathrm{Prob}[g_1^{\epsilon_1} g_2^{\epsilon_2} \ldots g_k^{\epsilon_k} \notin H] \geq 1/2.$$

Testing Commutativity Contd.

- **Claim** If $H < G$ is a proper subgroup then

$$\mathrm{Prob}[g_1^{\epsilon_1} g_2^{\epsilon_2} \dots g_k^{\epsilon_k} \notin H] \geq 1/2.$$

$$\begin{aligned}\mathrm{Prob}[xy = yx] &\leq \mathrm{Prob}[x \in Z(G)] + \mathrm{Prob}[y \in C(x) \wedge x \notin Z(G)] \\ &\leq p + (1-p)/2 \\ &= (1+p)/2 \\ &\leq 3/4\end{aligned}$$

Testing Commutativity Contd.

- **Claim** If $H < G$ is a proper subgroup then

$$\mathrm{Prob}[g_1^{\epsilon_1} g_2^{\epsilon_2} \dots g_k^{\epsilon_k} \notin H] \geq 1/2.$$

$$\begin{aligned} \mathrm{Prob}[xy = yx] &\leq \mathrm{Prob}[x \in Z(G)] + \mathrm{Prob}[y \in C(x) \wedge x \notin Z(G)] \\ &\leq p + (1-p)/2 \\ &= (1+p)/2 \\ &\leq 3/4 \end{aligned}$$

Curious Fact If G is nonabelian then

$$\mathrm{Prob}_{x,y \in G}[xy = yx] \leq 5/8.$$

Erdös-Rényi Sequences

- Random subproducts come from an Erdös-Rényi paper titled "Probabilistic methods in group theory".

Erdös-Rényi Sequences

- Random subproducts come from an Erdös-Rényi paper titled "Probabilistic methods in group theory".

A random subproduct $g_1^{e_1} g_2^{e_2} \ldots g_k^{e_k}$ is ε-uniform in G if for all $x \in G$

$$(1 - \varepsilon)/|G| \leq \mathrm{Prob}[g_1^{e_1} g_2^{e_2} \ldots g_k^{e_k} = x \leq (1 + \varepsilon)/|G|.$$

If $k \geq 2 \log |G| + 2 \log(1/\varepsilon) + \log(1/\delta)$ and $g_1, g_2, \ldots, g_k$ are randomly picked, then $g_1^{e_1} g_2^{e_2} \ldots g_k^{e_k}$ is ε-uniform in G with probability $1 - \delta$.

Straight-line programs for G

- $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Straight-line programs for G

- $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Define the cube $C = \{g_1^{e_1} \ldots g_k^{e_k} \mid e_i \in \{0,1\}\}$ and C^{-1} be the inverses of the elements in C.

Straight-line programs for G

- $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Define the cube $C = \{g_1^{e_1} \ldots g_k^{e_k} \mid e_i \in \{0,1\}\}$ and C^{-1} be the inverses of the elements in C.

If $G = C^{-1}C$, we have short st-line programs for all of G.

Straight-line programs for G

- $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Define the cube $C = \{g_1^{e_1} \ldots g_k^{e_k} \mid e_i \in \{0, 1\}\}$ and C^{-1} be the inverses of the elements in C.

If $G = C^{-1}C$, we have short st-line programs for all of G.

Otherwise, there is a generator g_j such that

$$C^{-1}Cg_j \not\subset C^{-1}C.$$

Straight-line programs for G

- $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Define the cube $C = \{g_1^{e_1} \ldots g_k^{e_k} \mid e_i \in \{0,1\}\}$ and C^{-1} be the inverses of the elements in C.

If $G = C^{-1}C$, we have short st-line programs for all of G.

Otherwise, there is a generator g_j such that

$$C^{-1}Cg_j \not\subset C^{-1}C.$$

Include an element $g_{k+1} \in C^{-1}Cg_j \setminus C^{-1}C$ to extend the sequence.

Straight-line programs for G

- $G = \langle g_1, g_2, \ldots, g_k \rangle$.

Define the cube $C = \{g_1^{e_1} \ldots g_k^{e_k} \mid e_i \in \{0, 1\}\}$ and C^{-1} be the inverses of the elements in C.

If $G = C^{-1}C$, we have short st-line programs for all of G.

Otherwise, there is a generator g_j such that

$$C^{-1}Cg_j \not\subset C^{-1}C.$$

Include an element $g_{k+1} \in C^{-1}Cg_j \setminus C^{-1}C$ to extend the sequence. As $Cg_{k+1} \cap C = \emptyset$ we have doubled the size of the cube.

Short straight-line programs

In $\ell \leq \log |G|$ steps we obtain elements $g_{k+1}, g_{k+2}, \ldots, g_{k+\ell} \in G$ such that for the cube

$$C_\ell = \{g_1^{e_1} \ldots g_{k+\ell}^{e_{k+\ell}} \mid e_i \in \{0,1\}\}.$$

we have

$$G = C_\ell^{-1} C_\ell.$$

Short straight-line programs

In $\ell \leq \log|G|$ steps we obtain elements $g_{k+1}, g_{k+2}, \ldots, g_{k+\ell} \in G$ such that for the cube

$$C_\ell = \{g_1^{e_1} \ldots g_{k+\ell}^{e_{k+\ell}} \mid e_i \in \{0,1\}\}.$$

we have

$$G = C_\ell^{-1} C_\ell.$$

Each $g \in G$ has a straight-line program of length

$$\sum_{i=0}^{\log|G|} (2(k+i)+1) + 2(k+\log|G|) = O((k+\log|G|)\log|G|)$$

in terms of the original generators.

THANKS!