Algorithmic Abstract Algebra 1

Peter Mayr

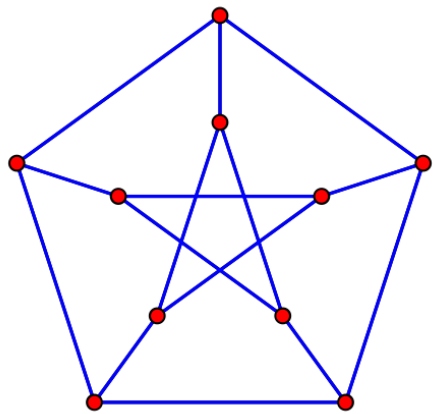
Institute for Algebra, JKU
peter.mayr@jku.at

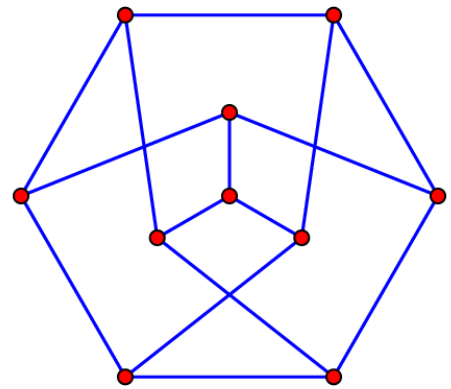
Linz, October, 2013

JOHANNES KEPLER
UNIVERSITY LINZ | JKU

FWF Der Wissenschaftsfonds.

1. Can one twist a single corner of Rubik's cube?
2. When do 2 sequences of moves yield the same result?
3. Are the following graphs isomorphic?





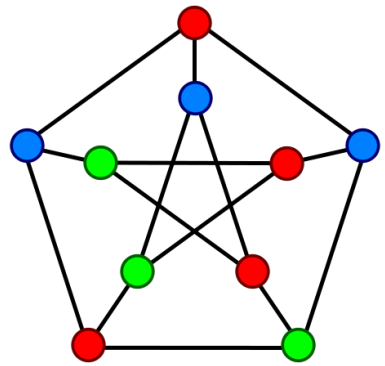
4. What are the possible symmetry groups of crystals?

These questions lead to problems in **Computational Group Theory** (Lectures 1 and 2).

1. SAT: Is a given propositional formula in conjunctive normal form satisfiable?

$$F = (x \vee y' \vee z) \wedge (x \vee y' \vee z') \wedge (x' \vee y \vee z).$$

2. Graph coloring: Can we color the vertices of a given graph with 3 colors so that no adjacent vertices have the same color?



3. Linear equations: Does a given system of linear equations have a solution?
4. Sudoku, Database queries, Scheduling problems, ...

All these questions can be expressed naturally as **Constraint Satisfaction Problems (CSP)** (Lectures 3 and 4).

An Introduction to Computational Group Theory

Overview

1. Group actions
   - 1.1 Orbits, stabilizers, representatives
   - 1.2 Random elements
2. Permutation groups
   - 2.1 Stabilizer chains
   - 2.2 Sifting
   - 2.3 Schreier-Sims algorithm
   - 2.4 Backtrack

References:

1. Alexander Hulpke. Notes on Computational Group Theory, 2010. Available from `http://www.math.colostate.edu/~hulpke/`
2. Derek Holt, Bettina Eick, Eamonn O'Brien. Handbook of Computational Group Theory, 2005
3. Ákos Seress. Permutation Group Algorithms, 2003.

The following slides are mainly based on 1.

On the computer

Computer algebra systems for groups:
GAP (University of St. Andrews), Magma (University of Sydney)

Representations and memory usage for GAP on a 32-bit system:

1. **Small number:** $|n| < 2^{28}$ uses 32 bits — 4 bytes
2. **Permutation:** stored as list of images for points $[1..n]$ — $4n$ bytes

A permutation of degree 100.000 uses 400 KB.
2 GB memory are filled by 5000 permutations.
$S_{100.000}$ has $10^5!$ elements and is generated by 2.

Remark

For groups of large degree, we can only store few elements.
We need algorithms to work with generators of groups and avoid enumerating elements.

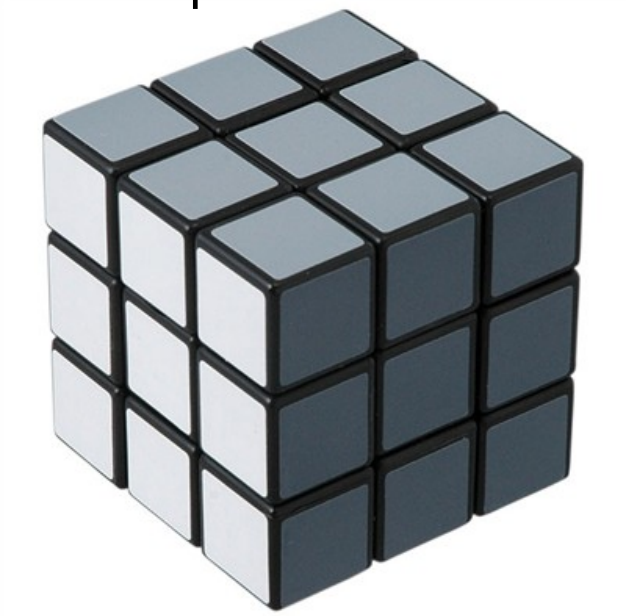
Exercise

Show that every finite group G can be generated by $\leq \log_2(|G|)$ elements.

Making the cube into a group

Name the 6 faces of Rubik's cube: front, back, up, down, left, right.

up

front  right

Call f the rotation by 90° counter clockwise of the front-face, etc. Rotations u, d, l, r, f, b generate a group that acts on positions and orientation of the 8 corner and 12 edge "cubies".

Exercise

Give an upper bound for the number of possible configurations of the cube.

1. Group actions

Group action

A group G **acts** on a set Ω if we have a function $\Omega \times G \to \Omega, (\omega, g) \mapsto \omega^g$, such that

1. $\omega^1 = \omega \ \forall \omega \in \Omega$,
2. $(\omega^g)^h = \omega^{gh} \ \forall g, h \in G$.

Convention

Groups will always act on the right.

1. The symmetric group S_3 acts on $\Omega = \{1, 2, 3\}$, e.g., $1^{(1,2)} = 2, 3^{(1,2)} = 3$.
   So $(1, 2)(2, 3) = (1, 3, 2)$.
2. A group G acts on itself by right multiplication, $(x, g) \mapsto xg$.
3. A matrix group acts on a vector space by right multiplication.
4. G acts on itself by conjugation, $(x, g) \mapsto x^g$ where $x^g = g^{-1}xg$.

Orbits and stabilizers

For G acting on Ω and $\omega \in \Omega$ we define

1. the **orbit** $\omega^G := \{\omega^g \mid g \in G\}$ of ω under G,
2. the **stabilizer** $\mathrm{Stab}_G(\omega) := \{g \in G \mid \omega^g = \omega\}$.

Note that stabilizers are subgroups, $\mathrm{Stab}_G(\omega) \leq G$.

$1^{S_3} = \{1, 2, 3\}, \mathrm{Stab}_{S_3}(1) = \{(), (2, 3)\}$.

Lemma

$$\begin{aligned} f : \quad \omega^G &\to \mathrm{Stab}_G(\omega)\backslash G \\ \omega^g &\mapsto \mathrm{Stab}_G(\omega) \cdot g \end{aligned}$$

is a bijection between ω^G and the set of **right cosets** of $\mathrm{Stab}_G(\omega)$. So $|\omega^G| = |G : \mathrm{Stab}_G(\omega)|$, the **index** of the stabilizer in G.

Computing orbits

TFAE:

1. One can twist a single corner of Rubik's cube.
2. The configuration with one twisted corner is in the orbit of the standard configuration under the group $G := \langle u, d, l, r, f, b \rangle$ generated by rotations.

So to answer (1), we want to compute orbits.

Convention

If G is infinite, then the generating set of G will always be closed under inverses.

Then every $g \in G$ is a product of generators of G.

"Plain vanilla" orbit algorithm

Orbit algorithm

Input : $G = \langle g_1, \ldots, g_m \rangle$ acting on Ω, $\omega \in \Omega$
Output: ω^G

```
1  Δ := [ω];
2  for δ ∈ Δ do
3      for i ∈ {1, ..., m} do
4          γ := δ^{g_i} ;
5          if γ ∉ Δ then
6              Append γ to Δ;
7          end
8      end
9  end
10 return Δ
```

Correctness of the orbit algorithm

1. $\Delta \subseteq \omega^G$ by construction (lines 4,6).
2. When returned, Δ is closed under G, hence a union of orbits.
3. So $\Delta = \omega^G$.

Performance

1. For m generators and $n := |\omega^G|$, we need to compute mn images δ^g.
2. Checking $\gamma \in \Delta$ is a search problem: $\rightarrow O(\log(n))$ by binary search in sorted list.

Remark

All elements of G can be obtained as the orbit of 1 under right multiplication.

Representatives

For solving Rubik's cube, we not only want to know whether some configuration is possible, but how to obtain it from the standard configuration (or conversely).

Problem

Given $\delta \in \omega^G$, find $g \in G : \omega^g = \delta$.

Equivalently: Find a set of representatives (a **transversal**) for the right cosets of $\mathrm{Stab}_G(\omega)$ in G.

Consider $\Delta = \omega^G$ as list with fixed order of elements.
$T[\delta] := T[i]$ where $\Delta[i] = \delta$.

Orbit algorithm with transversal

Input : $G = \langle g_1, \ldots, g_m \rangle$ acting on Ω, $\omega \in \Omega$
Output: ω^G and transversal for $\mathrm{Stab}_G(\omega)\backslash G$

```
1  Δ := [ω];
2  T := [1];
3  for δ ∈ Δ do
4      for i ∈ {1, ..., m} do
5          γ := δ^{g_i};
6          if γ ∉ Δ then
7              Append γ to Δ;
8              Append T[δ] · g_i to T;
9          end
10     end
11 end
12 return Δ, T
```

Remark

- $T[\delta]$ is a **shortest** product of generators such that $\omega^{T[\delta]} = \delta$.
- So we have a minimal factorization of elements in G via brute-force enumeration.
- In general, finding minimal factorizations requires some kind of enumeration.
- See Rokicky, et al (2010). God's number is 20.
  `http://www.cube20.org`
  Every configuration of Rubik's cube can be solved by 20 moves in the half turn metric (35 CPU years, heavy parallel computing on Google's computers). Configurations requiring 20 moves were known before

Exercise

Consider $S_{100} = \langle (1, 2, \ldots, 100), (1, 2) \rangle$. What representatives do you get by computing the orbit of 1?
Give 3 generators of S_{100} that yield shorter words for the representatives.

Schreier vectors

Recall:

Permutations are big (n times the size of an integer for $n = |\omega^G|$).
Storing one group element per orbit element needs too much space.

A **Schreier vector** for $\Delta = \omega^G$ (considered as list) is a list of length $|\Delta|$ such that

1. entries of S are (pointers to) generators of G or 1,
2. $S[\omega] = 1$, and
3. if $S[\delta] = g$ and $\gamma = \delta^g$, then δ comes before γ in Δ.

Orbit algorithm with Schreier vectors

Adapt the standard orbit algorithm by:

1. Initializing $S := [1]$.
2. If $\gamma = \delta^{g_i}$ is a new point, append (the pointer to) g_i to S.

Schreier vectors replace transversals

Representative from Schreier vector

Input : Schreier vector S for ω, $\gamma \in \omega^G$
Output: r such that $\omega^r = \gamma$

```
1 δ := γ;
2 r = 1;
3 while δ ≠ ω do
4     g := S[δ];
5     r := g · r;
6     δ := δ^(g^-1);
7 end
8 return r
```

Correctness

The algorithm terminates by condition 3 of the definition of a Schreier vector. After each step $\delta^r = \gamma$.

Finding generators for stabilizers

Given generators of a group G and coset representatives for a subgroup S, the next result yields generators for S.

Schreier's Subgroup Lemma

Let G be a group that is generated by a finite set X, and let $S \leq G$ with $|G : S| < \infty$. Let T be a transversal for S in G with $1 \in T$. For $g \in G$, let $\bar{g} := r$ such that $Sr = Sg$. Then

$$S = \langle rx(\overline{rx})^{-1} \mid r \in T, x \in X \rangle.$$

(These generators of S are called **Schreier generators**.)

Corollary

In a finitely generated group, every subgroup of finite index is finitely generated.

Orbit & stabilizer algorithm

Input : $G = \langle g_1, \ldots, g_m \rangle$ acting on Ω, $\omega \in \Omega$
Output: ω^G, transversal T, stabilizer $S := \mathrm{Stab}_G(\omega)$

1 $\Delta := [\omega]$; $T := [1]$; $S := \langle 1 \rangle$;
2 **for** $\delta \in \Delta$ **do**
3 **for** $i \in \{1, \ldots, m\}$ **do**
4 $\gamma := \delta^{g_i}$;
5 **if** $\gamma \notin \Delta$ **then**
6 Append γ to Δ;
7 Append $T[\delta] \cdot g_i$ to T;
8 **else**
9 $S := \langle S, T[\delta] \cdot g_i \cdot T[\gamma]^{-1} \rangle$;
10 **end**
11 **end**
12 **end**
13 **return** Δ, T, S

Performance

If $|\omega^G| = n$, the algorithm computes mn images of which only $n-1$ are new. Thus we have

$$mn - (n-1) = |G : S|(m-1) + 1$$

Schreier generators for S (**linear in index**).
Typically these generators are highly redundant. Two ways to deal with that are:

1. In step 9, test whether the new Schreier generator is already contained in S (Requires **many element tests**).
2. Pick a small (random) subset of the Schreier generators or their products and hope that they already generate $\mathrm{Stab}_G(\omega)$ (Requires analysis of **generation probability** (Babai, Luks, Seress, 1997) and means for **verification**).

Application: Normal Closure

Let $U \leq G$. The **normal closure** $\langle U \rangle_G$ of U in G is the smallest normal subgroup of G containing U.

If $G = \langle g_1, \ldots, g_m \rangle$, then for its **derived subgroup** G' we have

$$G' = \langle [g_i, g_j] \mid 1 \leq i, j \leq m \rangle_G.$$

A variation of the orbit algorithm yields generators for the normal closure.

Algorithm for normal closure of a subgroup

Input : $G = \langle g_1, \ldots, g_m \rangle$, $U = \langle u_1, \ldots, u_k \rangle \leq G$
Output: generators for the normal closure $\langle U \rangle_G$

```
1  N := [u_1, ..., u_k];
2  for d ∈ N do
3      for i ∈ {1, ..., m} do
4          c := d^{g_i};
5          if c ∉ ⟨N⟩ then
6              Append c to N;
7          end
8      end
9  end
10 return N
```

Correctness

1. Algorithm terminates in finitely many steps on finite groups.
2. $U \leq \langle N \rangle$ since N is initialized by $u_1, \ldots, u_k$.
3. $\langle N \rangle \leq \langle U \rangle_G$ because only conjugates of $u_1, \ldots, u_k$ are added to N.
4. When N is returned, then for all $x \in N$ and $i \in \{1, \ldots, m\}$: $x^{g_i} \in \langle N \rangle$. Hence $\langle N \rangle$ is normal in G.
5. Since $\langle U \rangle_G$ is the smallest normal subgroup containing U, we have $\langle N \rangle = \langle U \rangle_G$.

Summary

1. Orbits can be computed essentially at a cost linear to their length.
2. Instead of listing and storing all elements of an orbit, one representative suffices.
   E.g., instead of computing all subgroups of a given group, find one representative for each conjugacy class. (Here stabilizers are normalizers of subgroups.)

Random elements

Problem

Given a group G by generators $g_1, \ldots, g_m$, generate (pseudo)-random elements of G (under uniform distribution).

First try:

Assume we have a random number generators. Multiplying random generators, gives elements whose word length grows very slowly.

Instead we give a product-replacement algorithm that was proposed by Leedham-Green, et al (1995), that takes iterated products and behaves well in practice.

The following algorithm consists of an initialization of a global list X of group elements and one more element a. Then a subroutine PseudoRandom2 is iterated and returns a pseudo-random group element.

PseudoRandom

Input : $g_1, \ldots, g_m$

Output: pseudo-random element of $\langle g_1, \ldots, g_m \rangle$

```
1 r := max(11, m) ;                          /* r > 10 is heuristic */
2 X := [g_1, ..., g_m, g_1, ..., g_m, ...] of length r;
3 a := 1;
4 for i ∈ [1..50] ;                          /* r > 50 is heuristic */
5 do
6     PseudoRandom2();
7 end
8 return a
```

PseudoRandom2

```
1  s :=Random([1..r]);          /* 2 random elements from X */
2  t :=Random([1..r] − [s]);
3  e :=Random([−1, 1]);                  /* random exponent */
4  if Random([1, 2]) = 1;              /* random product order */
5  then
6      X[s] := X[s]X[t]^e;
7      a := aX[s];
8  else
9      X[s] := X[t]^e X[s];
10     a := X[s]a;
11 end
12 return a
```