Computing with group orbits

Alexander Hulpke

Department of Mathematics
Colorado State University
Fort Collins, CO, 80523-1874

Polyhedral Computation, Montréal, 10/06

GAP

GAP `http://www.gap-system.org` is a free and open system for discrete mathematics, in particular group theory and related areas.

- Comfortable programming environment for mathematics
- Reasonably efficient implementations of many basic algorithms.
- Many packages provide extensions.

Ask me for more details.

If you are using GAP and ran into problems talk to me!

System vs. Standalone

A general system will never be able to beat a dedicated standalone on its game.
However:

- Programming is much faster if one does not need to rebuild from scratch
- Without availability of high-level routines many projects are just infeasible
- One can still call a standalone for brute-force processing

Potential difficulties of a system

- Many audiences (learners to experts, research in this area to utility use)
- Routines optimized for frequent use in an inner loop might not perform well on single large calculations (and vice versa).
- Authors might have implicitly assumed different uses.

Orbits

We have a (finite) group G, acting on a (finite) set Ω (from the right). I.e.

$$\begin{aligned} \omega^{1_G} &= \omega & \forall \omega \in \Omega \\ \omega^{gh} &= (\omega^g)^h & \forall \omega \in \Omega, g, h \in G \end{aligned}$$

We can therefore define the orbit of ω under G as the set of all images $\omega^G = \{\omega^g \mid g \in G\}$.
Similarly we define the Stabilizer of ω as
$\mathrm{Stab}_G(\omega) = \{g \in G \mid \omega^g = \omega\}$.
There is a bijection between ω^G and the right cosets $\mathrm{Stab}_G(\omega)\backslash G$ given by $\omega^g \leftrightarrow \mathrm{Stab}_G(\omega) \cdot g$. In particular $\left|\omega^G\right| = \left[G{:}\mathrm{Stab}_G(\omega)\right]$.

If G acts on Ω, we get an homomorphism $\varphi\colon G \to S_{|\Omega|}$, we call this the action homomorphism.

Tasks

We want to calculate:

- Orbits and stabilizers for particular points
- Representatives of all orbits
- Test whether two points $\omega_1, \omega_2 \in \Omega$ are in the same orbit and if so find a representative $g \in G$ such that $\omega_1^g = \omega_2$.
- Test whether an element is "minimal" in its orbit.

We shall assume that we have:

- The group G given by a generating set $\underline{\mathbf{g}} = \{g_1, \ldots, g_m\}$.
- A way of comparing (and remembering) orbit elements.
- A function that evaluates images ω^g.

Initial Reductions

If we act on sequences of numbers one can often reduce the computation — iterating through the entries of the sequence — to subsequent orbits under smaller subgroups.

If Ω consists of sets or sequences of objects we can enumerate these objects and thus get sets or sequences of numbers which are much more efficient to work with.
For example an element stabilizer then just becomes a point stabilizer and standard permutation group algorithms compute it quickly.

In particular there are (backtrack) routines to calculate:

- The stabilizer of a set under a permutation group
- An element $g \in G$ mapping one set of points to another (if such an element exists)
- Intersection of subgroups – often a stabilizer can be written down immediately in the symmetric group.
- `SmallestImageSet` in the GRAPE package.

The orbit algorithm

As every $g \in G$ is a product of generators (also inverses if $|G| = \infty$) we get every element of ω^G by iterative application of generators.

Input: $G = \langle gesyg \rangle$ with $\underline{\mathbf{g}} = \{g_1, \ldots, g_m\}$, also a point $\omega \in \Omega$
Output: return the orbit ω^G.

begin

```
Δ := [ω]
for δ ∈ Δ do
    for i ∈ {1, ..., m} do
        γ := δ^{g_i}
        if γ ∉ Δ then
            Append γ to Δ.
        fi;
    od;
od;
return Δ
```

end

Representatives

We keep a *transversal* T such that for every $\delta in \omega^G$ we have that $\omega^{T[\delta]} = \delta$.
This list gets initialized with $T[\omega] := 1_G$, whenever a new point $\delta = \gamma^{g_i}$ arises we store $T[\delta] := T[\gamma] \cdot g_i$.
Then for $\gamma, \rho \in \omega^G$ we have that $T[\gamma]^{-1} \cdot T[\rho]$ maps γ to ρ.

As storing all these group elements $T[\delta]$ is memory intensive, one can trade runtime for memory by storing a *factored transversal* or *Schreier Vector*:
Store for a new $\delta = \gamma^{g_i}$ the generator index i for δ. To uncover $T[\delta]$ one would calculate $\gamma = \delta^{g_i^{-1}}$ and then use recursively that $T[\delta] = T[\gamma] \cdot g_i$.

Stabilizers

Lemma

(Schreier) Let $G = \langle \underline{\mathbf{g}} \rangle$ finitely generated acting on Ω, $\left|\omega^G\right| < \infty$ and $\underline{\mathbf{r}} = \{r_\omega = 1, \ldots, r_{\omega_\infty}\}$ representatives for ω^G. For $h \in G$ let $\bar{h} \in \underline{\mathbf{r}}$ such that $\omega^{\bar{h}} = \omega^h$.

$$U := \{r_\gamma g_j (\overline{r_\gamma g_j})^{-1} \mid r_\gamma \in \underline{\mathbf{r}}, g_j \in \underline{\mathbf{g}}\}$$

Then $S = \langle U \rangle$.

$$\begin{tikzcd} \omega \arrow[r, "r_\gamma"'] \arrow[rr, bend left, "r_\delta"] & \gamma \arrow[r, "g_j"'] & \delta \end{tikzcd}$$

The transversal T can serve as representatives $\underline{\mathbf{r}}$.

Using these algorithms

Except from a few particular cases (e.g. permutation groups on sets or sequences of points) any orbit-type calculation in GAP goes through such a calculation.

- `Orbit(G,pt,actfun);`
- `Stabilizer(G,pt,actfun);`
- `RepresentativeAction(G,pt1,pt2,actfun);`
- `OrbitsDomain(G,Omega,actfun);` (just calculates orbits one by one)

At the moment Schreier trees are not used here.

Possible problems/optimizations

The cost of the orbit algorithm is $\left|\omega^{\mathcal{G}}\right| \cdot \left|\underline{\mathbf{g}}\right|$ image computations.

Furthermore each of the $\left|\omega^{\mathcal{G}}\right| \cdot (\left|\underline{\mathbf{g}}\right| - 1)$ redundant images yields a Schreier generator.

(There is an elaborate theory about using small randomized sets of Schreier generators. At the moment `Orbit` etc. do not use this but instead can eliminate redundant generators by doing element tests in the stabilizer generated so far.)

Generator Numbers

From the cost estimate it is clear that reducing the number of generators of G is the easiest way to get a substantial speedup.

On the other hand – even when skipping redundant Schreier generators – a stabilizer calculation (and thus many routines relying on it) produces generating sets that tend to be too large.

If we know that an orbit will be large it thus is beneficial to reduce the number of generators. A quick heuristic is given by `SmallGeneratingSet`.

One can provide a particular generating set to the orbit algorithm in the form

```
Orbit(G,dom,pt,gens,gens,act);
```

(More about the double given generators later.)

Stabilizer order

If computing the order of a (sub)group is reasonably cheap we can compare the order of the stabilizer so far with the orbit length so far.

If the product is the group order $\left|\omega^G\right| \cdot |\mathrm{Stab}_G(\omega)| = |G|$ one can stop.

Using divisibility arguments it even can be possible to certify the stabilizer after only part of the orbit has been found.

Similarly element tests in a partial stabilizer require subgroup operations equivalent to subgroup order.

The computation of subgroup orders is "cheap" for permutation groups and so-called `PcGroups` (which must be solvable). For any other group there can be substantial cost.

Action via a homomorphism

If the acting group does not have such a nice representation a remedy for this is to have two groups: A group H in a good representation and a homomorphic image G that actually acts (with a homomorphism $\varphi\colon H \to G$).

(Similarly computing the image under an action can be cheaper with generators in a homomorphic image.)

We then act with G but form Schreier generators in H. For this we just need once to determine generators $\underline{\mathbf{h}}$ for H and their images $\underline{\mathbf{h}}^{\varphi} \subset G$.

In GAP one can provide such generator systems as further arguments:

```
Orbit(G,dom,pt,gens,genimages,act);
```

Dictionaries

The core operation of the orbit algorithm is to test whether a new image γ is already in the orbit (and if yes, to determine the corresponding transversal element).

Depending on the kind of objects in Ω, one could use (sorted) lists, numbers derived from the object (e.g. vectors over finite fields) or hashing.

To separate this mechanism from the actual orbit algorithm, GAP uses a data structure called a dictionary.

At the start of the orbit algorithm, the function `NewDictionary` is called to create such a dictionary. One can help this function by also providing the domain to the orbit algorithm:

```
Orbit(G,dom,pt,act);
```

Imprimitivity

The action of G on a single orbit Δ is imprimitive if there is a nontrivial partition of Δ (a block system) which is invariant under G. If we know a partition of Ω that yields such block systems on every orbit — for example by transitioning to a projective action on a vector space — we can first compute orbits on blocks and then expand blocks to points.
As for $\omega \in \Delta \subset \Omega$ we have that $\mathrm{Stab}_G(\omega) < \mathrm{Stab}_G(\Delta) < G$ we get effectively a logarithmizing effect on orbit and stabilizer calculations.

Vice versa every subgroup $\mathrm{Stab}_G(\omega) < U < G$ yields such a block system, so this is the generic case of intermediate subgroups.

If you use orderly generation try to make your order compatible with possible imprimitivity.

Normal Subgroups

A prominent space of such a situation is that the orbits of a normal subgroup $N \lhd G$ always form a block system.
Thus if we know a normal subgroup N we could first compute an orbit under N and then simply take images of this orbit under representatives of G/N.
If we know the stabilizer in N, we need to add only one Schreier generator when orbit images coincide.

(This does not work if N would be just a subgroup.)

This approach is used iteratively for solvable groups but not by default used for arbitrary normal subgroups.

Counting Orbits

The Cauchy-Frobenius-Burnside Lemma states that the number of orbits of G on Ω equals the average number of fixpoints of Elements of G on Ω.

As conjugate elements of G have the same number of fixpoints, it suffices to calculate these counts for representatives of the conjugacy classes.

Often it is possible to calculate these fixpoint numbers cheaply without enumerating Ω (e.g. from dimensions of eigenspaces), thus obtaining a count of the orbits.

Using more group theory

If we know more about the subgroup structure of G we can save in a orbit/stabilizer calculation:

- Replace G by a maximal subgroup $M \leq G$ which is know to contain the stabilizer
- Verify a potentially partial stabilizer by checking that no larger subgroup stabilizes
- If the stabilizer is known use a transversal for the cosets to describe the orbit: `RightTransversal(G,S);` is cheap and does not use much memory.

Orbits under a subgroup

Sometimes we know (or can get cheaply) orbits under a group G (for example a symmetric group) but need orbits under a subgroup $U \leq G$.

Suppose that ω is a representative of a G-orbit. Then this orbit corresponds to $\mathrm{Stab}_G(\omega)\backslash G$ and the U-orbits correspond to the double cosets $\mathrm{Stab}_G(\omega)\backslash G/U$:
Map ω with representatives of the double cosets to obtain representatives for the U-orbits.

Huge orbits

If we want to compute orbits that are far larger than what will fit into memory work often has to go towards heuristics such as:

- Calculate images under random generator products. Coincidences (by the birthday paradox they should occur every $\sqrt{|\omega^G|}$ images) yield Schreier generators. Use these to approximate the stabilizer (and orbit length).
- For each image $\gamma \in \omega^G$ calculate the "minimal" element of γ^U for some smaller subgroup $U \leq G$. Then just store representatives for U-orbits. (However unless $U \lhd G$ one still needs to compute images of all elements of ω^G, not just of the U-representatives.)
- Adapt backtrack searches (that systematically go through group elements).

Some references

- Á. Seress: Permutation Group algorithms, CUP, 2003
- D. Holt: Handbook of Computational Group theory, CRC, 2005
- `http://www.math.colostate.edu/~hulpke`
  My webpages: These slides, also CGT lecture notes at
  `http://www.math.colostate.edu/~hulpke/lectures/m676cgt/index.html`