ORBIT LOOKUP TREES

KIRAN S. KEDLAYA

This document is an extended and streamlined treatment of the method of *orbit lookup trees* first described in [3, Appendix]. This provides a memory-efficient approach for solving certain cases of the *orbit-stabilizer problem* in computational group theory: given a finite group G and a finite set S equipped with a left G-action, compute:

- a set of orbit representatives for the action;
- the stabilizer of each representative;
- a function mapping each element x of S to an element g of G whose action on the corresponding orbit representative yields back x (sometimes called a *transporter*).

We will often consider cases where the set S is too large to instantiate in memory, but the number of orbits is small. In this case, the transporter function should be both compact in memory and efficiently computable.

We will focus on two particular instances of the order-stabilizer problem.

- We start with a left G-action on a set S and consider the induced action on n-element subsets of S for some small n.
- We start with a k-linear left G-action on a k-vector space S, for some finite field k, and consider the induced action on n-dimensional subspaces for some small n.

Our strategy in both cases will be inductive: given the full computation for some n, we transition to $n+1$ via the intermediate action on "flags" consisting of a nested pair of subsets (resp. subspaces). This yields an algorithm whose memory usage scales with the size of S times the number of orbits, without (much) regard as to the size of the target set of the induced action.

Our intended application for this construction is to the tabulation of points on certain group quotients in algebraic geometry over a finite field, notably moduli spaces of curves of low genus. For example, in [2] it was used to identify orbit representatives for the action of $\mathrm{GL}_5(\mathbb{F}_2)$ on 5-dimensional planes in the Plücker embedding of the Grassmannian $\mathrm{Gr}(2,5)$.

1. Spoke functions

Definition 1.1. Let G be a group and let S be a set equipped with a left G-action. We recall some more or less standard definitions.

- For any $x \in S$, the *orbit* $O_x := \{gx : g \in G\}$.
- For any $x \in S$, the *stabilizer* $G_x := \{g \in G : gx = x\}$. The *orbit-stabilizer formula* asserts that $\#G_x \cdot \#O_x = \#G$.
- For any $x \in S, y \in O_x$, the set $\{g \in G : gx = y\}$ is nonempty; any element is called a *transporter* from x to y.

Date: version of 23 Feb 2024.

There is a fairly standard algorithm to compute orbits, stabilizers, and transporters, although we are not aware of standard terminology around it and so we have introduced our own.

Definition 1.2. A *spoke function* for a left G-action on S is a function $h : S \to G$ with the property that $x \mapsto h(x)^{-1}x$ is constant on orbits. This data is equivalent to the choice of one representative x in each orbit and, for each $y \in O_x$, a transporter $h(y)$ from x to y; we refer to x as a *hub* and $h(y)$ as a *spoke* of h.

Algorithm 1.3. *Given a (computable) finite group G and a (computable) left action of G on a finite set S, the following algorithm produces a spoke function h for this action.*

1. *Fix an ordering on S and a generating subset H of G.*
2. *For each $v \in S$ in turn, if $h(v)$ is not yet specified, then set $hv := \mathrm{id}_G$ and $Q := \{v\}$, and repeat the following while Q is nonempty:*
   (a) *Pick any $w \in Q$ and remove w from Q.*
   (b) *For each $g \in H$ for which $x := gw$ has the property that $h(x)$ is unassigned, set $h(x) := gh(w)$ and add x to Q.*

Algorithm 1.4. *Given a (computable) finite group G, a (computable) left action of G on a finite set S, and a spoke function h for this action, for each $v \in S$ the following algorithm computes a generating subset I for G_v.*

1. *Compute the orbit $O_v := \{w \in S : h(w)^{-1}(w) = h(v)^{-1}(v)\}$.*
2. *Fix a generating set H for G and an ordering of $O_v \times H$.*
3. *Initialize $I := \emptyset$.*
4. *While I generates a subgroup of order less than $\#G/\#O_v$, pick a pair $(w, g) \in O_v$ and add $h(g(w))^{-1}gh(w)$ to I.*

Remark 1.5. The runtime of Algorithm 1.3 is linear in $\#H \cdot \#S$, so it is crucial in practice to choose a small generating set H. Fortunately, starting from any generating set, one can quickly construct a short generating set (say of size $O(\log \#G)$) by taking random products [1, Theorem 2.5]. Similar logic can be used to bound the runtime of Algorithm 1.4 assuming a random choice of ordering of $O_v \times H$.

2. Orbit lookup trees

We next recall the definition of an orbit lookup tree from [3, Definition A.2].

Definition 2.1. Let G be a finite group and let S be a finite set equipped with a left G-action. Let F be a subset of the power set of S not containing the empty set (the *forbidden subsets*). We say that a subset of S is *eligible* if it contains no forbidden subset.

For any positive integer n, an *orbit tree* of depth n (for G, S, F) is a rooted tree T_n of depth n with the following properties:

- Each node at depth i is an eligible i-element subset U of S, colored either red or green.
- For $i = 0, \dots, n$, the green nodes at depth i form a set of G-orbit representatives for the eligible i-element subsets of S.
- Red nodes have no children.
- Every green node U at depth $i < n$ has children which form a set of G_U-orbit representatives of the eligible $(i+1)$-element subsets of S containing U (which can be identified with elements of $S \setminus U$).

In particular, each node U admits a unique ordering $x_1, \ldots, x_i$ such that each initial segment of this sequence is also a node; we write $U = [x_1, \ldots, x_i]$ instead of $U = \{x_1, \ldots, x_i\}$ when we need to indicate this choice of ordering.

An *orbit lookup tree* is an orbit tree equipped with the following additional data:

- For each node U, an element $g_U \in G$ such that $g_U^{-1} U$ is a green node.
- For each green node U at depth $i < n$, a spoke function h_U for the action of G_U on the children of U.

Algorithm 2.2. *Given an orbit lookup tree T_n of depth n, for any $i \in \{0, \ldots, n\}$ and any sequence $x_1, \ldots, x_i$ of distinct elements of S, the following recursive algorithm determines whether $\{x_1, \ldots, x_i\}$ is eligible, and if so produces a green node U of T_n and an element $g \in G$ such that $gU = \{x_1, \ldots, x_i\}$.*

(1) *If $i = 0$, return $U := \emptyset$, $g := 1_G$ and stop.*
(2) *If $\{x_1, \ldots, x_{i-1}\}$ is a node of T, let U' be this node and set $g_0 := 1_G$. Otherwise, apply the algorithm to $x_1, \ldots, x_{i-1}$ to obtain a green node U' of T and an element $g_0 \in G$ for which $g_0 U' = \{x_1, \ldots, x_{i-1}\}$. If instead we find that $\{x_1, \ldots, x_{i-1}\}$ is ineligible, report that U is ineligible and stop.*
(3) *Set $y := g_0^{-1} x_i$, $U_1 := U' \cup \{h_{U'}(y)^{-1} y\}$, $g_1 := g_0 h_{U'}(y)$.*
(4) *Set $g_2 := g_{U_1}$ and return $U := g_2^{-1} U_1$, $g := g_1 g_2$. If instead we find that g_{U_1} is undefined, then report that U is ineligible.*

Remark 2.3. When constructing an orbit lookup tree in Algorithm 2.4, we will apply Algorithm 2.2 in a state where $i = n$ and T_{n-1} has been completely computed, but T_n has only been partially completed. In this context, it will still make sense to run steps (1)–(3) of Algorithm 2.2; in particular, the recursive call in step (2) can be executed.

The following construction of orbit lookup trees is a simplified version of [3, Algorithm A.5].

Algorithm 2.4. *Given an orbit lookup tree T_n of depth n, the following algorithm extends T_n to an orbit lookup tree T_{n+1} of depth $n+1$, and in addition computes the stabilizer of each green node U at depth $n+1$.*

(1) *For each green node U at depth n:*
    (a) *Apply Algorithm 1.3 to construct a spoke function h_U for the action of G_U on $S \setminus U$.*
    (b) *For each hub v for h_U, add to T_{n+1} an uncolored node $U \cup \{v\}$ with parent U.*
(2) *For each node U at depth $n+1$ which is not already colored:*
    (a) *For $j = 1, \ldots, n$, let w_j be the transposition $(j(n+1))$ in S_{n+1}. Apply steps (1)–(3) of Algorithm 2.2 (as in Remark 2.3) to the sequence $x_{w_j(1)}, \ldots, x_{w_j(n+1)}$ to find a node U_j and an element $g_j \in G$ such that $g_j U_j = U$, or to detect that U is ineligible.*
    (b) *If U is forbidden, or was detected to be ineligible in the previou step, then color U and each U_j black.*
    (c) *Otherwise:*
        (i) *Color U green and set $g_U = 1$.*
        (ii) *For each set V arising as U_j for some j, pick one such j, color V red, and set $g_V := g_j^{-1}$.*

(iii) *Let U' be the parent of U and write $U = U' \cup \{x_{n+1}\}$. Apply Algorithm 1.4 to the spoke function $h_{U'}$ to compute $G_{U'} \cap G_{x_{n+1}} = G_{U'} \cap G_U$.*
(iv) *Compute G_U as the subgroup of G generated by $G_{U'} \cap G_U$ together with g_j for each $j \in \{1, \dots, n\}$ for which $U_j = U$.*

(3) *Remove all black nodes.*

Proof. Let U be an arbitrary green node at depth $n+1$. Write U as $[x_1, \dots, x_{n+1}]$, so that $U' := \{x_1, \dots, x_n\}$ is the parent of U.

We first observe that U is eligible, thanks to the following points.

- Since T_n is an orbit lookup tree, $U' = U \setminus \{x_{n+1}\}$ contains no forbidden subset.
- By step (2)(a), for $j = 1, \dots, n$, $U \setminus \{x_j\}$ contains no forbidden subset.
- By step (2)(b), U is not forbidden.

We next verify by way of contradiction that there cannot be another green node V in the G-orbit of U. Without loss of generality we may assume that U occurs before V in step (2). Write $V = [y_1, \dots, y_{n+1}]$, let $V' := \{y_1, \dots, y_n\}$ be the parent of V, and choose an element g with $gU = V$. If $g(x_{n+1}) = y_{n+1}$, then $gU' = V'$ which would imply $U' = V'$ because T_n is an orbit lookup tree; then step (1)(a) would force $x_{n+1} = y_{n+1}$. Otherwise, we have $gx_j = y_{n+1}$ for some $j \in \{1, \dots, n\}$. For this j, by step (2)(a), the parent of U_j must equal V'; step (1)(a) would then force $U_j = V$. Step (2)(c)(ii) would then color V red, a contradiction.

We next verify that T_{n+1} is an orbit lookup tree. By step (2)(c)(ii), every red node at depth $n+1$ is in the G-orbit of a green node, and in particular is eligible. By the previous paragraphs, the green nodes at depth $n+1$ form a set of G-orbit representatives for the eligible $(n+1)$-element subsets of S. By step (1), the nodes at depth n have the requisite children.

We finally verify that G_U is computed correctly. The correctness of Algorithm 1.4 ensures that $G_{U'} \cap G_U$ is computed correctly. This intersection is the stabilizer of x_{n+1} for the action of G_U on U. If we interpret this action as a homomorphism $G_U \to S_{n+1}$, then the coset $g_j(G_{U'} \cap G_U)$ maps to the coset $w_j S_n$; step (2)(c)(iv) ensures that the former is included into G_U. $\square$

Remark 2.5. We recall a point from [3, Remark A.4]: if there are no forbidden subsets, then we may use the orbit-stabilizer formula as a consistency check for Algorithm 2.4: the sum of $[G : G_U]$ over green nodes U at depth n should equal $\binom{|S|}{n}$.

Remark 2.6. In [3, Remark A.7], it is suggested that one can further optimize Algorithm 2.2 by reducing the number of values of j used in step (2)(b) to a set of orbit representatives for the action of $G_{\{x_1,\dots,x_n\}}$ on $\{x_1, \dots, x_n\} \cong \{1, \dots, n\}$. There are several obstructions to doing so. One is that we need to check all values of j in order to ensure that U is eligible. Another is that even if no subsets are forbidden, it is not immediately clear how to extend the proof that there is no other green node in the G-orbit of U. Yet another is that we need to make sure we compute enough elements g_j to generate the stabilizer G_U.

3. Linear orbit lookup trees

We now take up the challenge of [3, Remark A.8], to adapt the construction of orbit lookup trees to the context of classifying subspaces of a vector space equipped with a linear group action.

Definition 3.1. Let G be a finite group, let k be a finite field, and let S be a finite-dimensional k-vector space equipped with a k-linear left G-action. For $x_1, \dots, x_n \in S$ linearly independent, let $\langle x_1, \dots, x_n \rangle$ denote the k-linear span of $x_1, \dots, x_n$.

Let F be a subset of the set of nonzero k-vector subspaces of S (the *forbidden subspace*). We say that a subspace of S is *eligible* if it contains no forbidden subspace.

For any positive integer n, a *linear orbit tree* of depth n (for G, S, F) is a rooted tree T_n of depth n with the following properties:

- Each node at depth i is an eligible i-dimensional subspace U of S, colored either red or green.
- For $i = 0, \dots, n$, the green nodes at depth i form a set of G-orbit representatives for the eligible k-element subspaces of S.
- Red nodes have no children.
- Every green node U at depth $i < n$ has children which form a set of G_U-orbit representatives of the $(i+1)$-dimensional subspaces of S containing U (which can be identified with one-dimensional subspaces of S/U).

A *linear orbit lookup tree* is a linear orbit tree equipped with the following additional data:

- For each node U, an element $g_U \in G$ such that $g_U^{-1} U$ is a green node.
- For each green node U, a spoke function h_U for the action of G_U on the children of U.

The analogue of Algorithm 2.2 runs as follows.

Algorithm 3.2. *Given a linear orbit lookup tree T_n of depth n, for any $i \in \{0, \dots, n\}$ and any sequence $x_1, \dots, x_i$ of linearly independent elements of S, the following recursive algorithm determines whether $\langle x_1, \dots, x_i \rangle$ is eligible, and if so produces a green node U of T_n and an element $g \in G$ such that $gU = \langle x_1, \dots, x_i \rangle$.*

1. *If $i = 0$, return $U := 0$, $g := 1_G$ and stop.*
2. *If $\langle x_1, \dots, x_{i-1} \rangle$ is a node of T, let U' be this node and set $g_0 := 1_G$. Otherwise, apply the algorithm to $x_1, \dots, x_{i-1}$ to obtain a green node U' of T and an element $g_0 \in G$ for which $g_0 U' = \langle x_1, \dots, x_{i-1} \rangle$. If instead we find that $\langle x_1, \dots, x_{i-1} \rangle$ is ineligible, report that U is ineligible and stop.*
3. *Set $y := g_0^{-1} x_i$, $U_1 := h_{U'}(U' + \langle y \rangle)^{-1}(U' + \langle y \rangle)$, $g_1 := g_0 h_{U'}(y)$.*
4. *Set $g_2 := g_{U_1}$ and return $U := g_2^{-1} U_1$, $g := g_1 g_2$. If instead we find that g_{U_1} is undefined, then report that U is ineligible.*

Algorithm 3.3. *Given a linear orbit lookup tree T_n of depth n, the following algorithm extends T_n to a linear orbit lookup tree T_{n+1} of depth $n+1$, and in addition computes the stabilizer of each green node U at depth $n+1$.*

1. *For each green node U at depth n:*
   (a) *Apply Algorithm 1.3 to construct a spoke function h_U for the action of G_U on $S \setminus U$.*
   (b) *For each hub v for h_U, add to T_{n+1} an uncolored node $U \cup \{v\}$ with parent U.*

(2) *For each node U at depth $n+1$ which is not already colored:*
  (a) *Let P_n be the maximal parabolic subgroup $\begin{pmatrix} \mathrm{GL}_n(k) & * \\ 0 & k^\times \end{pmatrix}$ of $\mathrm{GL}_{n+1}(k)$, and fix a set $\{w_j\}_{j \in J}$ of nontrivial left coset representatives for P_n in $\mathrm{GL}_{n+1}(k)$. For each j, apply steps (1)–(3) of Algorithm 2.2 (as in Remark 2.3) to the sequence $(\sum_{i=1}^{n+1} w_{ih} x_i)_{h=1}^{n+1}$ to find a node U_j and an element $g_j \in G$ such that $g_j U_j = U$, or to detect that U is ineligible.*
  (b) *If U is forbidden, or was detected to be ineligible in the previou step, then color U and each U_j black.*
  (c) *Otherwise:*
    (i) *Color U green and set $g_U = 1$.*
    (ii) *For each set V arising as U_j for some j, pick one such j, color V red, and set $g_V := g_j^{-1}$.*
    (iii) *Let U' be the parent of U and write $U = U' \cup \{x_{n+1}\}$. Apply Algorithm 1.4 to the spoke function $h_{U'}$ to compute $G_{U'} \cap G_{x_{n+1}} = G_{U'} \cap G_U$.*
    (iv) *Compute G_U as the subgroup of G generated by $G_{U'} \cap G_U$ together with g_j for each $j \in \{1, \dots, n\}$ for which $U_j = U$.*

(3) *Remove all black nodes.*

Proof. Let U be an arbitrary green node at depth $n+1$. Write U as $\langle x_1, \dots, x_{n+1} \rangle$ in such a way that $U' := \langle x_1, \dots, x_n \rangle$ is the parent of U.

We first observe that U is eligible, thanks to the following points.

- Since T_n is an orbit lookup tree, U' contains no forbidden subspace.
- By step (2)(a), no codimension-1 subspace of U other than U' contains a forbidden subspace.
- By step (2)(b), U is not forbidden.

We next verify by way of contradiction that there cannot be another green node V in the G-orbit of U. Without loss of generality we may assume that U occurs before V in step (2). Write $V = \langle y_1, \dots, y_{n+1} \rangle$ in such a way that $V' := \langle y_1, \dots, y_n \rangle$ is the parent of V, and choose an element g with $gU = V$. If $gU' = V'$, then we would have $U' = V'$ because T_n is an orbit lookup tree; then step (1)(a) would force $U' + \langle x_{n+1} \rangle = U' + \langle y_{n+1} \rangle$. Otherwise, g carries U' into some other codimension-1 subspace of V; this space is also the image of V' under some w_j for the left action on $\mathrm{GL}_{n+1}(k)$ on V. For this j, by step (2)(a), the parent of U_j must equal V'; step (1)(a) would then force $U_j = V$. Step (2)(c)(ii) would then color V red, a contradiction.

We next verify that T_{n+1} is an orbit lookup tree. By step (2)(c)(ii), every red node at depth $n+1$ is in the G-orbit of a green node, and in particular is eligible. By the previous paragraphs, the green nodes at depth $n+1$ form a set of G-orbit representatives for the eligible $(n+1)$-element subsets of S. By step (1), the nodes at depth n have the requisite children.

We finally verify that G_U is computed correctly. The correctness of Algorithm 1.4 ensures that $G_{U'} \cap G_U$ is computed correctly. This intersection is the stabilizer of the quotient U/U' for the action of G_U on the projective space $\mathbf{P}(U)$. If we interpret this action as a homomorphism $G_U \to \mathrm{PGL}_{n+1}(k)$, then the coset $g_j(G_{U'} \cap G_U)$ maps to the image of the coset $w_j P_n$ via $\mathrm{GL}_{n+1}(k) \to \mathrm{PGL}_{n+1}(k)$; step (2)(c)(iv) ensures that the former is included into G_U. □

To concretize step (2)(a) of Algorithm 3.3, we specify some coset representatives.

Lemma 3.4. *Let J be the set of vectors $j = (j_1, \dots, j_{n+1}) \in k^{n+1}$ such that $i_j := \min\{i \in \{1, \dots, n\} : j_i \neq 0\}$ exists and satisfies $j_{i_j} = 1$. For each $j \in J$, let $M_j \in \mathrm{GL}_{n+1}(k)$ be given by*

$$(M_j)_{ab} = \begin{cases} 1 & a = b \notin \{i_j, n+1\} \text{ or } (a,b) = (i_j, n+1) \\ j_b & a = n+1 \\ 0 & \text{otherwise}; \end{cases}$$

then set $w_j := M_j^{-1}$. The set $\{w_j\}_{j \in J}$ then forms a set of nontrivial left coset representatives of P_n in $\mathrm{GL}_{n+1}(k)$.

Proof. We first verify that M_j is invertible. The last row of M_j is the vector j, while the others are the standard basis vectors of k^{n+1} other than the i_j-th. Since $j_{i_j} \neq 0$, these vectors are linearly independent.

It now suffices to verify that the M_j form a set of *right* coset representatives of P_n in $\mathrm{GL}_{n+1}(k)$. For the right action of $\mathrm{GL}_{n+1}(k)$ on k^{n+1}, P_n is the stabilizer of the subspace generated by the last basis vector; the claim then reduces to the fact that the bottom rows of the M_j (which is to say, the vectors $j \in J$) form a transversal of the other one-dimensional subspaces of k^{n+1}. $\square$

Remark 3.5. The analogue of Remark 2.5 for Algorithm 3.3 is that if there are no forbidden subspaces, then the sum of $[G : G_U]$ over green nodes U at depth n should equal

$$\prod_{i=0}^{n-1} \frac{q^{\dim(S)-i} - 1}{q^{n-i} - 1}, \qquad q = \#k.$$

References

[1] L. Babai, G. Cooperman, L. Finkelstein, E. Luks, and Á. Seress, Fast Monte Carlo algorithms for permutation groups, 23rd Symposium on the Theory of Computing (New Orleans, LA, 1991), *J. Comput. System Sci.* **50** (1995), 296–308.

[2] Y. Huang, K.S. Kedlaya, and J.B. Lau, A census of genus 6 curves over $\mathbb{F}_2$, arXiv:2402.00716v1 (2024).

[3] K.S. Kedlaya, The relative class number one problem for function fields, III, in *LuCaNT: LMFDB, Computation, and Number Theory*, Contemp. Math. 796, Amer. Math. Soc., 2024.