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. Setup

Definition 1.1. Throughout, let G be a *computable finite group.* More precisely, G is specified as an explicit set of bitstrings; generators of G are specified as an explicit bitstring; and we are provided oracles for the following function.

- The multiplication map $\mu\colon G \times G \to G$.
- The inversion map $\iota\colon G \to G$. (Note that μ and ι can be used to produce the identity element 1_G.)

Date: version of 26 Feb 2024.

- The function taking a sequence S of elements of G to the order of the subgroup $\langle S \rangle$ generated by S. (Note that this can also be used to test membership in the subgroup: we have $g \in S$ iff $\#\langle S \rangle = \#\langle S \cup \{g\} \rangle$.)

Remark 1.2. Starting from any generating set of G, one can quickly construct a short generating set (of size at worst $O(\log \#G)$) by taking random products [1, Theorem 2.5]. In practice, we will work with such a short generating set, such as in Algorithm 2.2 where the complexity has a simple linear dependence on the size of a generating set.

Definition 1.3. Throughout, let S be a finite set equipped with a *computable left G-action*. More precisely, S is specified as an explicit set of bitstrings; and we are provided an oracle computing the action map $\mu \colon G \times S \to S$. We do *not* require that S can be enumerated.

Definition 1.4. We recall some more or less standard definitions.

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

2. Hub data for group actions

There is a standard (in all but terminology) algorithm to compute orbits, stabilizers, and transporters.

Definition 2.1. By a *hub datum* for (the G-action on) S, we will mean the following.

- A transversal T of the partition of S into orbit representatives (the *hubs*).
- The function assigning to each $v \in T$ its stabilizer G_v.
- A function $h \colon S \to G$ for which $h(x)^{-1}x \in T$ for all $x \in S$ (the *spoke function*).

We will notate the hub datum as (T, h) to indicate the symbols being used to represent these objects.

When S can be enumerated, we can produce a hub datum by finding a spanning tree in some Cayley digraph for the action using a breadth-first search. This also yields stabilizers as a byproduct.

Algorithm 2.2. *Assuming that S can be enumerated, the following algorithm computes a hub datum for the action of G on S.*

```
(1) Fix a generating subset $H$ of $G$ and orderings of $S, H, S \times H$.
(2) For each $v \in S$, if $h(v)$ is not yet assigned, then set $h(v) := \mathrm{id}_G$, $Q := \{v\}$,
    $R := \emptyset$ and repeat the following while $Q$ is nonempty:
    (a) Pick any $w \in Q$ and remove $w$ from $Q$.
    (b) For each $g \in H$, set $x := gw$.
        (i) If $h(x)$ is assigned, add $(w, g)$ to $R$.
        (ii) Otherwise, set $h(x) := gh(w)$ and add $x$ to $Q$.
(3) Let $O_v$ be the domain of $h$ and set $I := \emptyset$. For each $(w, g) \in R$, if $I$ generates
    a subgroup of $G$ of order less than $\#G/\#O_v$, add $h(gw)^{-1}gh(w)$ to $I$.
(4) Let $G_v$ be the subgroup of $G$ generated by $I$.
```

Proof. It is clear that h is correctly defined on its domain and that I consists of elements of the stabilizer of v. To check that O_v is correctly computed, note that any $w \in O_v$ can be written as $g_n \cdots g_1 v$ for some nonnegative integer n and some $g_1, \dots, g_n \in H$. By step (2) and induction on i, h is defined on $g_i \cdots g_1 v$ for $i = 0, \dots, n$, and in particular on w.

To check that step (3) terminates, write an arbitrary element c of G_v as $g_n \cdots g_1$ for some nonnegative integer n and some $g_1, \dots, g_n \in H$. Then set $v_i := g_i \cdots g_1 v$ for $i = 0, \dots, n$ and rewrite

$$c = (h(v_n)^{-1} g_n h(v_{n-1})) \cdots (h(v_2)^{-1} g_2 h(v_1))(h(v_1)^{-1} g_1 h(v_0));$$

each parenthesized expression eventually belongs to the subgroup generated by I. Hence at some point I generates G_v, as claimed. □

Remark 2.3. Note that Algorithm 2.2 produces the spoke function h in the form of a function table; it thus cannot be applied in a context where the set S is too large to store in memory. Our approach in such cases will be to produce computable spoke functions by a suitable composition of computable functions.

3. Relative orbit-stabilizer computations

The situations we will address using orbit lookup trees can be abstracted as follows.

Definition 3.1. Let S_1, S_2 be two finite sets equipped with computable left G-actions. A *correspondence* between S_1 and S_2 is a G-invariant subset X of $S_1 \times S_2$ which projects surjectively onto each factor. For a correspondence to be *computable*, we require an oracle for enumerating the fiber of the projection map to any specified element of S_1 or S_2.

In the context of computing hub data, we can use a computable correspondence to "step between" the actions on S_1 and S_2.

Algorithm 3.2. *For $X \subset S_1 \times S_2$ a computable correspondence in the sense of Definition 3.1, given a hub datum (T, h) for S_1, the following algorithm produces a hub datum (T, h) for X.*

(1) *For each $v \in T_1$, apply Algorithm 2.2 to compute a hub datum T_v, h_v for action of G_v on the fiber X_v of X over v.*
(2) *Put $T := \bigsqcup_{v \in T_1} \{v\} \times T_v$. For each $v \in T_1$, $w \in T_v$, compute $G_{(v,w)}$ as the stabilizer of w in G_v.*
(3) *Define $h \colon X \to G$ as follows: for $x = (x_1, x_2) \in X$, set $v := h_1(x_1)^{-1} x_1$, $w := h_1(x_1)^{-1} x_2$, and $h(x) := h_1(x_1) h_v(w)$.*

Proof. Straightforward. □

Algorithm 3.3. *For $X \subset S_1 \times S_2$ a computable correspondence in the sense of Definition 3.1, given a hub datum (T, h) for X, the following algorithm produces a hub datum (T_2, h_2) for S_2.*

(1) *Set $T_2 := \emptyset$ and $V := \emptyset$.*
(2) *For each $(v, w) \in T$, if $w \notin V$, then:*
    (a) *Add w to T_2 and V.*
    (b) *Apply Algorithm 2.2 to compute a hub datum (T_v, h_v) for the action of G_v on the fiber X_v over v.*
    (c) *For each $w' \in T_v$,*

4. Orbit lookup trees

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

Definition 4.1. 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 of S,
- For $i = 0, \dots, n$, the nodes at depth i form a set of G-orbit representatives for the eligible i-element subsets of S.
- Every child of a node is a superset of that node.

In particular, each node U admits a unique ordering $x_1, \dots, x_i$ such that each initial segment of this sequence is also a node; we write $U = [x_1, \dots, x_i]$ instead of $U = \{x_1, \dots, x_i\}$ 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 at depth $i < n$:

- A spoke function h_U for the action of G_U on the eligible $(i+1)$-element subsets of S containing U (which can be identified with certain elements of $S \setminus U$).
- For each hub V of h_U, an element $g_V \in G$ such that $g_V^{-1} V$ is a node.

Algorithm 4.2. *Given an orbit lookup tree T_n of depth n, for any $i \in \{0, \dots, n\}$ and any sequence $x_1, \dots, x_i$ of distinct elements of S, the following recursive algorithm determines whether $\{x_1, \dots, x_i\}$ is eligible, and if so produces a node U of T_n and an element $g \in G$ such that $gU = \{x_1, \dots, x_i\}$.*

(1) *If $i = 0$, return $U := \emptyset$, $g := 1_G$ and stop.*
(2) *If $\{x_1, \dots, x_{i-1}\}$ is a node of T, let U' be this node and set $g_0 := 1_G$. Otherwise, apply the algorithm to $x_1, \dots, x_{i-1}$ to obtain a node U' of T and an element $g_0 \in G$ for which $g_0 U' = \{x_1, \dots, x_{i-1}\}$. If instead we find that $\{x_1, \dots, x_{i-1}\}$ 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 4.3. When constructing an orbit lookup tree in Algorithm 4.4, we will apply Algorithm 4.2 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 4.2; in particular, the recursive call in step (2) can be executed.

In this context, it is useful to perform Algorithm 4.2 in batches; by choosing the inputs carefully we may reduce the number of recursive calls. See Remark 4.6 for more details.

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

Algorithm 4.4. *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 node at depth $n+1$.*

(1) *For each node U at depth n, apply Algorithm 2.2 to construct a spoke function h_U for the action of G_U on $S \setminus U$.*
(2) *Form the set X of $(n+1)$-element subsets U of S which occur as a hub for $h_{U'}$ for some node U' at depth n. For each $U \in X$:*
   (a) *For $j = 1, \dots, n$, let w_j be an element of the left coset $(j(n+1))S_n$ in S_{n+1}. Apply steps (1)–(3) of Algorithm 4.2 (as in Remark 4.3) to the sequence $x_{w_j(1)}, \dots, x_{w_j(n+1)}$ to find a set 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 not forbidden and U_j exists for all j:*
      (i) *Set $g_U := 1_G$.*
      (ii) *For each set V arising as U_j for some j, pick one such j and set $g_V := g_j^{-1}$.*
      (iii) *Let U' be the node for which U occurs as a hub for U'. Add U to T_{n+1} as a child of U'.*
      (iv) *Write $U = U' \cup \{x_{n+1}\}$. Apply Algorithm **??** to the spoke function $h_{U'}$ to compute $G_{U'} \cap G_{x_{n+1}} = G_{U'} \cap G_U$.*
      (v) *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 $U_j = U$.*
   (c) *Remove U and each U_j from X.*

Proof. Let U be an arbitrary 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 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) 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) would then cause V to be skipped, a contradiction.

We next verify that T_{n+1} is an orbit lookup tree. By the previous paragraphs, the 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 **??** 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 4.5. 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 4.4: the sum of $[G : G_U]$ over nodes U at depth n should equal $\binom{|S|}{n}$.

Remark 4.6. For the correctness of Algorithm 4.4 it does not matter which coset representatives w_j we use; however, making a good choice may have a practical impact in step (2)(a) if one executes Algorithm 4.2 in batch mode. For example, using the coset representatives produced by Algorithm 4.7, each instance of step (2)(a) at depth n incurs $O(n \log n)$ recursive calls to Algorithm 4.2 rather than $O(n^2)$.

Algorithm 4.7. *Given a positive integer n and an integer $j \in \{1, \dots, n\}$, the following algorithm returns a left coset representative of $(j(n+1))S_n$ in S_{n+1}.*

```
(1) Let $a$ be the sequence $1, \dots, n+1$ and let $b$ be the empty sequence.
(2) While $a$ has length greater than $1$:
    (a) Divide $a$ into two halves as evenly as possible; if $a$ is of odd length,
        make the right half longer.
    (b) Choose the half not containing $j$, append it to $b$, and delete it from $a$.
(3) Let $w_j$ be the permutation taking $1, \dots, n$ to $b$ and $n+1$ to $j$.
```

Remark 4.8. In [3, Remark A.7], it is suggested that one can further optimize Algorithm 4.2 by reducing the number of values of j used in step (2)(a) 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 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.

5. 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 5.1. 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 of S.
- 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.
- Every child of a node is a superset of that node.

A *linear orbit lookup tree* is a linear orbit tree equipped with the following additional data for each node U at depth $i < n$:
A spoke function h_U for the action of G_U on the eligible $(i+1)$-dimensional subspaces of S containing U (which can be identified with certain one-dimensional subspaces of S/U).

- For each hub V of h_U, an element $g_V \in G$ such that $g_V^{-1}V$ is a node.

The analogue of Algorithm 4.2 runs as follows.

Algorithm 5.2. *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 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 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.*

The analogue of Algorithm 4.4 runs as follows.

Algorithm 5.3. *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 node U at depth $n+1$.*

(1) *For each node U at depth n, apply Algorithm 2.2 to construct a spoke function h_U for the action of G_U on the one-dimensional subspaces of S/U.*

Form the set X of $(n+1)$-dimensional subspaces U of S which occur as a hub for $h_{U'}$ for some node U' at depth n. For each $U \in X$:

(1) *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 4.2 (as in Remark 4.3) to the sequence $(\sum_{i=1}^{n+1} w_{ih} x_i)_{h=1}^{n+1}$ to find a set U_j and an element $g_j \in G$ such that $g_j U_j = U$, or to detect that U is ineligible.*
(2) *If U is not forbidden and U_j exists for all j:*
   (i) *Set $g_U := 1_G$.*
   (ii) *For each set V arising as U_j for some j, pick one such j and set $g_V := g_j^{-1}$.*
   (iii) *Let U' be the node for which U occurs as a hub for U'. Add U to T_{n+1} as a child of U'.*
   (iv) *Apply Algorithm **??** to the spoke function $h_{U'}$ to compute $G_{U'} \cap G_U$.*
   (v) *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 $U_j = U$.*

Proof. Let U be an arbitrary 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 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 cause V to be skipped, a contradiction.

We next verify that T_{n+1} is an orbit lookup tree. 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 **??** 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. □

Remark 5.4. As in Remark 4.6, we pick the coset representatives w_j to maximize redundancy in the batch application of Algorithm 5.2. We may take J to be any set of generators of the one-dimensional subspaces of $k^{n+1} \cong k^n \times k$ not containing k^n. For each $j \in J$, we form w_j so that its first n column vectors form the lexicographically earliest basis of the orthogonal complement of j. Using this choice, each instance of step (2)(a) at depth n incurs $O(2^n)$ recursive calls to Algorithm 4.2 rather than $O(n2^n)$.

Remark 5.5. The analogue of Remark 4.5 for Algorithm 5.3 is that if there are no forbidden subspaces, then the sum of $[G : G_U]$ over 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.