Orbiter User's Guide

Anton Betten

September 4, 2018

Abstract

We discuss how to use the program system Orbiter for the classification of combinatorial objects.

1 Introduction

This is a User's guide to the program package Orbiter [1] [2] for the classification of combinatorial objects. Orbiter is a library of C++ classes for algebraic combinatorial objects. It does not have a user interface, but it comes with several pre-programmed applications which can be invoked using the command line or using makefiles. This means that there are basically two ways in which Orbiter can be used. The first way is recommended for novice users. Those kinds of users would use Orbiter applications through unix command lines, possibly using makefiles to simplify the typing of commands. The second type of user are more experienced programmers. They would build their own applications using the C++ library. In both cases, the orbiter application is developed using the orbiter C++ library, which in turn uses the C++ standard library (cf. Fig. 1). The Orbiter library is layered. There are five components, one building on top of the other. This is shown in Fig. 2.

2 Installing Orbiter

Orbiter is distributed through github (`https://github.com`). Search for "abetten/orbiter" or go directly to

`https://github.com/abetten/orbiter`

Once there, find the green button called "Clone or download". The button offers two options: "Open in Desktop" and "Download ZIP" Choose one of them to download Orbiter. Orbiter is compiled with makefiles. Some system specific comments are in order:

- For Microsoft Windows users, it is recommended to install Orbiter using `cygwin` [6]. Cygwin is a Unix-like environment and command-line interface for Microsoft Windows.

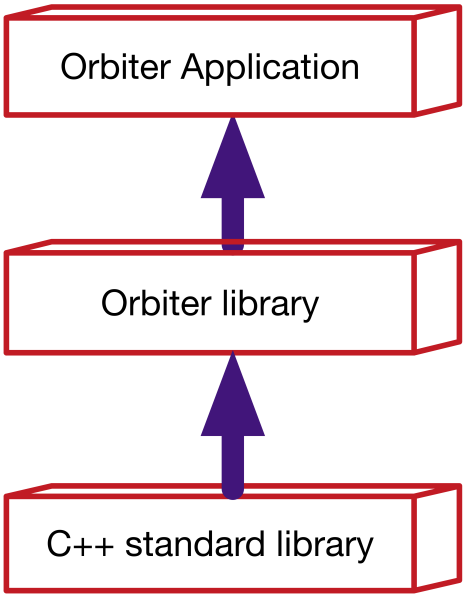


Figure 1: The orbiter model

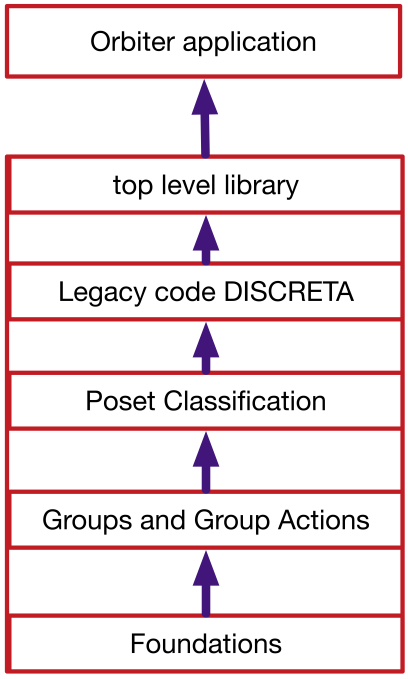


Figure 2: The layers of the orbiter library

- Macintosh users need to install Xcode (search for xcode in the app store). Xcode is an integrated development environment (IDE) for MacOS. It comes with command line tools for software development. In order to compile Orbiter, the command line tools are required. There is no need to use the integrated development environment though.
- For Linux users, the compiler environment (for instance Gnu C++) needs to be installed.

We will use a terminal window (console) to install Orbiter. Assuming that we have the various compiler tools avalable, the installation proceeds as follows. The following commands are typed into the terminal window.

Enter the directory `ORBITER/src` and type

```
make
```

This should create a lot of text output to the console. Assuming that the command executes without errors, orbiter is now ready. The specific purpose of this make command is to compile all C++ source code into object files, to bind the object files together into one library, and to link the orbiter executables. The C++ cource code is in files with extension .cpp. There are additional filed called header files with extension .h. The header files are needed to compile the .cpp files into .o files. This has to do that one source file needs to know a little bit what goes on in the other source files. The object files are corresponding files with extension .o.

$$\Big(\text{XXX.cpp},\ \text{orbiter.h} \Big) \ \mapsto \text{XXX.o} \ \mapsto \ \text{liborbiter.a}.$$

Here, XXX stands for all files in the /src/lib subtree, and the map XXX.cpp $\mapsto$ XXX.o is one-to-one. The map XXX.o $\mapsto$ liborbiter.a is many-to-one. Once the library has been compiled, the application executables are compiled:

$$\left.\begin{array}{c} \text{YYY.cpp} \ \mapsto \ \text{YYY.o} \\ \text{liborbiter.a} \\ \text{standard libraries (libc++ etc.)} \end{array}\right\} \ \mapsto \ \text{YYY.out}$$

This time, the source files YYY reside in the src/apps branch. The file YYY.out is the executable. Executables are programs which can be called. They are recognizable by the x flag in the directory list). The executable contains the actual program to do the work. It will be called for instance through the command line. This make command has to be executed only once. One can recognize the fact that make has been successful by verifying the presence of files with extension .o on the src subdirectories. Also, various files with the extension .out have been created in the subdirectories of src/apps. The make command descends into all subdirectories of src and performs make.

In order to test orbiter, go to the subdirectory `ORBITER/examples`. There are several subdirectories containing test problems. The test problems are calls to orbiter, asking it so solve small problems. For instance, the subdirectory groups contains some problems related to

groups. Once in the directory `ORBITER/examples/groups`, issue the command `make` to run the first problem. To see what kind of problems are available, open the file `makefile` and see what targets are defined. You can type `make target` where `target` is any of the targets in the makefile to run that particular problem. Most problems will create a lot of output to the console.

3 Combinatorial Objects

Orbiter is a software tool devoted to the study of combinatorial objects. So, what is a combinatorial object? It is not clear if there is an accepted definition for combinatorial object. The approach taken here is based on the notion of groups acting on sets. Let G be a group acting on a set X. The elements of X fall into orbits under G. For $x, y \in X$, the relation $x \sim_G y$ is defined whenere there exists an element $g \in G$ with $xg = y$. The equivalence classes of this relation are the orbits of G on X. We say that a combinatorial object is an orbit of a group G acting on a set X. Combinatorial objects are almost always given by means of representing elements. That is, an element $xinX$ is given to represent the orbit of x under G. If for two elements $x, y \in G$ we have $x \sim_G y$ then x and y are called isomorphic. The type of a combinatorial object is the group action (X, G). The classification problem for combinatorial objects of type (X, G) is the problem of determining the orbits of G on X. Usually, this is understood as the problem of finding a transversal of the orbits of G on X. That is, a set $T \subseteq X$ is produced such that T intersects each orbit of G on X in exactly one element.

4 Theoretical Background

Combinatorial objects are orbits of group actions. Hence classifying combinatorial objects is equivalent to computing the orbits of a group acting on a set. Let G be a group and let X be a finite set. For two elements x, y of X, write $x \sim y$ if x and y belong to the same G-orbit. A transversal for the orbits of G on X is a sequence $T = (t_1, \ldots, t_n)$ for some integer n such that for every $x \in X$ there is exactly one t_i with $x \sim t_i$. The classification problem for the orbits of G on X is computing a transversal for the orbits. The recognition problem for the group G acting on the set X is the following: Given an element $x \in X$, determine the element $t_i \in T$ auch that $x \sim t_i$. Here, T is the transversal that was computed previously. The constructive recognition problem for the group G acting on the set X is the following: Given an element $x \in X$, determine the element $t_i \in T$ auch that $x \sim t_i$ and in adition find a group element $g \in G$ such that $xg = t_i$. The main challenge is to find efficient algorithms for the classification problem, the recognition problem and the constructive recognition problem.

Standard orbit algorithms are related to the idea of Schreier trees. The complexity of these algorithms is linear in the size of the orbit. This is fine for small problems, but Orbiter is designed to handle cases that require faster algorithms. For many combinatorial objects, posets can be used to aid the classification task. The group induces an action on the poset

and the orbits of the group on the poset are computed. Often times, the orbits on the objects at a specific layer in the poset correspond to the combinatorial objects that are desired. For the theory of group action on posets, see [16]. For a description of poset based classification, see [3]. With poset classification, the algorithm often proceeds much faster than the ordinary orbit algorithm based on Schreier trees. This is because the orbits are never fully looked at. Poset classification looks at a small fraction of the elements only in order to compute a transversal for the orbits.
Poset classification is the process by which the orbits of a group G on a poset $\mathcal{P}$ are classified. Additional information is computed also. This additional information described how the orbits are related. An early application of this can be seen in [5], where the orbits of the Mathieu group acting on the set of subsets of the set $\{1, \ldots, 24\}$ are computed and a diagram is presented which contains detailed information about how the orbits are related in the sense of [16].

In order to use Orbiter, a group G is defined. Then, an action of this group on a set X is constructed. Using the induced action on k-subsets or k-subspaces, a poset action on a lattice $\mathcal{L}$ is defined. Then, a subposet $\mathcal{P}$ of the lattice is selected, and the orbit of the group G on this poset $\mathcal{P}$ are computed. The orbit representatives and the stabilizer groups are listed and stored. We will describe these steps next.

The algorithm described in [3] is based on earlier work of Schmalz [20]. Schmalz was concerned with computing double coset representatives in certain groups. The groups that Schmalz considered turned out to have good "ladders" of subgroups. A ladder of subgroups is a sequences of subgroups $G_1, G_2, \ldots, G_r$ where two consecutive groups are related. This means that for $i = 1, \ldots, r-1$ we have $G_i \leq G_{i+1}$ or $G_i \geq G_{i+1}$. A subgroup ladder is good if the indices of consecutive terms are small. The problem of computing orbits on k-subsets and orbits on k-dimensional subspaces can be formulated equivalently as a problem of computing double cosets in either the symmetric group or the full semilinear matrix groups. For these groups, good subgroup ladders exist. Because of the use of subgroup ladders, Schmalz coined the term "Leiterspiel" for his algorithm. Perhaps the closest translation into English would be "Snakes and Ladders." Much of Orbiter is based on the Schmalz algorithm.

There are other algorithms which can be used to classify combinatorial objects. Most notably there is the method of canonical ancestors proposed by McKay [15], which may be seen as part of the family of algorithms which are called "orderly generation", and which have been discovered and refined many times, see [17], [8], [13], [14], [18], [9].

The Schmalz algorithm differs from orderly generation in some important ways. First, while orderly generation proceeds depth-first, the algorithm of Schmalz proceeds in a breadth-first manner. While orderly generation only keeps the current object in memory, the algorithm of Schmalz builds up the whole tree of orbits, storing quite a lot of information. While the orderly generation algorithm relies on a backtrack procedure to compute canonical forms, the algorithm of Schmalz does not backtrack. It uses the data structure about previously computed orbits to perform recognition. The differences in the two families are sufficiently large to expect different behavior of the algorithms on different kinds of problems.

All groups in Orbiter are finite permutation groups. However, we distinguish between groups of linear or affine type and ordinary permutation groups. Groups of linear type are groups that act linearly or affinely (possibly semilinearly) on the elements of a vector space. This allows us to encode group elements efficiently using matrices. For semilinear actions, we store a field automorphism. For affine actions, we store a translation vector. All other groups need their elements stored as permutations. Memory issues are very important in Orbiter, so finding the most efficient way to encode a group element matters. At times, a representation of matrices over finite fields as bitvectors is utilized, reducing the storage requirement even further.

All groups in Orbiter are permutation group. For every group, a fixed permutation representation is chosen from the beginning. Later, new groups actions can be defined. It is important to know that any group always has a default permutation representation, independent of how the group elements are represented. For projective linear groups, a labeling of the points of $\mathrm{PG}(n-1,q)$ is used. For general linear and affine groups, a labeling of the vectors in $\mathbb{F}_q^n$ is used. For orthogonal groups, a labeling of the points of $Q^\epsilon(n-1,q)$ is used. For permutation groups, the set $\{0,\ldots,d-1\}$ is used, for some d.

5 Orbiter Applications

This section describes some of the available Orbiter applications and how they can be used. Mainly, Orbiter applications rely on the command line to being told what to do. Options are passed along using certain command line keys, which must be known. Suppose we want to create one of the orthogonal groups over a finite field. The orbiter application to do so is called `orthogonal_group.out`. Since an orthogonal group $O^\epsilon(d,q)$ involves three parameters, the Orbiter application requires us to use three options. By default, most Orbiter parameters are integers. They are `-epsilon` $\langle\epsilon\rangle$, `-d` $\langle d\rangle$ and `-q` $\langle q\rangle$. For instance, the command

```
orthogonal_group.out -v 2 -epsilon -1 -d 6 -q 2
```

creates the group $O^-(6,2)$ of order 51840. The parameter `-v` $\langle v\rangle$ specifies the verbose level. Higher values of v mean more text output. This group $O^-(6,2)$ acts on the 27 points of the $Q^-(5,2)$ quadric. The quadric is given by the quadratic form

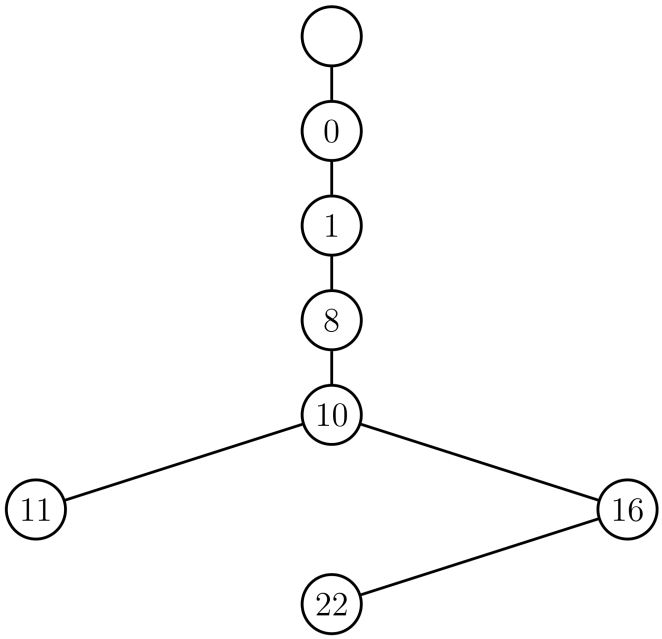
$$x_0x_1 + x_2x_3 + x_4^2 + x_4x_5 + x_5^2 = 0.$$

Generators for the group as 6×6 matrices over $\mathbb{F}_2$ are listed in the output, as are corresponding permutations of the 27 points. Finally, the generators are written in machine readable compact form.

The next problem we wish to consider is classifying ovoids in $\Omega^-(5,2)$ under the group $O^-(6,2)$ from before. A partial ovoid is a set $S=\{P_1,\ldots,P_n\}$ of points $P_i=\langle v_i\rangle$ such that $\beta(v_i,v_j)\neq 0$ for all $i\neq j$. The partial ovoids in $\Omega^-(5,2)$ can be classified using the orbiter command

```
ovoid.out -v 5 -epsilon -1 -n 5 -q 2 -draw_poset -embedded -W
```

This produces the following diagram:



which shows that the set $\{0, 1, 8, 10, 16, 22\}$ is the only partial ovoid in $\Omega^-(5, 2)$ up to equivalence. The numbers stand for points in $\Omega^-(5, 2)$ as listed in the output of the `orthogonal_group.out` command, mentioned earlier. They are

$$\begin{aligned} 0 &= (1, 0, 0, 0, 0, 0), \\ 1 &= (0, 1, 0, 0, 0, 0), \\ 8 &= (1, 1, 1, 1, 0, 0), \\ 10 &= (1, 1, 1, 0, 1, 0), \\ 16 &= (1, 1, 1, 0, 0, 1), \\ 22 &= (1, 1, 1, 0, 1, 1). \end{aligned}$$

6 Orbiter Design Principles

In order to fully utilize Orbiter, it is necessary to understand some of the design principles. There are several topics that are of great importance:

1. How groups and group actions are represented on a computer;
2. Which groups are available;

3. What kind of induced group actions are available;

4. How user-defined posets are realized;

5. How bijections are used in order to reduce the memory complexity of combinatorial objects;

6. How Orbiter trades time versus memory.

6.1 Using Bijections

Combinatorial objects are often built up from smaller objects. Hence we distinguish between atomic combinatorial objects and compound objects. Using bijections to integers can be helpful in two ways. First it reduces the the memory complexity of the object, thereby simplifying the data structures for atomic and compound objects. Second, it simplifies the handling of group actions on combinatorial objects. A group action can be realized as a function which takes a group element and a combinatorial object to yest another combinatorial object. If the combinatorial objects are mapped to a set of integers, then the group action function can take as input an integer and a group element. The function produces as output yet another integer, namely the combinatorial object that is the image. By using integers, the function to perform the mapping can have a standardized calling syntax for many different group actions. This is important for a sytem like Orbiter where there are many different group actions which all utilize the same scheme for function calling.

Orbiter uses standard bijections to map atomic combinatorial objects to integer intervals. For instance, the set of points of a finite dimensional projective or affine geometry over a finite field can be stored as integers rather than as vectors. Likewise, the set of subspaces of a projective geometry can be coded as integers. The points and lines of an orthogonal geometry can be coded. The points on a Hermitan geometry can be coded. The elements of a finite group can be coded. The set of k-subsets of a set can be coded. Ordered and unordered pairs of a set can be coded.

7 Poset Classification

There are a few software packages which can perform poset-classification. Why do we need another one? The answer to this question lies in performance considerations. Take a look at a sample problem. Suppose we want to classify hyperovals in the plane $\mathrm{PG}(2, 16)$. It is known that there are exactly two distinct hyperovals, but how do we find them and how do we show that there are no others? This is a typical question for a poset classification algorithm. The partially ordered set is the set of all arcs in $\mathrm{PG}(2, 16)$. The incidence is given by inclusion. The projective group $\mathrm{P\Gamma L}(3, 16)$ acts in this poset naturally. The orbits on subsets of size 18 are the isomorphism classes of hyperovals. These are the largest element in the poset. An interesting approach to classify those posets is McKay's algorithm canonical augmentation.

In this algorithm, a depth-first search is performed. Whenever a set S is considered (a set in the poset, so effectively an arc), the live points are computed. The live points are those points P of $\mathrm{PG}(2,16)$ which are not in S and which have the property that $S \cup \{P\}$ is an arc also. Let $\mathcal{L}(S)$ be the set of live points of the arc S. The stabilizer G_S of the set S acts on $\mathcal{L}(S)$. The orbits of this action are considered. A set of orbit representatives $t_1, \ldots, t_{\ell(S)}$ is computed and the arcs $S \cup \{t_i\}$ (for $i = 1, \ldots, \ell(S)$) are considered in turn. For each set $S \cup \{t_i\}$, a canonical form and the stabilizer $G_{S\cup\{t_i\}}$ is computed. This is done so that an element $p \in S \cup \{t_i\}$ is distinguished. If p and t_i belong to the same orbit of $G_{S\cup\{t_i\}}$, the set $S \cup \{t_i\}$ is accepted and the recursive procedure is applied to it. Otherwise, the set $S \cup \{t_i\}$ is rejected and the next set $S \cup \{t_{i+1}\}$ is considered. The sets of size 18 encountered in this search are the representatives of the hyperovals of $\mathrm{PG}(2,16)$.

Using the Orbiter-Nauty interface, this algorithm can be tested in Orbiter. Nauty is the program provided by McKay to compute the canonical form of graphs. The Orbiter program turns each set S in $\mathrm{PG}(2,16)$ into a graph. The canonical form of the graph allows to find the canonical predecessor p. Also, Nauty produces the set stabilizer G_S. The Orbiter-Nauty interface turns this group into a matrix group inside $\mathrm{P\Gamma L}(3,16)$. With this group at hand, the program can compute the orbits on the set of live points $\mathcal{L}(S)$ as required in the algorithm. In Figure 3, the calltree of this algorithm is displayed (produced using valgrind [7]).

Next, we use Orbiter to classify the hyperovals in $\mathrm{PG}(2,16)$. Orbiter uses a different algorithm. The algorithm originated in work of Schmalz, and got several face-lifts before the version implemented in Orbiter was conceived. Though both algorithms build up the same search tree, there are some main differences in the two algorithms. Most prominently, Orbiter proceeds in a breadth first manner, In Figures 4 and 5, the calltree of the Orbiter algorithm is displayed.

It is interesting to compare the two algorithms using these call-trees. The piece in the algorithm that takes up most of the time is the part where isomorph rejection happens. In the canonical predecessor algorithm, this is computing the canonical form, the automorphism and the canonical predecessor. In the call-tree, this is the function densenaut, which takes up 67% of the overall execution time. In Orbiter, the function which takes up most of the time is recognize, which also does isomorph rejection. This function takes up 71% of the overall execution time, comparable with the canonical form part in the first algorithm. However, the overall execution time is 50 seconds for canonical augmentation an only 1 second for Orbiter. Both algorithms create the same search tree. The search tree has about 35,000 nodes.

8 Orbiter History and Design Issues

Orbiter is based on an earlier system called DISCRETA [4]. DISCRETA is a system to construct combinatorial objects called t-designs using an assumed symmetry group. DISCRETA in turn was influenced by two systems: SYMMETRICA and DCC. SYMMETRICA

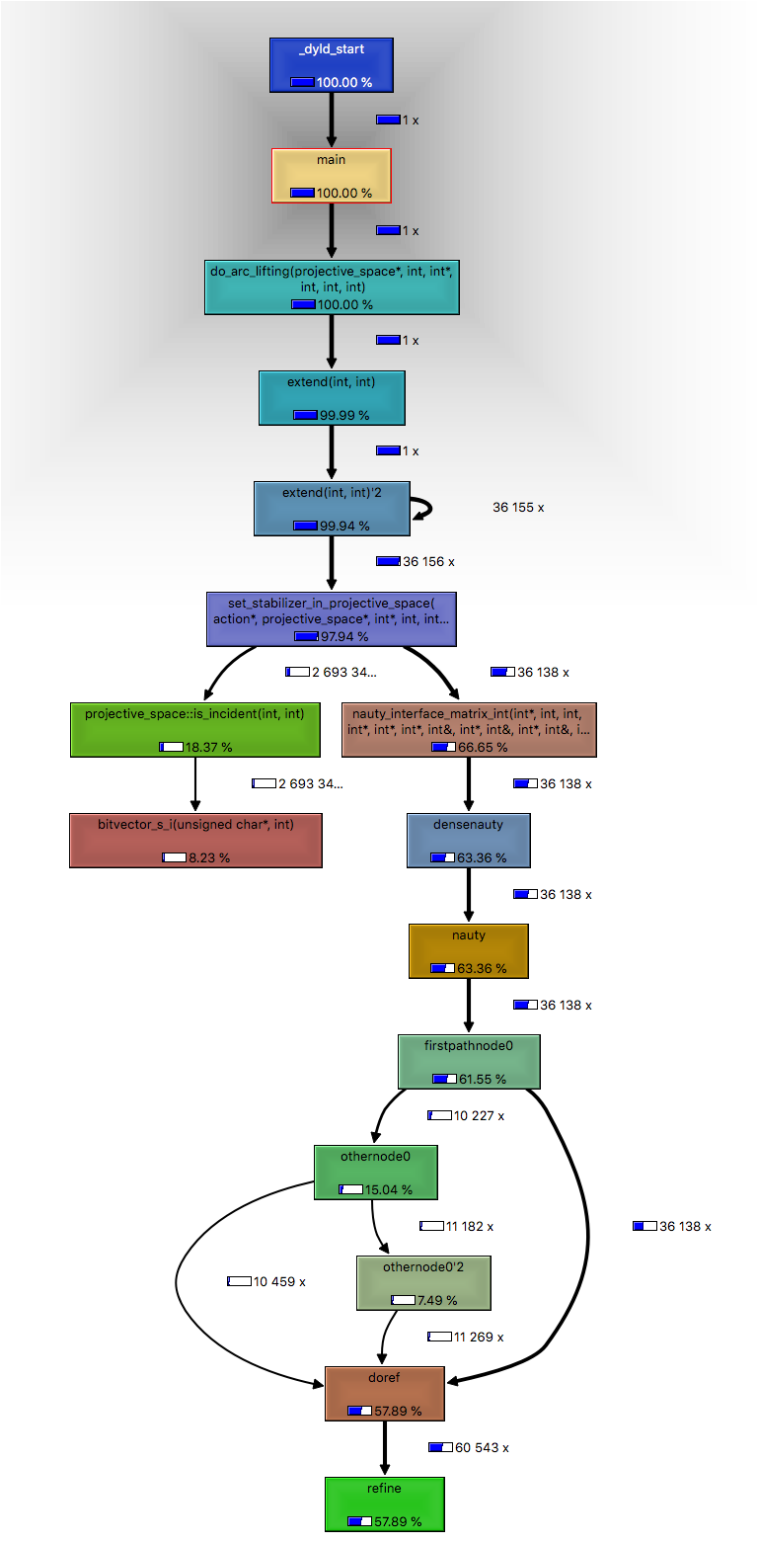


Figure 3: Classifying hyperovals in PG$(2, 16)$ using Nauty

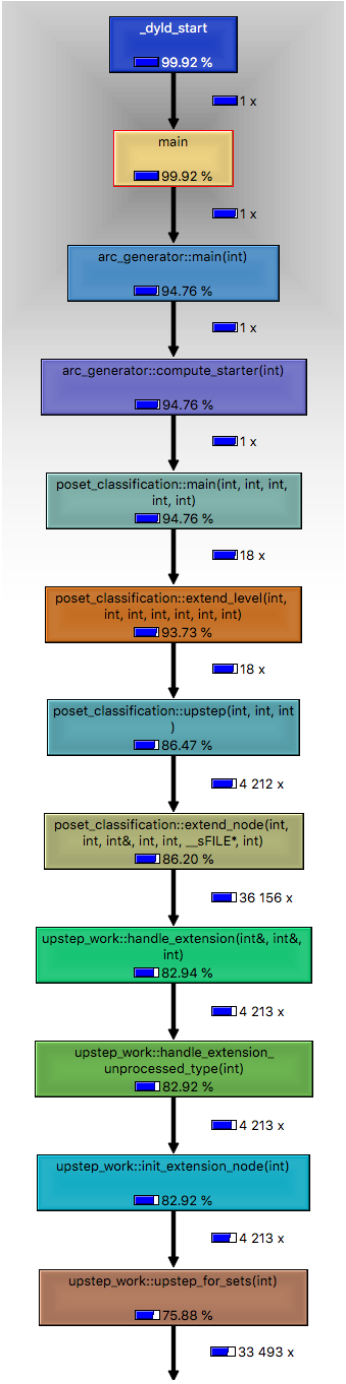


Figure 4: Classifying hyperovals in PG$(2, 16)$ using Orbiter (top part)

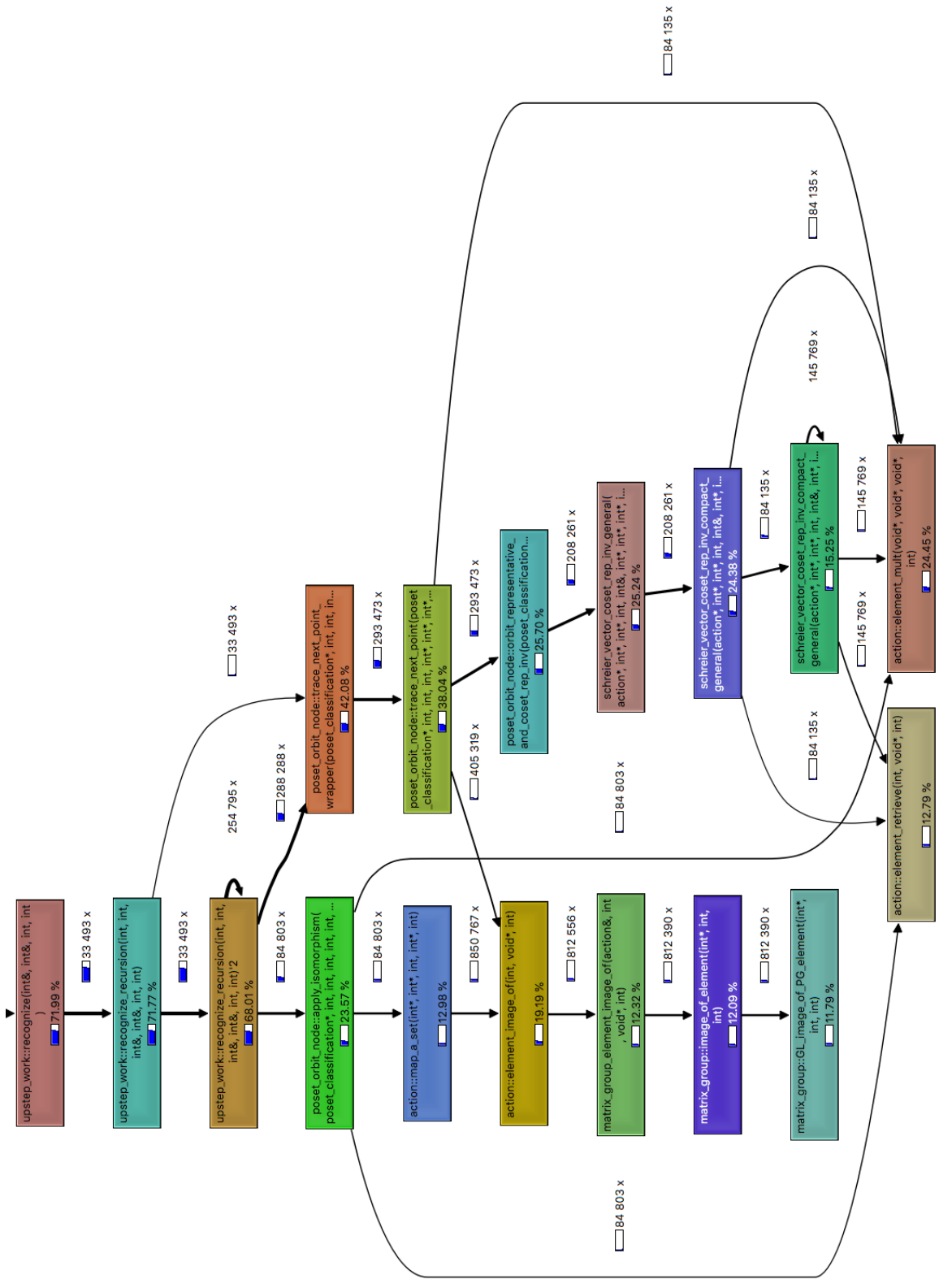


Figure 5: Classifying hyperovals in PG(2, 16) using Orbiter (bottom part)

(see [12], [10],[11]), was a system devoted to the representation theory of the symmetric group (the main author, Axel Kohnert, passed away in 2013 following a tragic bicycle incident). DCC ("double coset constructor") was a program to construct double cosets in groups due to Bernd Schmalz [20],[19]. SYMMETRICA, DCC, DISCRETA and many other software projects took place over a period of perhaps 30 years at the Lehrstuhl II of Mathematics of Professor Kerber and Professor Laue from the University of Bayreuth in Germany.

SYMMETRICA had an interesting design strategy. Though written in C, it followed a clear object-oriented design philosophy. The design was developed by Axel Kohnert in collaboration with professor T. P. McDonough from the university of Aberystwyth in the UK. There was only one class, of type object, and all types of objects were realized in this class. A type value inside this class indicates which specific type of object was realized. In Fig. 6, a collaboration diagram shows the relations between the different types of objects. The type value is used to dispatch function calls to the appropriate implementation for the type at hand. This mechanism resembles virtual functions in C++. Here is a piece of C source extracted from the header file of SYMMETRICA. This piece of code is responsible for the design of objects as a single type value plus one data element of type union:

```
1       typedef INT OBJECTKIND;
2
3       typedef union {
4          INT ob_INT;
5          INT * ob_INTpointer;
6          char *ob_charpointer;
7          struct bruch *ob_bruch;
8          struct graph *ob_graph;
9          struct list *ob_list;
10         struct longint *ob_longint;
11         struct matrix *ob_matrix;
12         struct monom *ob_monom;
13         struct number *ob_number;
14         struct partition *ob_partition;
15         struct permutation *ob_permutation;
16         struct reihe *ob_reihe;
17         struct skewpartition *ob_skewpartition;
18         struct symchar *ob_symchar;
19         struct tableaux *ob_tableaux;
20         struct vector *ob_vector;
21         } OBJECTSELF;
22
23
24      struct object { OBJECTKIND ob_kind; OBJECTSELF ob_self; };
25
26
```

Orbiter and DISCRETA both took some inspiration from this design. Perhaps it is illustrative to take a quick look at DISCRETA, the predecessor system of Orbiter. The design of Orbiter grew out of some ideas introduced in DISCRETA around 2000. We can trace this to a specific set of classes. One of the mathematical problems that drove the development of both DISCRETA and Orbiter was the problem of classifying optimal linear codes. In DISCRETA, we find the collaboration diagram for a class `linear_codes_data` shown in Figure 7. On the left side, we see classes base and OBJECTSELF which represent the SYMMETRICA tradition of storing objects. These objects are used to store the group by

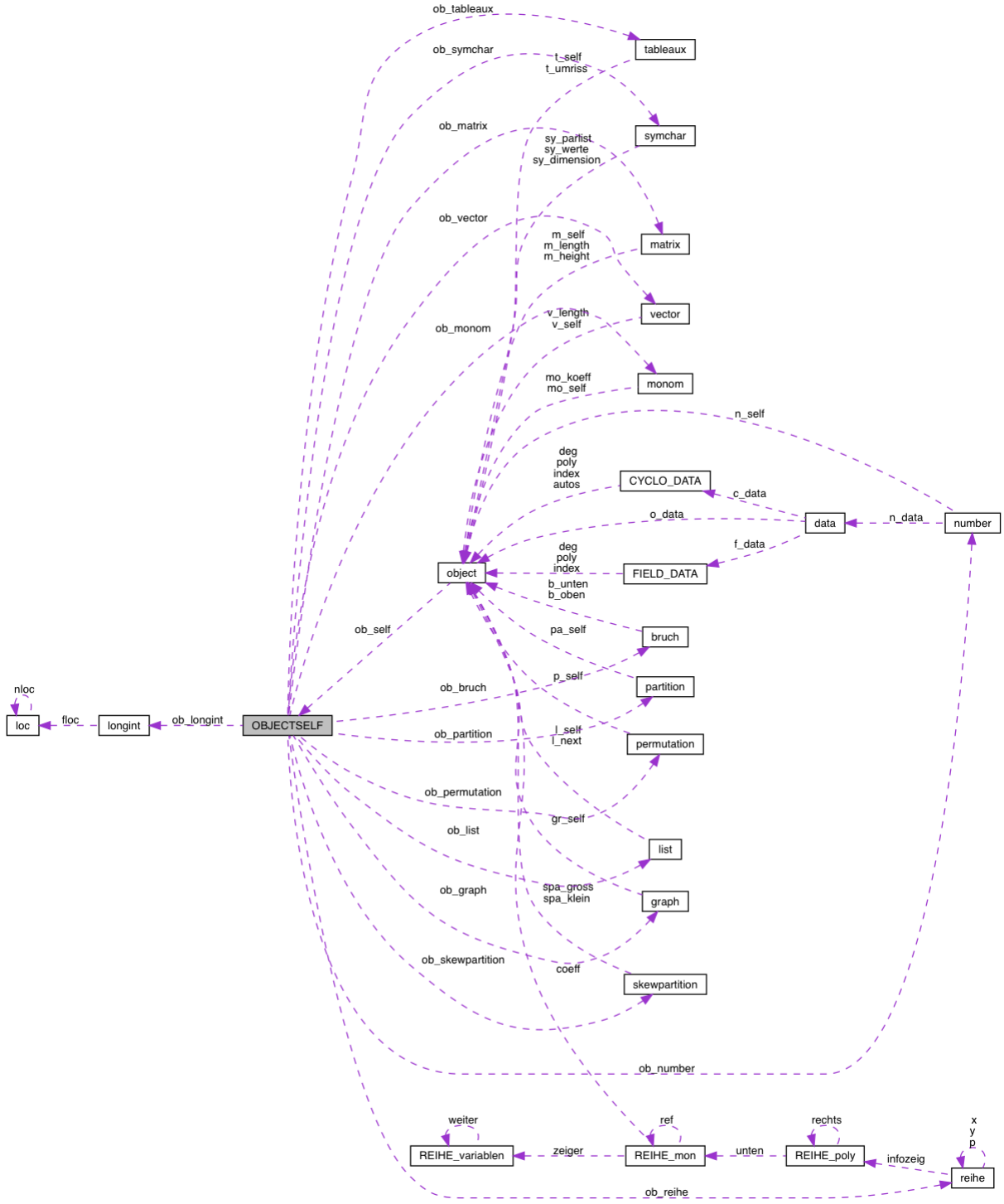


Figure 6: Collaboration graph for SYMMETRICA

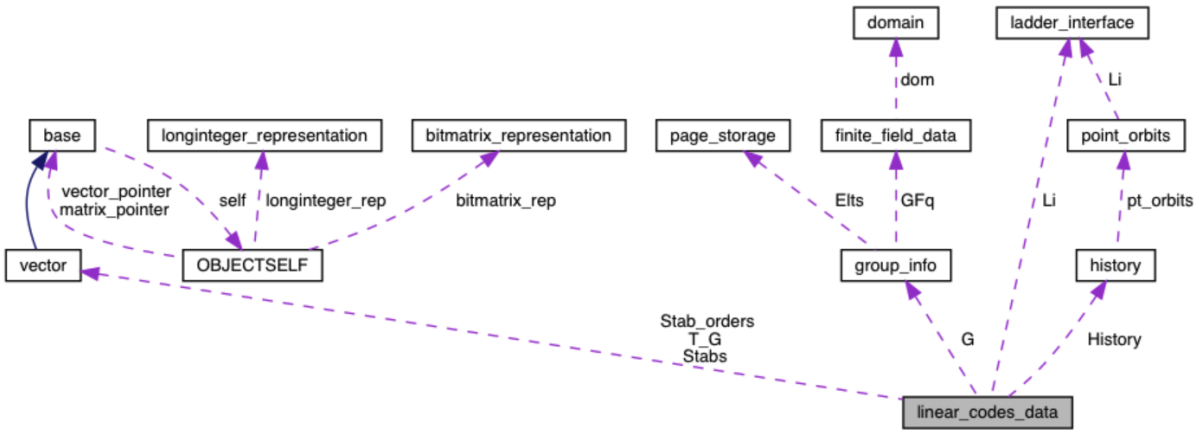


Figure 7: Collaboration graph for DISCRETA's class linear_codes_data

means of a stabilizer chain. On the right, we see several classes which were blueprints for important classes in Orbiter:

1. The class `group_info` developed into the Orbiter classes `matrix_group` and `perm_group`, implementing matrix groups and abstract permutation groups, respectively. At the time of DISCRETA, all groups were abstract permutation groups.

2. The class `page_storage` is identical to a class of the same name in Orbiter. This class provides bulk-storage for elements of a fixed group. In Orbiter, the group elements are stored in a compressed form, using a bitrepresentation of the group element. For matrix groups, this means that the matrices are encoded into bitvectors. In DISCRETA, the storage was less efficient because group elements were stored as abstract permutations.

3. The class `ladder_interface` ultimately grew into the class `action` in Orbiter. In DISCRETA, `ladder_interface` was simply a table of functions pointers which enable the abstract group action. In Orbiter, the function pointers are almost identical.

4. The class `history` was the implementation of the poset classification algorithm in DISCRETA. In orbiter, it is replaced by the classes `generator`, `poset_orbit_node` and `upstep_work`.

5. The class `point_orbits` was the implementation of Schreier trees for orbits of groups on sets. The implementation relied on a bijection between the set and an interval of integers. This class was the predecessor class of the Orbiter class `schreier`.

Let us take a closer look at Orbiter. The main concept of a symmetry group is realized in a way that resembles the object structure in SYMMETRICA. Different classes implement different kinds of groups. All classes are accessible through an object of type `symmetry group`, which is simply a union of pointers to objects of the various classes (cf. Fig. 8). The class `symmetry_group` is closely related to the class `action` which provides a framework for group

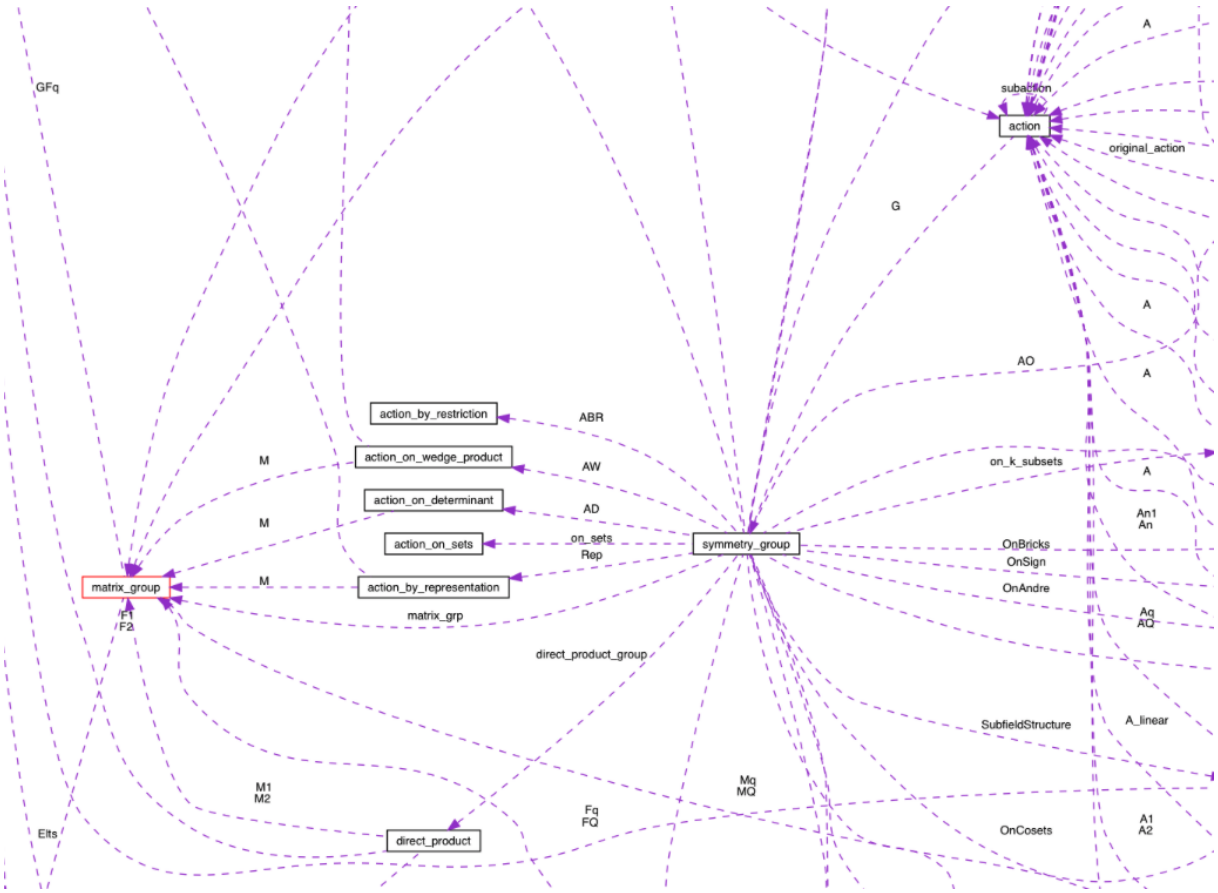


Figure 8: Collaboration graph for Orbiter's symmetry_group (cutout)

actions. A function pointer table in `action` realizes the dispatch of functions based on a type value. As mentioned earlier, the table of function pointers in `action` is almost identical to the table of function pointers in the DISCRETA class `ladder_interface`.

A Orbiter Class List

Here are the classes, structs, unions and interfaces with brief descriptions. In total, there are 214 classes and similar objects.

Table 1: Summary of Orbiter Classes.

Class	Purpose
a_domain	Related to the computation of Young representations
action	The class action implements a permutation group in a fixed action
action_by_conjugation	Action by conjugation on the elements of a given group
action_by_representation	Action of PSL(2,q) on a conic (the only type implemented so far)
action_by_restriction	Restricted action on an invariant subset
action_by_right_multiplication	Action on a the set of elements of a group by right multiplication
action_by_subfield_structure	Action on the vector space arising from a field over a subfield
action_is_minimal_data	Internal class for is_minimal backtracking used by class action
action_on_andre	Action on the elements of a projective plane constructed via Andre / Bruck / Bose
action_on_bricks	Related to a problem of Neil Sloane
action_on_cosets	Action on the cosets of a subgroup by right multiplication
action_on_determinant	Action on the determinant of a group of matrices (used to compute the subgroup PSL)
action_on_factor_space	Induced action on the factor space of a vector space modulo a subspace
action_on_flags	Action on flags
action_on_grassmannian	Action on the grassmannian (subspaces of a fixed dimension of a vectors space)
action_on_homogeneous_polynomials	Induced action on the set of homogeneous polynomials over a finite field
action_on_k_subsets	Induced action on k-subsets of a set of size n
action_on_orbits	Induced action on the set of orbits (usually by the normalizer)
action_on_orthogonal	Action on the orthogonal geometry

Continued on next page

Table 1 – continued from previous page

Class	Purpose
action_on_sets	Induced action on a given set of sets
action_on_sign	Action on the sign function of a permutation group (to compute the even subgroup)
action_on_spread_set	Induced action on a spread set via the associated spread
action_on_subgroups	Induced action on subgroups of a group
action_on_wedge_product	Wedge product action on exterior square of a vector space
andre_construction	Andre / Bruck / Bose construction of a translation plane from a spread
andre_construction_ line_element	Related to class andre_construction
andre_construction_ point_element	Related to class andre_construction
arc_generator	Poset classification for arcs in desarguesian projective planes
arc_lifting	Creates a cubic surface from a 6-arc in a plane
blt_set	Classification of BLT-sets
BLT_set_create	To create a BLT-set from a known construction
BLT_set_create_description	To describe a BLT set with a known construction from the command line
brick_domain	For a problem of Neil Sloane
bt_key	DISCRETA class for databases
btree	DISCRETA class for a database
btree_page_registry_key_pair	DISCRETA internal class related to class database
buekenhout_metz	Buekenhout Metz unitals
buffer	DISCRETA auxiliary class related to the class database
cayley_graph_search	For a problem of Ferdinand Ihringer
choose_points_or_lines	To classify objects in projective planes
classification	Poset classification data structure
classify	Statistical analysis of vectors of ints
classify_bitvectors	Stores the canonical form of 0/1 matrices for the purposes of classification
classify_double_sixes	To classify double sixes in PG(3,q)
classify_trihedral_pairs	To classify double triplets in PG(3,q)
clique_finder	A class that can be used to find cliques in graphs

Continued on next page

Table 1 – continued from previous page

Class	Purpose
code_generator	Classification of optimal linear codes over Fq
colored_graph	Graph with a vertex coloring
compute_stabilizer	Wrapper to compute the set stabilizer using the poset classification algorithm
coset_table_entry	Helper class for the poset classification algorithm
data_file	To read files of classifications from the poset classification algorithm
database	DISCRETA class for a database
datatype	DISCRETA auxiliary class related to the class database
decomposition	Decomposition of an incidence matrix
desarguesian_spread	Desarguesian spread
design_data	DISCRETA class for Kramer Mesner type problems
design_parameter	DISCRETA class for design parameters
design_parameter_source	DISCRETA class for the design parameters database
difference_set_in_heisenberg_group	Class related to a problem of Tao
diophant	For diophantine systems of equations (i.e., linear systems over the integers)
direct_product	Direct product of two matrix groups in product action
direct_product_action	Searching for line transitive point imprimitive designs preserving a grid structure on points
discreta_base	DISCRETA base class. All DISCRETA classes are derived from this class
dlx_node	Internal class for the dancing links exact cover algorithm
domain	DISCRETA class for influencing arithmetic operations
eckardt_point	Eckardt point on a cubic surface using the Schlaefli labeling
elliptic_curve	Fixed elliptic curve in Weierstrass form
exact_cover	Wrapper class for exact cover problems arising with the lifting of combinatorial objects
exact_cover_arguments	Command line arguments to control the lifting via exact cover
extension	Class representing a flag orbit
factor_group	Auxiliary class for create_factor_group, which is used in analyze_group.cpp
fancy_set	To store a subset of size k of a set of size n

Continued on next page

Table 1 – continued from previous page

Class	Purpose
ff_memory	DISCRETA auxilliary class for class domain
file_output	Wrapper class for an ofstream which allows to store extra data
finite_field	Finite field Fq
finite_ring	Finite chain rings
flag	Maximal chain of subspaces
flag_orbit_node	Related to the class flag_orbits
flag_orbits	Related to the class classification
generators_symplectic_group	Auxilliary class to construct generators of the symplectic group
geo_parameter	Input parameters for TDO-process
geometry	DISCRETA class for incidence matrices
gl_class_rep	To represent a conjugacy class of matrices in GL(n,q)
gl_classes	Conjugacy classes in GL(n,q)
graph_generator	Classification of graphs and tournaments
graph_layer	Part of the data structure layered_graph
graph_node	Part of the data structure layered_graph
grassmann	To rank and unrank subspaces of a fixed dimension in $\mathbb{F}_q^n$
grassmann_embedded	Subspaces with a fixed embedding
grid_frame	Class to help with drawing elements in a 2D grid fashion
group	Container data structure for groups
group_selection	DISCRETA class to choose a group from the command line or from a UI
hadamard	Classification of Hadamard matrices
heisenberg	The Heisenberg group of n x n matrices
hermitian	Hermitian space
hjelmslev	Hjelmslev geometry
hollerith	DISCRETA string class
homogeneous_polynomial_domain	Homogeneous polynomials in n variables over a finite field
incidence_structure	Incidence structure interface for many different types of geometries
int_matrix	Class to represent matrices over int
int_vector	Vector on ints

Continued on next page

Table 1 – continued from previous page

Class	Purpose
integer	DISCRETA integer class
isomorph	Hybrid algorithm to classify combinatorial bjects
isomorph_arguments	Helper class for isomorph
isomorph_worker_data	Auxiliary class to pass case specific data to the function isomorph_worker
itemtyp	DISCRETA auxiliary class related to the class database
job_table	Job table for the scheduler app for parallel processing
k_arc_generator	To classify k-arcs in the projective plane PG(2,q)
keycarrier	DISCRETA auxiliary class related to the class database
klein_correspondence	Klein correspondence between lines in PG(3,q) and points on the Klein quadric
knarr	Knarr construction of a GQ from a BLT-set
layered_graph	Data structure to store partially ordered sets
layered_graph_draw_options	Options for drawing an object of type layered_graph
linear_group	Used to create a linear group from command line arguments
linear_group_description	Description of a linear group from the command line
linear_set	Classification of linear sets
longinteger	DISCRETA class for integers of arbitrary magnitude
longinteger_domain	Domain to compute with objects of type longinteger
longinteger_object	Class to represent aritrary precision integers
longinteger_representation	DISCRETA internal class to represent long integers
matrix	DISCRETA matrix class
matrix_access	DISCRETA utility class for matrix access
matrix_block_data	Auxiliary class for conjugacy classes in GL(n,q)
matrix_group	A linear group implemented as matrices
mem_object_registry	Maintains a registry of allocated memory
mem_object_registry_entry	Class related to mem_object_registry
memory	DISCRETA class to serialize data structures
memory_object	Can be used for serialization
mindist	Internal class for the algorithm to compute the minimum distance of a linear code
mp_graphics	General 2D graphical output interface (metapost, tikz, postscript)
norm_tables	Tables for the norm map in a finite field

Continued on next page

Table 1 – continued from previous page

Class	Purpose
null_polarity_generator	Internal class to compute generators for the group of null polarities
number_partition	DISCRETA class for partitions of an integer
object_in_projective_space	Geometric object in projective space (points, lines or packings)
object_in_projective_space_with_action	To represent an object in projective space
OBJECTSELF	DISCRETA internal class
orbit_node	Related to the class classification
orbit_of_equations	Schreier tree for action on homogeneous equations
orbit_of_sets	Schreier tree for action on subsets
orbit_of_subspaces	Schreier tree for action on subspaces
orbit_rep	To hold one orbit after reading files from Orbiters poset classification
orbit_transversal	Container data structure for a poset classification from Orbiter output
orbiter_data_file	Class to read output files from orbiters poset classification
orthogonal	Orthogonal geometry $O^\epsilon(n, q)$
ovoid_generator	Classification of ovoids in orthogonal spaces
page_storage	Data structure to store group elements in compressed form
page_table	DISCRETA class for bulk storage
pagetyp	DISCRETA auxiliary class related to the class database
partitionstack	Leon type partitionstack class
perm_group	An abstract permutation group
permutation	DISCRETA permutation class
plane_data	Auxiliary class for the class point_line
point_line	Data structure for general projective planes, including nodesarguesian ones
polar	Orthogonal geometry as a polar space
poset_classification	Poset classification algorithm (Snakes and Ladders)
poset_orbit_node	Class representing one poset orbit, related to the class poset_classification
printing_mode	DISCRETA class related to printing of objects
product_action	Product action of two group actions

Continued on next page

Table 1 – continued from previous page

Class	Purpose
projective_space	Projective space PG(n,q) of dimension n over Fq
projective_space_with_action	Projective space PG(n,q) with automorphism group PGGL(n+1,q)
rainbow_cliques	To search for rainbow cliques in graphs
rank_checker	Used to check that any d-1 columns are linearly independent
recoordinatize	Utility class to classify spreads
regular_ls_generator	Classification of regular linear spaces
representatives	Auxiliary class for class isomorph
scene	Collection of 3D geometry objects
schreier	Schreier trees for orbits on points
schreier_sims	Schreier Sims algorithm
search_blocking_set	To classify blocking sets in projective planes
set_and_stabilizer	Set and its known set stabilizer
set_of_sets	To store a set of sets
set_stabilizer_compute	Wrapper to compute the set stabilizer with the class compute_stabilizer
sims	A stabilizer chain for a permutation group
singer_cycle	Singer cycle in PG(n-1,q)
six_arcs_not_on_a_conic	To classify six-arcs not on a conic in PG(2,q)
solid	DISCRETA class for polyhedra
solution_file_data	Internal class related to tdo_data
spread	To classify spreads of PG(k-1,q) in PG(n-1,q) where n=2*k
spread_create	To create a known spread
spread_create_description	To describe the construction of a known spread from the command line
spread_lifting	Create spreads from smaller spreads
spreadsheet	For reading and writing of csv files
strong_generators	Strong generating set for a permutation group with group order
subfield_structure	Finite field as a vector space over a subfield
subgroup	List a subgroup of a group by storing the element indices
subspace_orbits	Poset classification for orbits on subspaces
surface	Cubic surfaces in PG(3,q) with 27 lines

Continued on next page

Table 1 – continued from previous page

Class	Purpose
surface_classify_wedge	To classify cubic surfaces using double sixes as substructures
surface_create	To create a cubic surface from a known construction
surface_create_description	To describe a known construction of a cubic surface from the command line
surface_object	Particular cubic surface in PG(3,q), given by its equation
surface_object_with_action	Instance of a cubic surface together with its stabilizer
surface_with_action	Cubic surfaces in projective space with automorphism group
symmetry_group	Internal class related to action
tdo_data	Class related to the class tdo_scheme
tdo_parameter_calculation	Main class for tdo_refine to refine the parameters of a linear space
tdo_scheme	TDO scheme captures the combinatorics of a decomposed incidence structure
tensor_product	Class for the app wreath_product
translation_plane_via_andre_model	Translation plane created via Andre / Bruck / Bose
tree	Data structure for trees
tree_node	Part of the data structure tree
union_find	Union find data structure (used in the poset classification)
union_find_on_k_subsets	Union find data structure (used in the poset classification)
unipoly	DISCRETA class for poynomials in one variable
unipoly_domain	Domain to compute with polynomials in one variable over a finite field
unusual_model	Penttila's unusual model to create BLT-sets
upstep_work	Helper class for the poset classification algorithm
Vector	DISCRETA vector class for vectors of DISCRETA objects
vector_ge	Vector of group elements
vector_hashing	Hash tables
W3q	W(3,q) generalized quadrangle
with	DISCRETA class related to class domain

Continued on next page

Table 1 – continued from previous page

Class	Purpose
wreath_product	Wreath product group AGL(d,q) wreath Sym(n)
young	The Young representations of the symmetric group

B Ovoid search in $O^-(6,2)$

Here is the makefile for the ovoid search in $O^-(6,2)$:

```
1       MY_PATH=~/DEV.18
2       SRC=$(MY_PATH)/GITHUB/orbiter/ORBITER/SRC
3       SRC2=$(MY_PATH)/ORBITER2/SRC2
4
5       OVOID_PATH=$(SRC)/APPS/OVOID
6       TOOLS_PATH=$(SRC)/APPS/TOOLS
7
8
9       Op42:
10         $(OVOID_PATH)/ovoid.out -v 2 -epsilon 1 -d 4 -q 2
11
12      Om42:
13         $(OVOID_PATH)/ovoid.out -v 2 -epsilon -1 -d 4 -q 2
14
15      Op62:
16         $(OVOID_PATH)/ovoid.out -v 2 -epsilon 1 -d 6 -q 2
17
18      Om62:
19         $(OVOID_PATH)/ovoid.out -v 5 -epsilon -1 -n 5 -q 2 -draw_poset -embedded -W
20
21      # order 25920, degree 27 = U_4(2) = Weyl group of type E_6 = Sp_4(2) = O_5(3)
22
23
24      draw:
25         $(TOOLS_PATH)/layered_graph_main.out -v 4 \
26            -file ovoid_-1_6_2_poset_lvl_9.layered_graph \
27            -draw test \
28            -rad 25000 \
29            -xin 1000000 \
30            -yin 1000000 \
31            -xout 1000000 \
32            -yout 1000000 \
33            -embedded \
34            -scale .44 \
35            -line_width 1.0 \
36            -y_stretch 0.8
37         pdflatex ovoid_-1_6_2_poset_lvl_9_draw.tex
38         open ovoid_-1_6_2_poset_lvl_9_draw.pdf
39
40
```

C The class action

In Figur 9, the collaboration graph for the class action is shown.

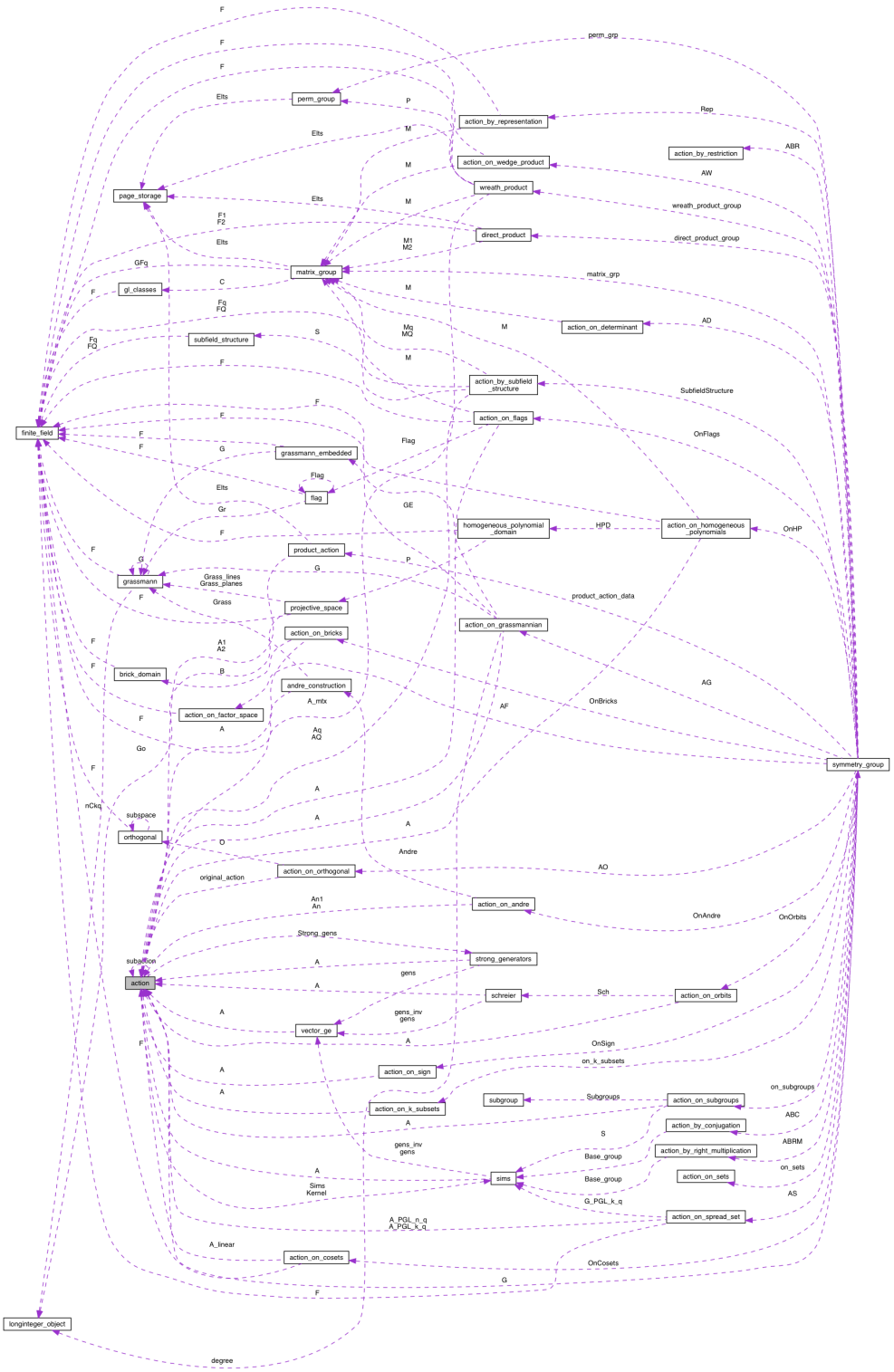


Figure 9: The collaboration graph for the action class

References

[1] Anton Betten. Classifying discrete objects with Orbiter. ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013.

[2] Anton Betten. Orbiter – a program to classify discrete objects. `https://github.com/abetten/orbiter`, 2013-2018.

[3] Anton Betten. How fast can we compute orbits of groups? In *ICMS 2018—Proceedings of the International Congress on Mathematical Software; James H. Davenport, Manuel Kauers, George Labahn, Josef Urban (ed.)*, pages 62–70. Springer, 2018.

[4] Anton Betten, Evi Haberberger, Reinhard Laue, and Alfred Wassermann. Discerta – a system to construct t-designs with prescibed automorphism group. `http://www.algorithm.uni-bayreuth.de/en/research/discreta/index.html`, circa 1996-2004.

[5] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*, volume 290 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, New York, third edition, 1999. With additional contributions by E. Bannai, R. E. Borcherds, J. Leech, S. P. Norton, A. M. Odlyzko, R. A. Parker, L. Queen and B. B. Venkov.

[6] Cygwin. A Unix-like environment and command-line interface for Microsoft Windows. `https://www.cygwin.com`.

[7] Valgrind Developers. Valgrind `http://valgrind.org`.

[8] I. A. Faradžev. Constructive enumeration of combinatorial objects. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloq. Internat. CNRS*, pages 131–135. CNRS, Paris, 1978.

[9] P. Kaski and P. Östergård. *Classification algorithms for codes and designs*, volume 15 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2006.

[10] A. Kerber and A. Kohnert. SYMMETRICA, an object oriented computer algebra system for representations, combinatorics and applications of symmetric groups. In *Symmetry and structural properties of condensed matter (Zajaczkowo, 1994)*, pages 494–503. World Sci. Publ., River Edge, NJ, 1995.

[11] Adalbert Kerber and Axel Kohnert. Modular irreducible representations of the symmetric group as linear codes. *European J. Combin.*, 25(8):1285–1299, 2004.

[12] Adalbert Kerber, Axel Kohnert, and Alain Lascoux. SYMMETRICA, an object oriented computer-algebra system for the symmetric group. *J. Symbolic Comput.*, 14(2-3):195–203, 1992.

[13] Jeffrey S. Leon. Permutation group algorithms based on partitions. I. Theory and algorithms. *J. Symbolic Comput.*, 12(4-5):533–583, 1991. Computational group theory, Part 2.

[14] Jeffrey S. Leon. Partitions, refinements, and permutation group computation. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 123–158. Amer. Math. Soc., Providence, RI, 1997.

[15] Brendan D. McKay. Isomorph-free exhaustive generation. *J. Algorithms*, 26(2):306–324, 1998.

[16] Wilhelm Plesken. Counting with groups and rings. *J. Reine Angew. Math.*, 334:40–68, 1982.

[17] Ronald C. Read. Every one a winner or how to avoid isomorphism search when cataloguing combinatorial configurations. *Ann. Discrete Math.*, 2:107–120, 1978. Algorithmic aspects of combinatorics (Conf., Vancouver Island, B.C., 1976).

[18] Gordon F. Royle. An orderly algorithm and some applications in finite geometry. *Discrete Math.*, 185(1-3):105–115, 1998.

[19] B. Schmalz. t-Designs zu vorgegebener Automorphismengruppe. *Bayreuth. Math. Schr.*, 41:1–164, 1992. Dissertation, Universität Bayreuth, Bayreuth, 1992.

[20] Bernd Schmalz. Verwendung von Untergruppenleitern zur Bestimmung von Doppelnebenklassen. *Bayreuth. Math. Schr.*, (31):109–143, 1990.