Orbiter User's Guide

Anton Betten

February 10, 2019

Abstract

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

1 Introduction

Orbiter [1] [2] is a software package for the classification of combinatorial objects, written in C++. Orbiter consists of a large library of objects related to combinatorics, algebra, group theory and geometry. A number of applications come bundled with Orbiter, many of which are concernde with the classification or combinatorial objects: optimal linear codes, cubic surfaces with 27 lines over a finite field, and other objects in geometry. There are applications related to graph theory, all the way from Cayley graphs to distance regular graphs to clique finding algorithms and to algorithms for the classification of small graphs. Many of the algorithms in Orbiter allow for parallel computing. At this point Orbiter does not have a user interface. As a substitute, the bundled applications can be invoked using command line arguments. These commands can be collected in shell scripts or makefiles. While this is perhaps a bit uncomfortable for the novice user, it has the advantage that the computations are documented. Also, the results are typically written to file, waiting to be processed by other programs. It is also possible to write new applications in Orbiter. After all, this is what a class library is supposed to be good for. For this, a slightly higher level of familiarity with the conventions and data structures in Orbiter is required. Documentation of Orbiter is available on a web site, created using Doxygen. Any Orbiter application is developed using the Orbiter library (liborbiter.a), and has to be linked with the C++ standard library (cf. Figure 1). The Orbiter library is layered. There are five separate namespaces, forming a hierarchy. This is shown in Figure 2.

2 Orbiter and Group Theory

The classifiction of combinatorial objects requires a group whose orbits correspond to the objects under consideration. For this reason, Orbiter provides finite groups and their group actions. Certain groups are provided with a standard action. New actions can be built from these using various constructions. The groups in Orbiter fall into four categories:

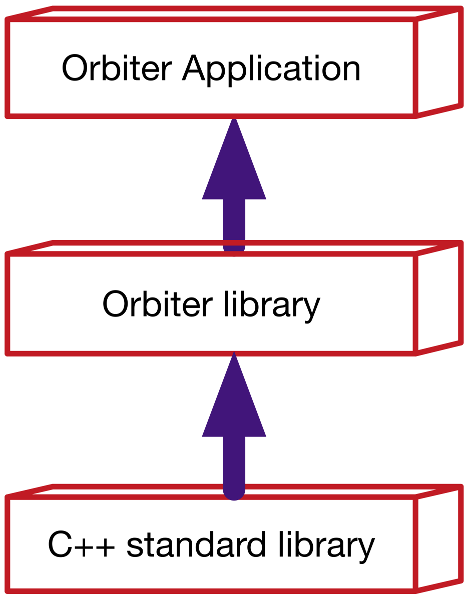

Figure 1: The orbiter model

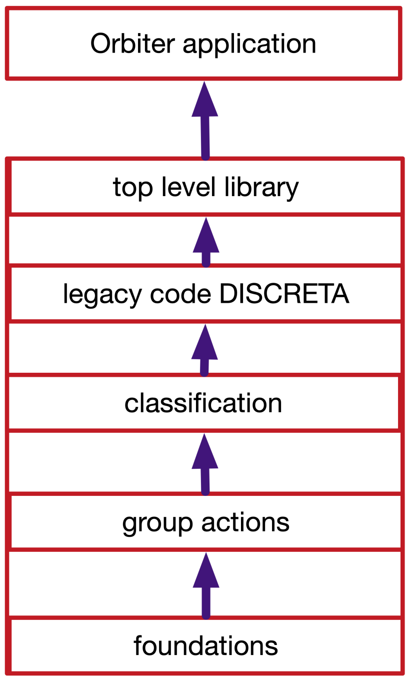

Figure 2: The namespaces of the orbiter library

Command	Arguments	Group
-GL	n, q	$\mathrm{GL}(n,q)$
-GGL	n, q	$\Gamma\mathrm{L}(n,q)$
-SL	n, q	$\mathrm{SL}(n,q)$
-SSL	n, q	$\Sigma\mathrm{L}(n,q)$
-PGL	n, q	$\mathrm{PGL}(n,q)$
-PGGL	n, q	$\mathrm{P}\Gamma\mathrm{L}(n,q)$
-PSL	n, q	$\mathrm{PSL}(n,q)$
-PSSL	n, q	$\mathrm{P}\Sigma\mathrm{L}(n,q)$
-AGL	n, q	$\mathrm{AGL}(n,q)$
-AGGL	n, q	$\mathrm{A}\Gamma\mathrm{L}(n,q)$
-ASL	n, q	$\mathrm{ASL}(n,q)$
-ASSL	n, q	$\mathrm{A}\Sigma\mathrm{L}(n,q)$

Table 1: Basic types of Orbiter matrix groups

(a) Matrix groups. These groups act linearly or semilinearly on a vector space. There are three types of group actions to be considered: projective groups, affine groups and general linear groups. Various other types exist, such as orthogonal groups and unitary groups. The elements of matrix groups are stored as matrices, possibly extended by a translation vector or a field automorphism.

(b) Permutation groups. These are groups of permutations of a finite set. The mappings are stored by listing each image, using a vector data structure.

(c) Direct product type.

(d) Wreath product type $\mathrm{GL}(n,q) \wr \mathrm{Sym}(n)$.

3 Matrix Groups

The command

`-linear`⟨arguments⟩⟨modifier⟩`-end`

can be used to select a matrix group. The arguments can be one of the commands in Table 1 (including two numerical values for n and q, respectively). The executable `linear_group.out` can be used to create a matrix group. For instance

```
linear_group.out -v 3 -linear -PGL 2 11 -end
```

creates $\mathrm{PGL}(2,11)$ in the action on the 12 points of the projective line. The option `-v` $\langle k\rangle$ can be used to specify the verbosity of the command. Higher values of k lead to more text

Modifier	Arguments	Meaning
-wedge		action on the exterior square
-PGL2OnConic		induced action of $\mathrm{PGL}(2,q)$ on the conic in the plane $\mathrm{PG}(2,q)$
-monomial		subgroup of monomial matrices
-diagonal		subgroup of diagonal matrices
-null_polarity_group		null polarity group
-symplectic_group		symplectic group
-singer	k	subgroup of index k in the Singer cycle
-singer_and_frobenius	k	subgroup of index k in the Singer cycle, extended by the Frobenius automorphism of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$
-subfield_structure_action	s	action by field reduction to the subfield of index s
-subgroup_from_file	f l	read subgroup from file f and give it the label l
-borel_subgroup_upper		Borel subgroup of upper triangular matrices
-borel_subgroup_lower		Borel subgroup of lower triangular matrices
-identity_group		identity subgroup
-on_k_subspaces	k	induced action on k dimensional subspaces
-orthogonal	ϵ	orthogonal group O^ϵ, with $\epsilon \in \{\pm 1\}$ when n is even

Table 2: Modifiers for creating matrix groups

output. A verbosity $k = 0$ means no or almost no text output. If the verbosity is positive, the program will print the chosen generators, as well as a list of points for the permutation representation.

The modifier can be any command from Table 2. For instance, the command

```
linear_group.out -v 3 -linear -PGL 3 11 -singer_and_frobenius 190 -end
```

creates a group of order 21 which is formed by taking the subgroup of order 7 in the Singer cycle of $\mathrm{PGL}(3,11)$ extended by the Frobenius automorphism of $\mathbb{F}_{11^3}$ over $\mathbb{F}_{11}$ of order 3. In order to create the subgroup of order 7, the generator of the Singer cycle is considered, of order $11^3 - 1 = 1330$. Since $1330 = 190 \cdot 7$, raising the generator to the power of 190 creates an element of order 7. The group generated acts on the $11^2 + 11 + 1 = 133$ points of $\mathrm{PG}(2,11)$. Thus, a matrix group of order 21 is generated with a permutation action of degree 133. The matrix generators are

$$\begin{bmatrix} 1 & 2 & 6 \\ 9 & 6 & 2 \\ 3 & 7 & 6 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 7 & 7 \\ 5 & 6 & 3 \end{bmatrix}.$$

The first matrix is the element of order 7 arising from the Singer cycle. The second matrix arises from the Frobenius automorphism.

4 Classification of Objects

Orbiter offers several algorithms to classify combinatorial objects. A poset classification algorithm can be used to classify posets under a group action. Two distinct strategies are offered. The first one is canonical augmentation, following McKay [8]. This strategy relies heavily on the software package Nauty [9], which is included in Orbiter. The second algorithm is based on Schmalz [10]. The main difference is that the McKay-Nauty approach classifies the poset in a depth-first manner, while Schmalz used breadth-first. The bottleneck in the McKay-Nauty approach is a function to compute the canonical form of a graph. Such a function is not necessary in the Schmalz approach. However, the Schmalz algorithm requires a lot more storage. It stores all orbit representatives, and certain group elements which establish isomorphisms between objects that are called flag-orbits. A more detailed description of a modernized version of the Schmalz algorithm can be found in [4] and [3].

4.1 Direct Classification of Objects using Nauty

Objects in projective space can be classified using the Orbiter-Nauty interface. There is no need to learn Nauty, as all commands run through Orbiter. Orbiter will issue the appropriate calls to Nauty internally. For instance, it is possible to classify a set of sets in a projective plane, and to compute the stabilizer groups of the orbit representatives. The stabilizer groups are generated as matrix groups. The role of Orbiter is to take in the objects that are to be classified, to perform the classification using Nauty, and to return the classified list of pairwise non-isomophic objects together with their stabilizers. Internally, Orbiter creates one graph for each object and hands this graph to Nauty. Nauty computes the canonical form of the graph as well as generators for the automorphism group. Orbiter then converts the automorphism group of the graph into the corresponding matrix group of the projective space. The graph that is created is a Levi graph. It is a bipartite graph, with one sets of vertices for all points of the geometry, and one set of vertices for all lines. A few additional vertices of each type are created to encode the combinatorial object.

Let us consider an example. Suppose we are interested in elliptic curves over small finite fields. The Hasse-Weil-Serre bound tells us that

$$\#C(\mathbb{F}_q) \leq q + 1 + \lfloor 2\sqrt{q} \rfloor,$$

h	monomial	vector
0	X^3	$(3,0,0)$
1	Y^3	$(0,3,0)$
2	Z^3	$(0,0,3)$
3	X^2Y	$(2,1,0)$
4	X^2Z	$(2,0,1)$
5	XY^2	$(1,2,0)$
6	Y^2Z	$(0,2,1)$
7	XZ^2	$(1,0,2)$
8	YZ^2	$(0,1,2)$
9	XYZ	$(1,1,1)$

Table 3: Orbiter ordering of cubic monomials in 3 variables

where C is an elliptiv curve and $\#C(\mathbb{F}_q)$ is the number of points of C in $\mathrm{PG}(2,q)$. Maximal elliptic curves are those for which equality holds. For instance, Soomro [11] lists examples of such curves over small fields. Suppose we are interested in a maximal elliptic curve in the plane $\mathrm{PG}(2,11)$. According to Soomro, the equation

$$y^2 = x^3 + x + 3$$

defines such a curve with 18 points. In order to create this curve using Orbiter, we need to create the algebraic set. For this purpose, we need to establish the homogeneous equation of the curve. Substituting

$$x = \frac{X}{Z},\ y = \frac{Y}{Z}$$

and clearing denominators yields

$$Y^2Z = X^3 + XZ^2 + 3Z^3.$$

To encode the equation, Orbiter uses a numeric indexing of the monomials. The numerical index h of each monomial is listed in Table 3. We rewrite the homogeneous equation as

$$X^3 + XZ^2 + 3Z^3 + 10Y^2Z = 0.$$

This homogeneous equation is then translated into the partial mapping from the index set of monomials to the field elements:

$$0 \mapsto 1,\ 7 \mapsto 1,\ 2 \mapsto 3,\ 6 \mapsto 10.$$

Here, it is important that the field elements are encoded using integers k with $0 \le k < q$ (so, in particular, encoding the coefficient of Y^2Z as -1 would be a problem for Orbiter). The partial mapping is encoded as the following set of pairs

$$(1,0),\ (1,7),\ (3,2),\ (10,6),$$

i	a_i	P_{a_i}	i	a_i	P_{a_i}
0	1	$(0,1,0)$	9	67	$(0,5,1)$
1	16	$(3,0,1)$	10	78	$(0,6,1)$
2	28	$(5,1,1)$	11	90	$(1,7,1)$
3	30	$(7,1,1)$	12	93	$(4,7,1)$
4	33	$(10,1,1)$	13	95	$(6,7,1)$
5	43	$(9,2,1)$	14	120	$(9,9,1)$
6	57	$(1,4,1)$	15	127	$(5,10,1)$
7	60	$(4,4,1)$	16	129	$(7,10,1)$
8	62	$(6,4,1)$	17	132	$(10,10,1)$

Table 4: The $\mathbb{F}_{11}$ rational points of $y^2 = x^3 + x + 3$

which is then concatenated into an even-length vector

$$(1,0,1,7,3,2,10,6).$$

The Orbiter command

```
create_object.out -v 2 -q 11 -n 2 \
    -projective_variety "elliptic_curve_q11"
    3 "1,0,10,6,1,7,3,2"
```

is issued. It computes the projective variety determined by the equation and creates a file `elliptic_curve_q11.txt` (The filename is taken from the command line.) We find that there are 18 $\mathbb{F}_{11}$ rational points, shown in Table 4. The next step is to create the automorphism group of this curve. For this, the command

```
canonical_form.out -v 2 \
 -q 11 -n 2 \
 -input -file_of_point_set \
 elliptic_curve_q11.txt -end \
 -classify_nauty \
 -prefix elliptic_curve_q11 -latex
```

is issued. This command produces a latex file with information about the curve. For instance, we can see that the automorphism group of the curve has order 6 and is generated by

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 8 \\ 5 & 9 & 5 \\ 8 & 1 & 1 \end{bmatrix}.$$

In addition to the group, the report produced by Orbiter shows tactical decompositions of the extended incidence matrix of the geometry. The extended incidence matrix is the point-line incidence matrix of the underlying projective space, extended by an additional point and an additional line. The additional line is incident with the points in the object and with the additional point. The reason for extending the incidence matrix is to allow the communicate the combinatorial object to the graph canonication algorithm Nauty. The presence of the incidence matrix means that the stricture of the geometry is encoded in the graph. The presence of the additional point and line encodes the object.

After the canonical form and the automorphism group have been computed, Orbiter computes a canonical decomposition of the extended geometry. This decomposition is preserved by the automorphism group, though it is not guaranteed that the decomposition is equal to the decomposition by the orbits of the automorphism group. The way that this decomposition is computed is by looking at the multiplicities with which elements of a point class are incident with elements of column classes. Likewise, we also consider the multiplicities with which elements of a column class are incident with elements of a row class. The partition is refined whenever there are two element in one class which can be distinguished by their incidence pattern. In this case, we find

$\rightarrow$	133_1	1_2
18_0	12	1
115_4	12	0
1_3	0	1

and

$\downarrow$	46_1	15_6	48_5	24_7	1_2
18_0	3	2	1	0	18
115_4	9	10	11	12	0
1_3	0	0	0	0	1

The first decomposition matrix shows a row partition of type $18 + 115 + 1$. The 18 points are the points on the curve, and the 115 points are the points off the curve. The final 1 is a auxiliary point, created in order to encode the object. The column partition is of type $133 + 1$, which recordes the fact that there are 133 lines in the geometry $\mathrm{PG}(2, 11)$, and that there is one special line that was added to encode the object. The arrow to the right in the top-left corner indicated that the entries in the decomposition matrix count the multiplicities of incidences of an element in a row with elements in each of the column classes. The second decomposition matrix is obtained by refining this decomposition along

h	monomial	vector	h	monomial	vector
0	X_0^3	$(3,0,0,0)$	10	$X_0X_2^2$	$(1,0,2,0)$
1	X_1^3	$(0,3,0,0)$	11	$X_1X_2^2$	$(0,1,2,0)$
2	X_2^3	$(0,0,3,0)$	12	$X_2^2X_3$	$(0,0,2,1)$
3	X_3^3	$(0,0,0,3)$	13	$X_0X_3^2$	$(1,0,0,2)$
4	$X_0^2X_1$	$(2,1,0,0)$	14	$X_1X_3^2$	$(0,1,0,2)$
5	$X_0^2X_2$	$(2,0,1,0)$	15	$X_2X_3^2$	$(0,0,1,2)$
6	$X_0^2X_3$	$(2,0,0,1)$	16	$X_0X_1X_2$	$(1,1,1,0)$
7	$X_0X_1^2$	$(1,2,0,0)$	17	$X_0X_1X_3$	$(1,1,0,1)$
8	$X_1^2X_2$	$(0,2,1,0)$	18	$X_0X_2X_3$	$(1,0,1,1)$
9	$X_1^2X_3$	$(0,2,0,1)$	19	$X_1X_2X_3$	$(0,1,1,1)$

Table 5: Orbiter ordering of cubic monomials in 4 variables

columns. The column partition splits into $46 + 15 + 48 + 24 + 1$. The top left corner shows an arrow pointing downwards. This signifies that the entries in the decomposition matrix are the multiplicities in which an element of a column class is incident with elements in each of the row-classes. So, for instance, the first 46 lines are trisecants, as they intersect the elliptic curve in 18 points. All told, the decomposition shows that there are 46 trisecant lines, 15 bisecants, 48 tangents and 24 external lines. The subscripts in the row and column partitions refer to the actual classes in the partition. This way, it is possible to find out the exact number of the points (or lines) which comprise one class. For this, additional output created by Orbiter is used which is not shown here.

As a second example, consider cubic surfaces over finite fields. The Dickson-Hirschfeld surface (cf. [6],[7]) has the equation

$$X_0^2X_3 + X_1^2X_2 + X_1X_2^2 + X_0X_3^2 = 0.$$

Using the monomial ordering as shown in Table 5, this translates into a partial mapping like so

$$6 \mapsto 1,\ 8 \mapsto 1,\ 11 \mapsto 1,\ 13 \mapsto 1.$$

The corresponding pairs are

$$(1,6),\ (1,8),\ (1,11),\ (1,13),$$

which then translate into the vector

$$(1,6,1,8,1,11,1,13).$$

The orbiter command

```
create_object.out -v 2 -q 4 -n 3 \
   -projective_variety "DH_surface_q4" 3 "1,6,1,8,1,11,1,13"
```

creates the variety of 45 points, and stores the set of points in the file `DH_surface_q4.txt`. Next, the command

```
canonical_form.out -v 2 \
    -q 4 -n 3 \
    -input -file_of_point_set \
        DH_surface_q4.txt -end \
    -classify_nauty \
    -prefix DH_surface_q4 -latex
```

computes the automorphism group of the surface, which has order 51840 and is generated by

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}_1, \begin{bmatrix}1&0&0&0\\0&\alpha&0&0\\0&0&\alpha&0\\0&0&0&1\end{bmatrix}_0, \begin{bmatrix}1&0&0&0\\0&\alpha&0&0\\0&0&\alpha&0\\1&0&0&1\end{bmatrix}_1,$$

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&1&1&0\\0&0&0&1\end{bmatrix}_0, \begin{bmatrix}1&0&0&0\\0&1&0&0\\1&0&1&0\\1&1&0&1\end{bmatrix}_1, \begin{bmatrix}1&0&0&0\\0&1&1&0\\1&1&0&0\\1&1&1&1\end{bmatrix}_1, \begin{bmatrix}0&1&0&0\\1&0&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}_0$$

The refinement of decomposition schemes is

$\rightarrow$	357_1	1_2
86_0	21	1

$\downarrow$	357_1	1_2
86_0	5	46

$\rightarrow$	357_1	1_2
45_0	21	1
40_4	21	0
1_3	0	1

$\downarrow$	27_1	240_5	90_6	1_2
45_0	5	3	1	45
40_4	0	2	4	0
1_3	0	0	0	1

$\rightarrow$	27_1	240_5	90_6	1_2
45_0	3	16	2	1
40_4	0	12	9	0
1_3	0	0	0	1

$\downarrow$	27_1	240_5	90_6	1_2
45_0	5	3	1	45
40_4	0	2	4	0
1_3	0	0	0	1

This shows that the surface has 45 points and 27 lines, and each point lies on exactly three lines (such points are called Eckardt points).

4.2 Poset Classification

Suppose we want to classify the subspaces in $\mathrm{PG}(3,2)$ under the action of the orthogonal group. The orthogonal group is the stabilizer of a quadric. In $\mathrm{PG}(3,2)$ there are two disting nondegenerate quadrics, $\mathcal{Q}^+(3,2)$ and $\mathcal{Q}^-(3,2)$. The $\mathcal{Q}^+(3,2)$ quadric is a finite version of the quadric given by the equation

$$x_0x_1 + x_2x_3 = 0,$$

and depicted over the real numbers in Figure 3. $\mathrm{PG}(3,2)$ has 15 points:

$P_0 = (1,0,0,0)$	$P_4 = (1,1,1,1)$	$P_8 = (1,1,1,0)$	$P_{12} = (0,0,1,1)$
$P_1 = (0,1,0,0)$	$P_5 = (1,1,0,0)$	$P_9 = (1,0,0,1)$	$P_{13} = (1,0,1,1)$
$P_2 = (0,0,1,0)$	$P_6 = (1,0,1,0)$	$P_{10} = (0,1,0,1)$	$P_{14} = (0,1,1,1)$
$P_3 = (0,0,0,1)$	$P_7 = (0,1,1,0)$	$P_{11} = (1,1,0,1)$	

The $\mathcal{Q}^+(3,2)$ quadric fiven by the equation above consists of the nine points

$$P_0, P_1, P_2, P_3, P_4, P_6, P_7, P_9, P_{10}.$$

The quadric is stabilized by the group $\mathrm{PGO}^+(4,2)$ of order 72. The command

```
subspace_orbits_main.out -v 5 \
    -depth 4 -group -PGL 4 2 -orthogonal 1 -end \
    -draw_poset -embedded \
```

produces a classification of all subspaces of $\mathrm{PG}(3,2)$ under $\mathrm{PGO}^+(4,2)$. A Hasse diagram of the classification is shown in Figure 4. Let us try to understand this output a little bit. Every

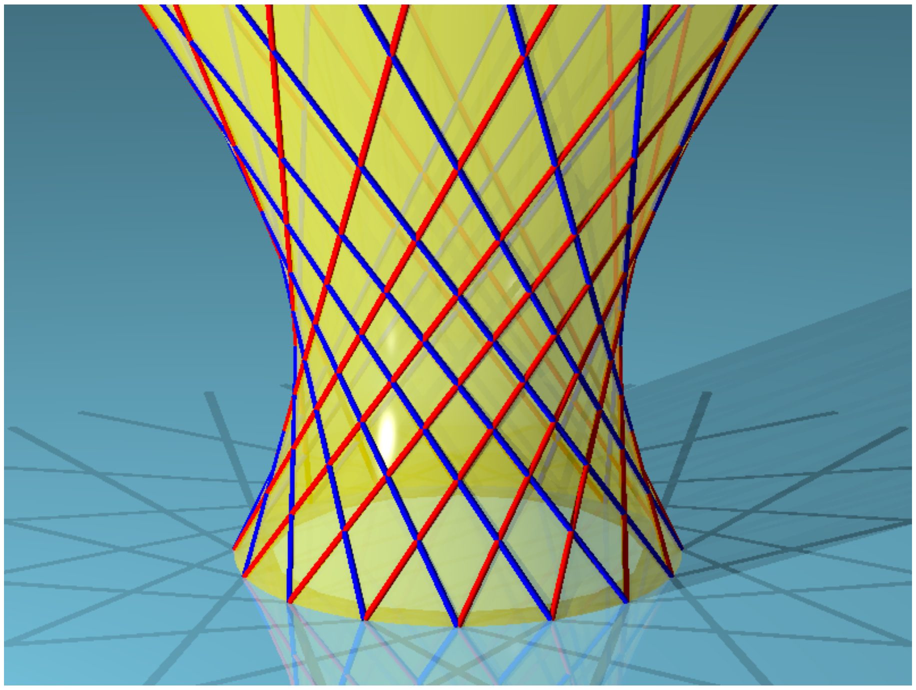

Figure 3: The hyperbolic quadric over $\mathbb{R}$

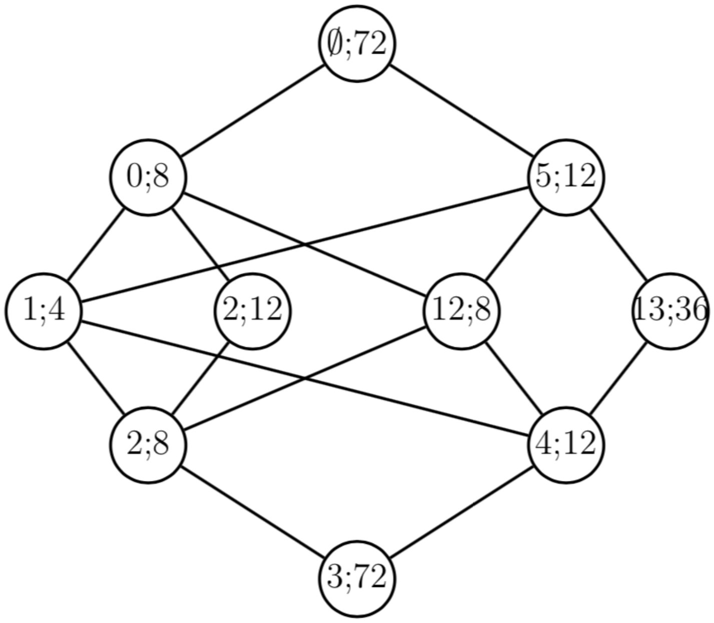

Figure 4: Hasse-diagram of the types of subspaces of $\mathrm{PG}(3,2)$

node stands for one isomorphism class of orbits of the orthogonal group on subspaces. The number before the semicolon refers to the orbit representative at that node. The number after the semicolon gives the order of the stabilizer of the node. The node at the top represents the zero subspace, with a stabilizer of order 72 (the full group). Every node below this represents a non-trivial subspace. Each subspace is described using the numerical representation of the basis elements, according to the labeling of points that was given above. In order to make the presentation more compact, only the index of the last of the basis vectors is listed at each node. The other basis vectors can be recovered by following the leftmost path to the root. For instance, the node at the very bottom is labeled by 3, representing P_3. The other basis elements are P_0, P_1, P_2 because $0, 1, 2$ are the labels encountered along the unique leftmost path to the root. Since $P_0, \ldots, P_3$ represent the four unit vectors, it is clear that the bottom node represents the whole space $\mathrm{PG}(3, 2)$. The stabilizer is the full group, of order 72. The two nodes at level one represent the two types of points. P_0 represents points on the quadric (with a point stabilizer of order 9), and P_5 represents the points off the quadric (with a point stabilizer of order 12). The middle node has 4 orbits. Reading left to right, these nodes represent the following orbits on lines:

(a) Secant lines. Such lines have two points on the quadric and $q - 1$ points off the quadric. A representative is the line P_0P_1. These lines give rise to hyperbolic pairs.

(b) Totally isotropic lines. These are lines contained in the quadric (these correspond to the colored lines in Fig. 3). A representative is the line P_0P_2.

(c) Tangent lines. Such lines have exactly one point on the quadric. A representative is the line P_0P_{12}.

(d) External lines. Such lines contain no quadric point. A representative is the line P_5P_{13}.

There are two types of planes:

(a) Planes which intersect the quadric in two totally isotropic lines. A representative is the plane $P_0P_1P_2$.

(b) Planes which intersect the quadric in a conic. A representative is the plane $P_0P_1P_4$.

5 Installing Orbiter

Orbiter is available on 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` [5]. Cygwin is a Unix-like environment and command-line interface for Microsoft Windows.

- 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.

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

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.

6 Acknowledgements

Nauty is due to Brendan McKay from the Australian National University. The Orbiter-Nauty interface is joint work with Abdullah AlAzemi from the University of Kuwait.

A Orbiter Class List

Here are the classes, structs, unions and interfaces with brief descriptions. In total, there are over 200 classes and similar objects in Orbiter. They are spread over 5 namespaces: foundations, group actions, classification, toplevel, and DISCRETA.

A.1 Namespace Foundations

Table 6: Orbiter Namespace Foundations.

Class	Purpose
a_domain	related to the computation of Young representations
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_lifting_with_two_lines	creates a cubic surface from a 6-arc in a plane
brick_domain	a problem of Neil Sloane
buekenhout_metz	Buekenhout Metz unitals.
classify	a statistical analysis of vectors of ints
classify_bitvectors	classification of 0/1 matrices using canonical forms
clique_finder	A class that can be used to find cliques in graphs.
clique_finder_control	a class that controls the clique finding process
colored_graph	a graph with a vertex coloring
data_file	to read data files from the poset classification algorithm
decomposition	decomposition of an incidence matrix
desarguesian_spread	desarguesian spread
diophant	diophantine systems of equations (i.e., linear systems over the integers)
dlx_node	internal class for the dancing links exact cover algorithm
eckardt_point	Eckardt point on a cubic surface using the Schlaefli labeling.
	Continued on next page

Table 6 – continued from previous page

Class	Purpose
eckardt_point_info	information about the Eckardt points of a surface derived from a six-arc
elliptic_curve	a fixed elliptic curve in Weierstrass form
fancy_set	subset of size k of a set of size n
file_output	a wrapper class for an ofstream which allows to store extra data
finite_field	finite field $\mathbb{F}_q^n$
finite_ring	finite chain rings
flag	a maximal chain of subspaces
generators_symplectic_group	generators of the symplectic group
geo_parameter	decomposition stack of a linear space or incidence geometry
gl_class_rep	conjugacy class in $\mathrm{GL}(n, q)$ described using rational normal form
gl_classes	to list all conjugacy classes in $\mathrm{GL}(n, q)$
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	a class to help with drawing elements in a 2D grid fashion
heisenberg	Heisenberg group of nxn matrices.
hermitian	hermitian space
hjelmslev	Hjelmslev geometry.
homogeneous_polynomial_domain	homogeneous polynomials in n variables over a finite field
incidence_structure	interface for various incidence geometries
int_matrix	matrices over int
int_vector	vector on ints
klein_correspondence	the Klein correspondence between lines in $\mathrm{PG}(3, q)$ and points on the Klein quadric
knarr	the Knarr conion of a GQ from a BLT-set
layered_graph	a data structure to store partially ordered sets

Continued on next page

Table 6 – continued from previous page

Class	Purpose
layered_graph_draw_options	options for drawing an object of type layered_graph
longinteger_domain	domain to compute with objects of type longinteger
longinteger_object	a class to represent aritrary precision integers
matrix_block_data	rational normal form of a matrix in $\mathrm{GL}(n,q)$ for gl_class_rep
mem_object_registry	maintains a registry of allocated memory
mem_object_registry_entry	a class related to mem_object_registry
memory_object	for serialization of complex data types
mindist	internal class for the algorithm to compute the minimum distance of a linear code
mp_graphics	a general 2D graphical output interface (metapost, tikz, postscript)
norm_tables	tables for the norm map in a finite field
null_polarity_generator	all null polarities
object_in_projective_space	a geometric object in projective space (points, lines or packings)
orbiter_data_file	read output files from the poset classification
orthogonal	an orthogonal geometry $O^{\epsilon}(n,q)$
page_storage	bulk storage of group elements in compressed form
partitionstack	partitionstack for set partitions following Jeffrey Leon
plane_data	auxiliary class for the class point_line
point_line	a data ure for general projective planes, including nodesarguesian ones
projective_space	a projective space $\mathrm{PG}(n,q)$ of dimension n over $\mathbb{F}_q$
rainbow_cliques	to search for rainbow cliques in graphs
rank_checker	checking whether any $d-1$ columns are linearly independent
scene	a collection of 3D geometry objects
set_of_sets	set of sets
solution_file_data	internal class related to tdo_data
spreadsheet	for reading and writing of csv files

Continued on next page

Table 6 – continued from previous page

Class	Purpose
subfield_structure	a finite field as a vector space over a subfield
surface	cubic surfaces in $\mathrm{PG}(3,q)$ with 27 lines
surface_object	a particular cubic surface in $\mathrm{PG}(3,q)$, given by its equation
tdo_data	a class related to the class tdo_scheme
tdo_scheme	canonical tactical decomposition of an incidence ure
tree	a data strucure for trees
tree_node	part of the data structure tree
unipoly_domain	domain of polynomials in one variable over a finite field
unusual_model	Penttila's unusual model to create BLT-sets.
vector_hashing	hash tables
vector_space	finite dimensional vector space over a finite field
W3q	a $\mathrm{W}(3,q)$ generalized quadrangle

A.2 Namespace Group Actions

Table 7: Orbiter Namespace Group Actions.

Class	Purpose
action	a permutation group in a fixed action.
action_by_conjugation	action by conjugation on the elements of a given group
action_by_representation	the action of $\mathrm{PSL}(2,q)$ on a conic
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

Continued on next page

Table 7 – continued from previous page

Class	Purpose
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
action_on_set_partitions	induced action on a set partitions.
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	the wedge product action on exterior square of a vector space
action_pointer_table	interface to the implementation functions for group actions
data_input_stream	description of input data for classification of geometric objects from the command line
direct_product	the direct product of two matrix groups in product action
group	a container data structure for groups
linear_group	creates a linear group from command line arguments using linear_group_description

Continued on next page

Table 7 – continued from previous page

Class	Purpose
linear_group_description	description of a linear group from the command line
matrix_group	a matrix group over a finite field in projective, linear or affine action
object_in_projective_space_with_action	to represent an object in projective space
orbit_transversal	a set of orbits using a vector of orbit representatives and stabilizers
perm_group	a domain for permutation groups whose elements are given in list notation
product_action	the product action of two group actions
projective_space_with_action	projective space $\mathrm{PG}(n, q)$ with automorphism group $\mathrm{PGGL}(n + 1, q)$
schreier	Schreier trees for orbits of groups on points.
schreier_sims	Schreier Sims algorithm to create the stabilizer chain of a permutation group.
schreier_vector	compact storage of schreier vectors
schreier_vector_handler	manages access to schreier vectors
set_and_stabilizer	a set and its known set stabilizer
sims	a stabilizer chain for a permutation group is used to represent a permutation group
strong_generators	a strong generating set for a permutation group with respect to a fixed action
subgroup	a subgroup of a group using a list of elements
symmetry_group	interface for the various types of group actions
union_find	a union find data structure (used in the poset classification)
union_find_on_k_subsets	a union find data structure (used in the poset classification)
vector_ge	vector of group elements
wreath_product	the wreath product group $\mathrm{GL}(d, q)$ wreath $\mathrm{Sym}(n)$

A.3 Namespace Classification

Table 8: Orbiter Namespace Classification.

Class	**Purpose**
classification_step	a single step classification of combinatorial objects
compute_stabilizer	to compute the stabilizer of a set under a given action
coset_table_entry	a helper class for the poset classification algorithm
extension	to represent a flag; related to poset_orbit_node
flag_orbit_node	to represent a flag orbit; related to the class flag_orbits
flag_orbits	stores the set of flag orbits; related to the class classification_step
orbit_based_testing	maintains a list of test functions which define a G-invariant poset
orbit_node	to encode one group orbit, associated to the class classification_step
poset	a poset on which a group acts
poset_classification	the poset classification algorithm (aka Snakes and Ladders)
poset_description	description of a poset from the command line
poset_orbit_node	to represent one poset orbit; related to the class poset_classification
set_stabilizer_compute	to compute the stabilizer of a set under a given action
upstep_work	a helper class for the poset classification algorithm

A.4 Namespace Toplevel

Table 9: Orbiter Namespace Toplevel.

Class	**Purpose**
arc_generator	poset classification for arcs in desarguesian projective planes

Continued on next page

Table 9 – continued from previous page

Class	Purpose
arc_lifting	creates a cubic surface from a 6-arc in a plane
arc_lifting_simeon	arc lifting according to Simeon Ball and Ray Hill
arc_orbits_on_pairs	orbits on pairs of points of a nonconical six-arc
arc_partition	orbits on the partitions of the remaining four point of a nonconical arc
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
choose_points_or_lines	to classify objects in projective planes
classify_double_sixes	to classify double sixes in $\mathrm{PG}(3, q)$
classify_trihedral_pairs	classification of double triplets in $\mathrm{PG}(3, q)$
exact_cover	exact cover problems arising with the lifting of combinatorial objects
exact_cover_arguments	command line arguments to control the lifting via exact cover
factor_group	auxiliary class for create_factor_group, which is used in analyze_group.cpp
invariants_packing	collection of invariants of a set of packings in $\mathrm{PG}(3, q)$
isomorph	hybrid algorithm to classify combinatorial bjects
isomorph_arguments	a helper class for isomorph
isomorph_worker_data	auxiliary class to pass case specific data to the function isomorph_worker
k_arc_generator	classification of k-arcs in the projective plane $\mathrm{PG}(2, q)$
kramer_mesner	poset classification and orbital matrices
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.

Continued on next page

Table 9 – continued from previous page

Class	Purpose
orbit_rep	to hold one orbit after reading files from Orbiters poset classification
packing	classification of packings in $\mathrm{PG}(3,q)$
packing_invariants	geometric invariants of a packing in $\mathrm{PG}(3,q)$
polar	the orthogonal geometry as a polar space
recoordinatize	three skew lines in $\mathrm{PG}(3,q)$, used to classify spreads
representatives	auxiliary class for class isomorph
search_blocking_set	classification of blocking sets in projective planes
singer_cycle	the Singer cycle in $\mathrm{PG}(n-1,q)$
six_arcs_not_on_a_conic	to classify six-arcs not on a conic in $\mathrm{PG}(2,q)$
spread	to classify spreads of $\mathrm{PG}(k-1,q)$ in $\mathrm{PG}(n-1,q)$ where $n=2k$
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
subspace_orbits	poset classification for orbits on subspaces
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_with_action	an instance of a cubic surface together with its stabilizer
surface_with_action	cubic surfaces in projective space with automorphism group
surfaces_arc_lifting	to classify cubic surfaces using lifted arcs
translation_plane_via_andre_model	a translation plane created via Andre / Bruck / Bose
young	The Young representations of the symmetric group.

A.5 Namespace Discreta

Table 10: Orbiter Namespace Discreta.

Class	Purpose
bt_key	DISCRETA class for databases.
btree	DISCRETA class for a database.
btree_page_registry_key_pair	DISCRETA internal class related to class database.
buffer	DISCRETA auxiliary class related to the class database.
database	DISCRETA class for a database.
datatype	DISCRETA auxiliary class related to the class database.
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.
discreta_base	DISCRETA base class. All DISCRETA classes are derived from this class.
domain	DISCRETA class for influencing arithmetic operations.
ff_memory	DISCRETA auxilliary class for class domain.
geometry	DISCRETA class for incidence matrices.
group_selection	DISCRETA class to parse a group description from the command line.
hollerith	DISCRETA string class.
integer	DISCRETA integer class.
itemtyp	DISCRETA auxiliary class related to the class database.
keycarrier	DISCRETA auxiliary class related to the class database.
longinteger	DISCRETA class for integers of arbitrary magnitude.
longinteger_representation	DISCRETA internal class to represent long integers.
matrix	DISCRETA matrix class.
matrix_access	DISCRETA utility class for matrix access.

Continued on next page

Table 10 – continued from previous page

Class	Purpose
memory	DISCRETA class to serialize data structures.
number_partition	DISCRETA class for partitions of an integer.
OBJECTSELF	DISCRETA internal class.
page_table	DISCRETA class for bulk storage.
pagetyp	DISCRETA auxiliary class related to the class database.
permutation	DISCRETA permutation class.
printing_mode	DISCRETA class related to printing of objects.
solid	DISCRETA class for polyhedra.
unipoly	DISCRETA class for poynomials in one variable.
Vector	DISCRETA vector class for vectors of DISCRETA objects.
with	DISCRETA class related to class domain.

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. Poset classification and optimal linear codes. Submitted.

[4] Anton Betten and Fatma Karaoglu. Cubic surfaces over small finite fields, to appear in Designs, Codes, Cryptography.

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

[6] L.E. Dickson. Projective classification of cubic surfaces modulo 2, Ann. of Math. 16 (1915), 139-157.

[7] James W. P. Hirschfeld. The double-six of lines over $PG(3, 4)$. *J. Austral. Math. Soc.*, 4:83–89, 1964.

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

[9] Nauty User's Guide (Version 2.6), Brendan McKay, Australian National University, 2016.

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

[11] Muhammed Afsal Soomro. Algebraic curves over finite fields, Thesis, University of Groningen, 2013.