Orbiter User’s Guide

Anton Betten

February 15, 2021

Abstract

Orbiter is a computer algebra system devoted to the classification problem of combinatorial and algebraic-geometric objects. It uses computational methods from the theory of permutation groups to provide efficient algorithms for computing orbits. This guide explains how to use Orbiter through this command line interface. Programmers who want to use the Orbiter C++ class library directly in their own programs should consult the programmer’s guide.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Group Theoretic Activities</td>
<td>100</td>
</tr>
<tr>
<td>26</td>
<td>Group Theoretic Activities Based on Magma</td>
<td>105</td>
</tr>
<tr>
<td>27</td>
<td>Orbit Algorithms</td>
<td>111</td>
</tr>
<tr>
<td>28</td>
<td>Poset Classification</td>
<td>119</td>
</tr>
<tr>
<td>29</td>
<td>Orbits on Subsets</td>
<td>123</td>
</tr>
<tr>
<td>30</td>
<td>Orbits on Subspaces</td>
<td>126</td>
</tr>
<tr>
<td>31</td>
<td>Creating Objects in Projective Geometries</td>
<td>128</td>
</tr>
<tr>
<td>32</td>
<td>Activities for Objects in Projective Geometries</td>
<td>132</td>
</tr>
<tr>
<td>33</td>
<td>Canonical Forms of Objects in Projective Geometries</td>
<td>134</td>
</tr>
<tr>
<td>34</td>
<td>Arcs and Caps in Projective Spaces</td>
<td>136</td>
</tr>
<tr>
<td>35</td>
<td>Cubic Curves</td>
<td>142</td>
</tr>
<tr>
<td>36</td>
<td>Cubic Surfaces: Creation</td>
<td>143</td>
</tr>
<tr>
<td>37</td>
<td>Cubic Surfaces: Classification and Recognition</td>
<td>147</td>
</tr>
<tr>
<td>38</td>
<td>Number Theory</td>
<td>165</td>
</tr>
<tr>
<td>39</td>
<td>Cryptography</td>
<td>167</td>
</tr>
<tr>
<td>40</td>
<td>Coding Theory</td>
<td>173</td>
</tr>
<tr>
<td>41</td>
<td>Diophantine Systems</td>
<td>179</td>
</tr>
<tr>
<td>42</td>
<td>Combinatorial Linear Spaces</td>
<td>182</td>
</tr>
<tr>
<td>43</td>
<td>Design Theory</td>
<td>184</td>
</tr>
<tr>
<td>44</td>
<td>Tactical Decompositions</td>
<td>188</td>
</tr>
<tr>
<td>45</td>
<td>Spreads</td>
<td>194</td>
</tr>
<tr>
<td>46</td>
<td>Packings</td>
<td>202</td>
</tr>
<tr>
<td>47</td>
<td>BLT-Sets</td>
<td>209</td>
</tr>
<tr>
<td>48</td>
<td>Graph Theory</td>
<td>217</td>
</tr>
<tr>
<td>49</td>
<td>Graphical Output</td>
<td>221</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>50 The Povray Interface</td>
<td>228</td>
<td></td>
</tr>
<tr>
<td>51 Creating Animations</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>52 Continuous Function Plotter</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>53 Miscellaneous</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>54 Limitations</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>55 Contributors</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>References</td>
<td>252</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

Orbiter is a computer algebra system for the classification of combinatorial objects. Orbiter contributes to the knowledge base of combinatorial structures, and to provide useful tools to investigate structures from various points of view, including their symmetry properties. Orbiter is optimized for efficiency in terms of memory and execution speed. Orbiter is a library of C++ classes, together with a command line driven front end. There is no graphical user interface. The system offers two modes of use, programming or command line interface. This manual is about the command line interface. Readers who are interested in the Orbiter C++ class library should consult the programmer’s guide. A makefile with all commands used in this guide can be found in the examples subdirectory.
2 Installation

The installation of Orbiter requires the following steps:

(a) Ensure that git and the C++ development suite are installed (gnuc and make). Windows users may have to install cygwin (plus the extra packages git, make, gnuc). Macintosh users may have to install the xcode development tools from the appstore (it is free). Linux users may have to install the development packages. Orbiter often produces latex reports. In order to compile these files, make sure you have latex installed.

(b) Clone the Orbiter source tree from github (abetten/orbiter). The commands are:

\begin{verbatim}
git clone <github-orbiter-path>
\end{verbatim}

where <github-orbiter-path> has to be replaced by the actual address provided by github. To obtain this path, find Orbiter on github, then click on the green box that says “Code” and copy the address into the clipboard by clicking the clipboard symbol (see Figure 1). Back in the terminal, you can paste this text after the \texttt{git clone} command.

(c) Issue the following commands to compile Orbiter:

\begin{verbatim}
made
make install
\end{verbatim}

These two commands compile the Orbiter source tree and copy the executables to the subdirectory bin inside the Orbiter source tree. The orbiter executable is called orbiter.out.

Figure 1: GitHub Orbiter Page
3 Makefiles and Shell Scripts

Orbiter commands are often long and convoluted. It is therefore helpful to use unix-tools like makefiles and shell scripts to compile these commands or command sequences in a file. Shell scripts are simple text files that are executed by the unix-shell. Makefiles are text files that are processed by the make program. In both cases, the commands are listed one-by-one in a text file. The Orbiter commands can be supplemented with other system commands. The standard suite of unix commands on unix-like systems provides a lot of tools that can be used together with Orbiter. Makefiles go beyond shell scripts in that they allow several command sequences to be collected in one file. Each command sequence is labeled, and the make command is able to select the command sequence based on the label. For instance,

```
make PG_2_16
```

selects the command sequence `PG_2_16` in makefile. By default, a makefile must be called `makefile` (hence the name). It is possible to change the name of the makefile and use the `-f` option to tell `make` the name of the makefile to be used. So, if the makefile that contained the `PG_2_16` sequence was called `projective`, then

```
make -f projective PG_2_16
```

could be used to invoke the same command sequence.

Working with Orbiter relies a lot on the file system. Data files are read and created all the time. Most of the files are created in the current working directory. For this reason, it is best to do Orbiter work in a separate work directory. The work directory should be separate from the Orbiter source tree, which was created using the `git clone` command. A makefile inside the work directory can be used to collect the Orbiter commands. By keeping the work directory and the Orbiter distribution separate, it is easier to upgrade Orbiter to the most recent version. This is because `git` will not complain about any changes in the Orbiter source tree. The task of selecting the path to the orbiter executable can be solved using makefile techniques. For instance, makefile variables can be used to specify the path as in the following example. The line numbers are added for display purposes only, they do not belong to the file. The file is called `makefile`.

```make
MY_PATH=~/DEV.20/GITHUB/orbiter
ORBITER_PATH=$(MY_PATH)/src/apps/orbiter

PG_2_16:
  $(ORBITER_PATH)/orbiter.out -cheat_sheet_PG_2_16
  pdflatex PG_2_16.tex
  open PG_2_16.pdf
```

With this makefile, the command

```
make PG_2_16
```
issues the command on line 5, which produces a cheat sheet for the projective plane of order 16. Let us look at how this works in some detail. The lines 4-7 form a command block labeled PG_2_16. The actual on lines 5-7 are executed by typing

```
make PG_2_16
```

The command on line 5 is

```
orbiter.out -cheat_sheet_PG 2 16
```

This command instructs Orbiter to create a cheat sheet for PG(2,16), the desarguesian projective plane over the field \( \mathbb{F}_{16} \). The Orbiter command creates a file called `PG_2_16.tex`. The command on line 6 translates the tex file into a pdf file. The command on line 7 is Macintosh specific and displays the pdf file on screen.

In line 1, the variable `MY_PATH` is defined to point to the Orbiter installation directory. In this particular example, this path is \texttt{DEV.20/GITHUB/orbiter} in the user’s home directory (the tilda sign represents the home directory). Of course, for different installations, this path will change. To access the variable, the dollar-parenthesis notation is used. For instance, \texttt{$(MY\_PATH)$} refers to the string \(~/DEV.20/GITHUB/orbiter\). Since the orbiter executable resides in the subdirectory \texttt{src/apps/orbiter}, the command in line 2 creates a variable \texttt{ORBITER\_PATH} whose value is composed of the path and the subdirectory using concatenation of strings. Thus, the value of \texttt{$(ORBITER\_PATH)$} is set to the path of the Orbiter directory. This will simplify the command to invoke Orbiter.
4 The Orbiter Session

Any orbiter session is invoked through the orbiter command `orbiter.out`, which is the name of the executable. Unless the executable resides in a directory contained in the search path of the shell, a path must be given. Several options apply to the orbiter session. They are listed in Table 1. Once started, the Orbiter session will produce a short welcome message:

```plaintext
Welcome to Orbiter! Your build number is 1081.
A user's guide is available here:
https://www.math.colostate.edu/~betten/orbiter/users_guide.pdf
The sources are available here:
https://github.com/abetten/orbiter
An example makefile with many commands from the user's guide is here:
https://github.com/abetten/orbiter/tree/master/examples/users_guide/makefile
Orbiter session finished.
User time: 0:00
```

The build number is the version number of the Orbiter software, as defined by the number of submits to the Git repository. Higher numbers mean more recent versions. After this message, Orbiter will start parsing the command line arguments. Once this is done, the session will execute these commands. At the end of the session, a short message is given that specifies the processor time used up by the session.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>v</td>
<td>Set verbosity to v. Larger values of v lead to more text output. v = 0 gives minimal output.</td>
</tr>
<tr>
<td>-list_arguments</td>
<td></td>
<td>Prints the command line arguments.</td>
</tr>
<tr>
<td>-seed</td>
<td>s</td>
<td>Seed the pseudo random number generator with the integer value s.</td>
</tr>
<tr>
<td>-memory_debug</td>
<td></td>
<td>Turn on dynamic memory debugging.</td>
</tr>
<tr>
<td>-override_polynomial</td>
<td>poly</td>
<td>Set the override polynomial for finite fields to poly.</td>
</tr>
<tr>
<td>-orbiter_path</td>
<td>p</td>
<td>Set the orbiter path to p. This is useful in case the Orbiter session has to clone or fork new Orbiter sessions. In most cases, the orbiter path will end with a forward slash “/.”</td>
</tr>
<tr>
<td>-magma_path</td>
<td>p</td>
<td>Set the magma path to p. This is useful in case the Orbiter session has to create a magma process.</td>
</tr>
<tr>
<td>-fork</td>
<td>L M f t s</td>
<td>Fork new Orbiter sessions in parallel. The new sessions will be indexed by the values i that result from a loop with start value f and increment s bounded from above by t, equivalent to a C-loop of type “for (i=f; i &lt; t; i+= s).” Every occurrence of the string L in the argument list is replaced by the resulting value of the loop variable i. The forked process will write to a file whose name is described through the mask M. The actual file name results from using the printf command from the C-library for M with the integer value of the loop variable. All of the command line arguments after the fork command are passed through to the new Orbiter session, with all arguments L replaced by the integer value of the loop counter. The number of Orbiter sessions forked is (t − f)/s. The orbiter path from -orbiter_path is used when starting the forked sessions.</td>
</tr>
</tbody>
</table>

Table 1: Orbiter session commands
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-loop</td>
<td>$i_0 \ i_1 \ s$</td>
<td>Creates the set $S = {i_0 + ks \mid i_0 + ks &lt; i_1, \ k = 0,1,\ldots}$.</td>
</tr>
<tr>
<td>-index_set</td>
<td>$S$</td>
<td>Initialize the set $S$ from a sequence of given numbers.</td>
</tr>
<tr>
<td>-affine_function</td>
<td>$a \ b$</td>
<td>Creates the set ${ax + b \mid x \in S}$. Here, $a,b \in \mathbb{Z}$.</td>
</tr>
<tr>
<td>-clone_with_</td>
<td>$a \ b$</td>
<td>Creates the set ${ax + b \mid x \in S} \cup S$. Here, $a,b \in \mathbb{Z}$.</td>
</tr>
<tr>
<td>affine_function</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-set_builder .. -end</td>
<td>descr</td>
<td>Recursively creates a set.</td>
</tr>
</tbody>
</table>

Table 2: Set Builder Commands

5 Set Builders

It is possible to create sets of integers using the \texttt{-set_builder .. -end} command, see Table 2. Let us look at some examples:

\texttt{-set_builder -loop 0 64 1 -end}
creates the set $\{0,1,\ldots,63\}$.

\texttt{-set_builder -index_set "2,3,5,7,11,13" -end}
creates the set $\{2,3,5,7,11,13\}$.

\texttt{-set_builder -loop 0 32 1 -affine_function 2 1 -end}
creates the set $\{1,3,5,\ldots,63\}$.

\texttt{-set_builder -loop 0 16 1 -affine_function 4 2 \}
\texttt{-clone_with_affine_function 4 3 -end}
creates the set

\[
\{2,3,6,7,18,19,22,23,10,11,14,15,26,27,30,31,34,35,38,39,42,43,46,47, \\
50,51,54,55,58,59,62,63.\}
\]

\texttt{-set_builder -loop 0 16 1 -affine_function 1 16 \}
\texttt{-clone_with_affine_function 1 48 -end}
creates the set

\[
\{16,18,24,26,48,50,56,58,17,19,25,27,49,51,57,59,20,22,28,30,52,54, \\
60,62,21,23,29,31,53,55,61,63\}.
\]
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-set_of_points</td>
<td>$S$</td>
<td>A set $S$ of points.</td>
</tr>
<tr>
<td>-set_of_lines</td>
<td>$S$</td>
<td>A set $S$ of lines.</td>
</tr>
<tr>
<td>-file_of_points</td>
<td>fname</td>
<td>Read a set of points from the given file.</td>
</tr>
<tr>
<td>-file_of_lines</td>
<td>fname</td>
<td>Read a set of lines from the given file.</td>
</tr>
<tr>
<td>-file_of_packings_through_spread_table</td>
<td>fname, fname2</td>
<td>Read a set of packings from the file fname. The packings are coded as sets of spreads refering to the table of spreads stored in fname2.</td>
</tr>
<tr>
<td>-file_of_point_set</td>
<td>fname</td>
<td>A file containing a set of points.</td>
</tr>
<tr>
<td>-file_of_designs</td>
<td>fname $v b k$</td>
<td>A file containing designs. $v$ is the number of points, $b$ is the number of blocks, $k$ is the block size and $s$ is the size of the classes in the block-partition.</td>
</tr>
</tbody>
</table>

Table 3: Orbiter commands to define an input stream of objects

6 Input Streams

An input stream is a collection of objects defined in files or on the command line that will be processed one-by-one. Table 3 list the commands to define input streams. For a list of commands to define input streams, see Table 3. Input streams will be used in Sections 32 and 33.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-power_mod</td>
<td>a n p</td>
<td>Computes $a^n \mod p$.</td>
</tr>
<tr>
<td>-discrete_log</td>
<td>b a p</td>
<td>Computes $n$ such that $a^n \equiv b \mod p$.</td>
</tr>
<tr>
<td>-extended_gcd</td>
<td>a b</td>
<td>Computes integers $g$, $u$, and $v$ such that $g = \gcd(a, b) = ua + vb$.</td>
</tr>
<tr>
<td>-square_root_mod</td>
<td>a p</td>
<td>Computes a square root of $a$ modulo $p$, i.e. an integer $b$ such that $b^2 \equiv a \mod p$.</td>
</tr>
<tr>
<td>-square_root</td>
<td>a</td>
<td>Computes $\lfloor \sqrt{a} \rfloor$ of an integer $a$.</td>
</tr>
<tr>
<td>-inverse_mod</td>
<td>a p</td>
<td>Computes the modular inverse of $a$ modulo $p$, i.e. an integer $b$ with $ab \equiv 1 \mod p$.</td>
</tr>
<tr>
<td>-draw_mod_n</td>
<td>n fname</td>
<td>Draws the integers modulo $n$ on a circle.</td>
</tr>
<tr>
<td>-f_draw_mod_n_inverse</td>
<td></td>
<td>Connect $a$ and $a^{-1} \mod n$ (if it exists).</td>
</tr>
<tr>
<td>-f_draw_mod_n_additive_inverse</td>
<td></td>
<td>Connect $a$ and $-a \mod n$.</td>
</tr>
<tr>
<td>-f_draw_mod_n_power_cycle</td>
<td>b</td>
<td>Connect $b, b^2, b^3, \ldots \mod n$.</td>
</tr>
</tbody>
</table>

Table 4: Basic Number Theory Commands

7 Basic Number Theory

Orbiter provides functions for computing with the ring of integers and integer factor rings. Computations with large integers are supported through a long integer data type which allows unrestricted precision. Table 4 shows Orbiter commands for basic number theory, including integer factor rings and the euclidean algorithm. For instance, the command

```
orbiter.out -v 5 -primitive_root 915839
```

computes a primitive root modulo 915839 using a randomized algorithm. The answer in this case is 43085. The command

```
orbiter.out -v 5 -power_mod 43085 49842 915839
```

computes

$43085^{49842} \mod 915839$

which is 487320. Conversely, the discrete log of 487320 with respect to the base 43085 modulo 915839 can be computed using the command

```
orbiter.out -v 5 -discrete_log 487320 43085 915839
```
orbiter.out -v 5 -discrete_log 487320 43085 915839

The answer to this command is 49842. This command is a brute force search, and can be quite expensive. The command

orbiter.out -v 5 -inverse_mod 1865025205 2147483647

computes the inverse of 1865025205 modulo 2147483647 which is 579785381. A different way of computing the inverse is using the 1-trick. To this end, the gcd can be used:

orbiter.out -v 5 -extended_gcd 1865025205 2147483647

This command produces the output

$$1 = -503526232 \times 2147483647 + 579785381 \times 1865025205$$

which is the gcd written as a lattice combination of the input arguments. The inverse of 1865025205 mod 2147483647 is the coefficient in front of the 1865025205. In order to compute the modular power

$$a^e \mod n,$$

the -power_mod command can be used. For instance,

orbiter.out -v 5 -power_mod 16807 1073741823 2147483647

computes 16807 raised to the power 1073741823 modulo 2147483647, which is 2147483646. In order to compute the modular square root, i.e. to solve for $x$ in

$$x^2 \equiv a \mod p$$

the -square_root_mod command can be used. For instance,

orbiter.out -v 2 -square_root_mod 33 41

finds that the square root of 33 mod 41 is 19, i.e.

$$19^2 \equiv 33 \mod 41.$$ 

This command applies the algorithm of Tonelli and Shanks (cf. [15]).

The command

orbiter.out -v 2 -draw_options -end \ 
 -draw_mod_n 13 mod_13_powers_of_2 -draw_mod_n_power_cycle 2 -end

computes the powers of 2 mod 13 and connects consecutive powers along the circle modulo 13. By changing the value of the base, the diagrams in Figure 2 are created. The cases $b = 2$ and $b = 6$ are special. In those cases, the sequence of powers of $b$ mod 13 loops back unto itself after visiting all non-zero elements modulo 13. This is because 2 and 6 are primitive elements modulo 13. Because $-1$ is a square modulo 13, the power cycles of $b$ and of $-b$ have the same length, so $-2 = 11$ and $-6 = 5$ are primitive elements also. In total, there are
Figure 2: Cycle of powers of $b$ modulo 13
4 primitive elements modulo 13. This agrees with \( \varphi(12) = 4 \), where \( \varphi(k) \) is Euler’s totient function, which counts the number of generators in the cyclic group of order \( k \). However, this reasoning relies on the fact that 13 is prime, which implies that the group of prime residues modulo 13 is cyclic.

The command

```
orbiter.out -v 2 -draw_options -scale 0.4 -embedded -end \
-draw_mod_n 127 mod_127 -draw_mod_n_power_cycle 3
```

creates the drawing shown in Figure 3.
8 Prime Fields

Let $\mathbb{F}_q$ denote the finite field with $q$ elements. Up to isomorphism, there is only one field of order $q$. Finite fields of prime order can be created as integer factor ring.

Important comment: *Orbiter implements finite fields using tables for addition and multiplication. This imposes a limitation on the size of the field that can be created.*

See Section 54 for a list of limitations of Orbiter.

If $p$ is a prime number, the integer factor ring $\mathbb{Z}/I(p)$ is a finite field. Here,

$$I(p) = p\mathbb{Z} = \{pk \mid k \in \mathbb{Z}\} = \{0, \pm k, \pm 2k, \pm 3k, \ldots\}$$

is the ideal of all integer multiples of $p$. The elements of $\mathbb{F}_p$ are the residue classes of the ideal given by the integer multiples of $p$. Each residue class has the form

$$\{a + kp \mid k \in \mathbb{Z}\}.$$

Standard representatives of the equivalence classes can be chosen as the smallest non-negative member in each class. This means that the standard representatives are the integers from 0 to $p - 1$. This canonical representative is the remainder after division by $p$. Two integers belong to the same residue class if they have the same remainder after division by $p$. For instance, 11 and 46 are in the same residue class modulo 5 because both have a remainder of 1 after division by five. It is convenient to identify the residue classes mod $p$ with the integers from 0 to $p - 1$. In Orbiter, this convention is used automatically. The addition table and the multiplication table can be used to add and multiply in $\mathbb{F}_p$. For instance, in Figure 4 the addition and multiplication tables of $\mathbb{F}_7$ are shown, both numerically and using colors. The natural ordering of the integers in the interval $[0, 6]$ is used. Different integers are represented by different colors. It is customary to restrict the multiplication table to the non-zero elements of the field.

A finite field $\mathbb{F}_q$ can be created using the

-`finite_field`

command. Table 5 lists Orbiter commands for creating a finite field that can come after `-finite_field`. For instance,

-`finite_field -q 3 -end`

creates the finite field $\mathbb{F}_3$. It is possible to create a symbolic variable for a finite field. For instance, the command

-`define F -finite_field -q 3 -end -end`
Figure 4: Addition and multiplication tables of $\mathbb{F}_7$

<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>$q$</td>
<td>Specify the order of the field. Here, $q = p^k$ for some prime $p$ and some positive integer $k$.</td>
</tr>
<tr>
<td>-override_polynomial</td>
<td>$n$</td>
<td>Specify the polynomial used to create the finite field. The polynomial is given as integer, using the base $p$ representation. See Section 10.</td>
</tr>
</tbody>
</table>

Table 5: Options for Creating Finite Fields
creates a symbolic variable $F = \mathbb{F}_3$.

Table 6 lists basic Orbiter activities for finite fields. More activities will follow in Section 10. Orbiter can create cheat sheets for finite fields. These are latex files which contain a report about properties of the field. For instance, the following Orbiter command creates a cheat sheet for $\mathbb{F}_7$. It also creates the tables for addition and multiplication and writes them as csv-file.

```
orbiter.out -v 3 -define F -finite_field -q 7 -end -end \
  -with F -do -finite_field_activity -cheat_sheet_GF -end
```

Here is the information about $\mathbb{F}_7$ from the cheat sheet. The element $\alpha$ is a primitive element.

$$Z_i = \log_\alpha(1 + \alpha^i)$$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\gamma_i$</th>
<th>$-\gamma_i$</th>
<th>$\gamma_i^{-1}$</th>
<th>$\log_\alpha(\gamma_i)$</th>
<th>$\alpha^i$</th>
<th>$Z_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DNE</td>
<td>DNE</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2 = \alpha^2</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3 = \alpha</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>DNE</td>
</tr>
<tr>
<td>4</td>
<td>4 = \alpha^4</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5 = \alpha^5</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6 = \alpha^3</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$$
\begin{array}{ccccccc}
+ & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 & 0 \\
2 & 2 & 3 & 4 & 5 & 6 & 0 & 1 \\
3 & 3 & 4 & 5 & 6 & 0 & 1 & 2 \\
4 & 4 & 5 & 6 & 0 & 1 & 2 & 3 \\
5 & 5 & 6 & 0 & 1 & 2 & 3 & 4 \\
6 & 6 & 0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
$$
Figure 5: Addition and multiplication table of $\mathbb{F}_7$ using a primitive element

\[
\begin{array}{ccccccc}
\cdot & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 4 & 6 & 1 & 3 & 5 \\
3 & 3 & 6 & 2 & 5 & 1 & 4 \\
4 & 4 & 1 & 5 & 2 & 6 & 3 \\
5 & 5 & 3 & 1 & 6 & 4 & 2 \\
6 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

\[
3^0 \equiv 1, \quad 3^1 \equiv 3, \quad 3^2 \equiv 2, \quad 3^3 \equiv 6
\]

\[
3^4 \equiv 4, \quad 3^5 \equiv 5, \quad 3^6 \equiv 1
\]

There is a second way of labeling the elements which is sometimes used. When the non-zero elements are arranged according to powers of a primitive element, the cyclic structure of the multiplicative group becomes apparent. If $\alpha$ is a primitive element, we arrange the elements of $\mathbb{F}_p$ as

\[0, 1, \alpha, \alpha^2, \ldots, \alpha^{q-2}.\]

The cheat sheet contains this list of field elements at the very end. In Figure 5, the addition and multiplication tables of $\mathbb{F}_7$ are shown with respect to the cyclic ordering of elements as

\[0, 3^0, 3^1, 3^2, \ldots, 3^6 = 0, 1, 3, 2, 6, 4, 5, 1.\]

In the second ordering, the addition table of the prime field no longer exhibits cyclic structure.
9 Polynomials Over Finite Fields

Finite Fields and Polynomials over them are connected. Finite fields of prime order $p$ can be constructed as factoring of the integers. This yields the fields denoted as $\mathbb{F}_p$. Polynomials with coefficients in $\mathbb{F}_p$ are constructed next. They are denoted as $\mathbb{F}_p[X]$. From these, finite extension fields can be constructed, using polynomial factor rings of the form

$$\mathbb{F}_p[X]/I(m(X))$$

where $I(m(X))$ denotes the principal ideal of all polynomials which are multiples of $m(X)$. Here, $m(X)$ needs to be an irreducible polynomial in $\mathbb{F}_p[X]$. This means that $m(X)$ cannot be written as a product of two polynomials $A(X)$ and $B(X)$ in $\mathbb{F}_p[X]$ with degree strictly less than the degree of $m(X)$. This process creates fields $\mathbb{F}_q$ where $q = p^e$ with $e = \deg m(X)$.

Once extension fields $\mathbb{F}_q$ have been created, it is possible to create extension fields of extension fields, using the same construction as polynomial factor rings, but with the ideal generated by a polynomial in $\mathbb{F}_q[X]$. Extension fields of extension fields do not give anything new, however. They could be constructed directly from the prime field $\mathbb{F}_p$. However, in some applications we want to consider extension fields of extension fields, so it is good to have them available.

The

```
-finite_field_activity ... -end
```

command sequence can be used to start a command requiring a finite field. The `-q q` option can be used to specify the order of the finite field. The `-override_polynomial a` option can be used to specify the polynomial $m(X)$ as integer $a$ in the base $p$ representation. This option can be omitted, in which case Orbiter will use a precomputed and built-in polynomial. Table 7 lists Orbiter activities for polynomials over finite fields. For instance, the command

```
orbiter.out -v 2 -define F -finite_field -q 2 -end -end \
   -with F -do -finite_field_activity -polynomial_division \ 
   "1,0,0,0,0,0,0,0,0,0,1" "1,0,1,1" -end
```

computes the polynomial long division of $A(X)$ by $B(X)$ over $\mathbb{F}_2$ where

$$A(X) = X^{10} + 1, \quad B(X) = X^3 + X^2 + 1.$$ 

The result is $Q(X)$ and $R(X)$ with

$$A(X) = Q(X) \cdot B(X) + R(X)$$

with

$$Q(X) = X^7 + X^6 + X^5 + X^3 + 1, \quad R(X) = X^2.$$ 

Note that the coefficient lists in the arguments are in the reverse order of how we print the polynomials.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-polynomial_division</td>
<td>$A(X) B(X)$</td>
<td>Polynomial division of $A(X)$ by $B(X)$ over $\mathbb{F}_q$. $A(X)$ and $B(X)$ are given as coefficient list, starting from the lowest coefficient.</td>
</tr>
<tr>
<td>-extended_gcd_for_polynomials</td>
<td>$A(X) B(X)$</td>
<td>Extended gcd for polynomials $A(X)$ and $B(X)$ over $\mathbb{F}_q$. $A(X)$ and $B(X)$ are given as coefficient list, starting from the lowest coefficient.</td>
</tr>
<tr>
<td>-polynomial_mult_mod</td>
<td>$A(X) B(X)$ $M(X)$</td>
<td>Multiply the polynomials $A(X)$ and $B(X)$ modulo $M(X)$ in $\mathbb{F}_q[X]$.</td>
</tr>
<tr>
<td>-Berlekamp_matrix</td>
<td>$A(X)$</td>
<td>Computes the rank of the Berlekamp matrix associated to the polynomial $A(X)$ over $\mathbb{F}_q$. The polynomial $A(X)$ is irreducible over $\mathbb{F}_q$ if the Berlekamp matrix has rank $d - 1$ where $d$ is the degree of $A(X)$. The Berlekamp matrix is $F - I$ where $F$ is the Frobenius matrix and $I$ is the identity matrix. The Frobenius matrix is the matrix of the Frobenius endomorphism with respect to the standard basis of the polynomial ring: $1, X, X^2, \ldots, X^{d-1}$.</td>
</tr>
<tr>
<td>-polynomial_find_roots</td>
<td>$A(X)$</td>
<td>Find the roots of $A(X)$ over $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-make_table_of_irreducible_polynomials</td>
<td>$d$</td>
<td>Produces a list of all irreducible polynomials of degree $d$ over $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-find_CRC_polynomials</td>
<td>$t n k$</td>
<td>Computes all CRC polynomials of degree $k$ over $\mathbb{F}_q$ who detect all error patterns of Hamming weight $t$ or less in messages of length $n$.</td>
</tr>
</tbody>
</table>

Table 7: Finite Field Activities Related to Polynomials
The command `-extended_gcd_for_polynomials` takes two polynomials $A(X)$ and $B(X)$ and computes polynomials $U(X)$ and $V(X)$ and $G(X)$ such that $G(X)$ is the greatest common divisor of $A(X)$ and $B(X)$ and
\[
G(X) = U(X) \cdot A(X) + V(X) \cdot B(X).
\]

For instance,
```
orbiter.out -v 2 -define F -finite_field -q 2 -end -end \
   -with F -do -finite_field_activity -extended_gcd_for_polynomials \
   "1,0,0,0,0,0,0,0,0,0,1" "1,0,1,1" -end
```
computes
\[
U(X) = X + 1, \quad V(X) = X^8 + X^5 + X^4 + X^3 + X, \quad G(X) = 1.
\]

The Berlekamp matrix can be used to test if a polynomial is irreducible over a given finite field. The polynomial is irreducible if and only if the rank of the Berlekamp matrix is $d - 1$, where $d$ is the degree of the polynomial. For instance, the command
```
orbiter.out -v 2 -define F -finite_field -q 2 -end -end \
   -with F -do -finite_field_activity -Berlekamp_matrix "1,1,0,1" -end
```
computes the Berlekamp matrix associated with the polynomial $X^3 + X + 1$ over $\mathbb{F}_2$. The matrix is
\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 0
\end{bmatrix}.
\]
Since the matrix has rank 2, the polynomial is irreducible.

Orbiter can compute irreducible polynomials. For a given degree over a given field $\mathbb{F}_q$ we distinguish two tasks: The first task is finding one irreducible polynomial of the given degree and with the given field of coefficients. The second task is finding all irreducible polynomials given that one has already been found.

For instance, the command
```
orbiter.out -v 3 -search_for_primitive_polynomial_in_range 2 2 2 10 | grep //
```
searches for primitive polynomials over $\mathbb{F}_2$ of degree 2 to 10. The output of the program is lengthy. For this reason, the unix command `grep` is used to filter for lines containing the given pattern “//”. This yields the list
```
"7", // X^2 + X + 1
"13", // X^3 + X^2 + 1
"25", // X^4 + X^3 + 1
"37", // X^5 + X^2 + 1
"97", // X^6 + X^5 + 1
"193", // X^7 + X^6 + 1
```
Primitive polynomials over the base field $\mathbb{F}_s$ are converted into integers, using the base-$s$ representation of integers. For instance, the polynomial $X^2 + X + 1$ is read as binary string 111, which in turn translates to the integer 7 (we use $s = 2$).

Regarding the problem of creating all irreducible polynomials, we can use the following command:

```
orbiter.out -v 6 -define F -finite_field -q 4 -end -end \
   -with F -do -finite_field_activity \
   -make_table_of_irreducible_polynomials 3 -end
```

It produces a table of all irreducible polynomials of degree 3 over $\mathbb{F}_4$. The output is:

```
The 20 irreducible polynomials of degree 3 over F_4 are:
3 2 1 1
1 3 0 1
3 1 2 1
3 2 3 1
2 2 3 1
2 2 2 1
1 2 0 1
1 0 1 1
3 3 3 1
2 3 2 1
3 1 1 1
3 3 2 1
1 0 3 1
3 0 0 1
2 1 1 1
2 0 0 1
2 1 3 1
1 1 0 1
2 3 1 1
1 0 2 1
```

Let us check this list a little bit. The command

```
orbiter.out -v 2 -define F -finite_field -q 4 -end -end \
   -with F -do -finite_field_activity -Berlekamp_matrix "1,3,0,1" -end
```
shows that the Berlekamp matrix has rank 2, so $1 + 3X + X^3$ (read over $\mathbb{F}_4$) is irreducible. Indeed, the coefficient vector $1, 3, 0, 1$ is the second entry in the list. However,

```
orbiter.out -v 2 -define F -finite_field -q 4 -end -end \
   -with F -do -finite_field_activity -Berlekamp_matrix "1,3,1,1" -end
```

has a Berlekamp matrix of rank one, so $1 + 3X + X^2 + X^3$ is reducible. Indeed, the coefficient vector $1, 3, 1, 1$ does not appear in the list.

The following command computes all binary CRC polynomials of degree $k = 10$ which detect every error pattern of Hamming weight at most $t = 3$ in messages of length $n = 128$.

```
orbiter.out orbiter.out -v 1 -define F -finite_field -q 2 -end -end \
   -with F -do -finite_field_activity -find_CRC_polynomials 3 128 10 -end
```

The program finds 244 polynomials in about 8 minutes time.
10 Extension Fields

Orbiter treats prime fields and extension fields in a unified manner. This section will discuss extension fields. These are fields of order \( q = p^e \) where \( p \) is a prime and \( e \) is a positive integer greater than one. The prime \( p \) is called the characteristic of the field. Such fields are constructed as a polynomial factorring. The polynomial ring is the ring \( \mathbb{F}_p[X] \). For a polynomial \( m(X) \) we consider the ideal

\[
\mathcal{I}(m) = m(X)\mathbb{F}_p[X] = \{m(X)k(X) \mid k(X) \in \mathbb{F}_p[X]\}
\]

of all polynomial multiples of \( m(X) \). Under the assumption that \( m(X) \) has degree \( e > 1 \) and is irreducible, the residue class ring

\[
\mathbb{F}_p[X]/\mathcal{I}(m)
\]

is a field with \( q = p^e \) elements. Each residue class has a canonical representative. The canonical representative is the unique element in the residue class which has degree less than \( e \) and leading coefficient one. By means of identification, we can take these polynomials to be the set of standard representatives of the residue classes. So, for instance, for \( q = 4 = 2^2 \), we can pick the irreducible polynomial \( m(X) = X^2 + X + 1 \) over \( \mathbb{F}_2 \) and have four standard representatives modulo \( \mathcal{I}(m) \), namely

\[
0, \\
1, \\
X, \\
X + 1.
\]

Together, these make up a complete set of representatives of the residue classes modulo \( \mathcal{I}(m) \), and hence can be identified with the elements of \( \mathbb{F}_4 \):

\[
\mathbb{F}_4 = \{0, 1, X, X + 1\}.
\]

The addition of polynomials is as in \( \mathbb{F}_2[X] \), so

\[
\begin{array}{cccc}
0 & 0 & X & X + 1 \\
1 & 1 & 0 & X + 1 \\
X & X & X + 1 & 0 \\
X + 1 & X + 1 & X & 1
\end{array}
\]

To compute the multiplication table for the field \( \mathbb{F}_4 \). We can use polynomial arithmetic modulo \( m(X) \): It is clear how multiplication by 0 or 1 works, so we need to focus on the polynomials \( X \) and \( X + 1 \):

\[
\begin{align*}
X \cdot X &= X^2 \equiv X + 1 \pmod{X^2 + X + 1}, \\
X \cdot (X + 1) &= X^2 + X \equiv 1 \pmod{X^2 + X + 1}, \\
(X + 1) \cdot X &= X^2 + X \equiv 1 \pmod{X^2 + X + 1}, \\
(X + 1) \cdot (X + 1) &= X^2 + 1 \equiv X \pmod{X^2 + X + 1},
\end{align*}
\]
The multiplication table of $F_4$ turns out to be

\[
\begin{array}{c|cccc}
 & 0 & 1 & X & X + 1 \\
\hline
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & X & X + 1 \\
X & 0 & X & X + 1 & 1 \\
X + 1 & 0 & X + 1 & 1 & X \\
\end{array}
\]

Figure 6 shows a graphical representation of the addition and multiplication tables of $F_4$ using colors to represent the different elements: White is zero, black is one, red is $X$ and green is $X + 1$. In the multiplication table, the row and column associated with the zero elements are removed.

Table 8 lists Orbiter activities for finite fields. This extends Table 6 in Section 10.

The isomorphism type of the resulting field only depends on the order $q$ of the field, and not on the choice of the polynomial. However, for practical computations, the choice of the polynomial matters. For instance, results can only be shared between different computer algebra systems if the same polynomials are used. Orbiter has a large collection of polynomials built...
in. Besides these, a polynomial can be specified. The polynomials that Orbiter offers are in fact primitive, which means that the root \( \alpha \) is a primitive element for the field \( \mathbb{F}_q \). This just means that it is a generator of the multiplicative group. So, any non-zero element in \( \mathbb{F}_q \) is a suitable power of \( \alpha \).

If \( \mathbb{F}_q \) is an extension of the prime field \( \mathbb{F}_p \), we use a different labeling. This time, we exploit the fact that \( \mathbb{F}_q \) is a vector space over \( \mathbb{F}_p \). Let \( \alpha \) be a root of the irreducible polynomial \( m(X) \in \mathbb{F}_p[X] \) used to create the field. Suppose that \( q = p^e \), i.e., the degree of \( m(X) \) is \( e \). An \( \mathbb{F}_p \)-basis for the vector space \( \mathbb{F}_q \) over \( \mathbb{F}_p \) is given by the powers \( \alpha^i \), for \( 0 \leq i < e \). Therefore, any element \( \gamma \) of \( \mathbb{F}_q \) has a unique expression of the form

\[
\gamma = \sum_{h=0}^{e-1} a_i \alpha^i, \quad 0 \leq a_i < p \text{ for all } i.
\]

The associated integer rank of \( \gamma \) is obtained by replacing \( \alpha \) by \( p \) in this expression and evaluating the expression over the integers. So, the rank of \( \gamma \) is

\[
\sum_{h=0}^{e-1} a_i p^i.
\]

As \( \gamma \) ranges over all field element in \( \mathbb{F}_q \), the rank values take on every value in the interval \([0, q-1]\). The ordering of elements of \( \mathbb{F}_q \) according to these ranks is called the lexicographical ordering. The numerical rank of zero is 0 and the numerical rank of one is 1. The numerical rank of \( \alpha \), the primitive element, is \( p \). The numerical ranks of the elements of the prime subfield are exactly the elements of \([0, p-1]\).

The primitive polynomials used by Orbiter to create small finite fields are listed in Table 9. The relation is given using the Greek letter that is used in orbiter cheat sheets for that particular field.

Let us look at a few examples. The command

```
orbiter.out -define F -finite_field -q 4 -end -end \
   -with F -do -finite_field_activity -cheat_sheet_GF -end
```

creates a report for the field \( \mathbb{F}_4 \). The command

```
orbiter.out -define F -finite_field -q 16 -end -end \
   -with F -do -finite_field_activity -cheat_sheet_GF -end
```

creates a cheat sheet for \( \mathbb{F}_{16} \). This command produces Table 10.

Unlike other computer algebra systems (GAP [20] and Magma [12]), Orbiter does not use Conway polynomials. Instead, it provides the option to override the polynomial used to create the finite field. For subfield relationships, the cheat sheet will indicate the irreducible polynomials of all subfields for a given field. For instance, Table 11 shows the subfields of \( \mathbb{F}_{64} \) generated by the polynomial \( X^6 + X^5 + 1 \) whose numerical rank is 97.
<table>
<thead>
<tr>
<th>$q$</th>
<th>Polynomial</th>
<th>Numerical</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$X^2 + X + 1$</td>
<td>7</td>
<td>$\omega^2 = \omega + 1$</td>
</tr>
<tr>
<td>8</td>
<td>$X^3 + X^2 + 1$</td>
<td>13</td>
<td>$\gamma^3 = \gamma^2 + 1$</td>
</tr>
<tr>
<td>9</td>
<td>$X^2 + X + 2$</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>$X^4 + X^3 + 1$</td>
<td>25</td>
<td>$\delta^4 = \delta^3 + 1$</td>
</tr>
<tr>
<td>25</td>
<td>$X^2 + X + 2$</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>$X^3 + 2X + 1$</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>$X^5 + X^2 + 1$</td>
<td>37</td>
<td>$\eta^5 = \eta^2 + 1$</td>
</tr>
<tr>
<td>49</td>
<td>$X^2 + X + 3$</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>$X^6 + X^5 + 1$</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>$X^4 + X^3 + 2$</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>$X^2 + 4X + 2$</td>
<td>167</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>$X^3 + X^2 + X + 2$</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>$X^7 + X^6 + 1$</td>
<td>193</td>
<td>$\zeta^7 = \zeta^6 + 1$</td>
</tr>
<tr>
<td>169</td>
<td>$X^2 + X + 2$</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>$X^5 + 2X + 1$</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>$X^8 + X^4 + X^3 + X^2 + 1$</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>289</td>
<td>$X^2 + X + 3$</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>343</td>
<td>$X^3 + 3X + 2$</td>
<td>366</td>
<td></td>
</tr>
<tr>
<td>361</td>
<td>$X^2 + X + 2$</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>$X^9 + X^4 + 1$</td>
<td>529</td>
<td></td>
</tr>
<tr>
<td>529</td>
<td>$X^2 + 2X + 5$</td>
<td>580</td>
<td></td>
</tr>
<tr>
<td>625</td>
<td>$X^4 + X^3 + X + 2$</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>729</td>
<td>$X^6 + X^5 + 2$</td>
<td>974</td>
<td></td>
</tr>
<tr>
<td>841</td>
<td>$X^2 + 5X + 2$</td>
<td>988</td>
<td></td>
</tr>
<tr>
<td>961</td>
<td>$X^2 + 2X + 3$</td>
<td>1026</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td>$X^{10} + X^3 + 1$</td>
<td>1033</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Orbiter primitive polynomials for fields $\mathbb{F}_q$ with $q \leq 1024$
polynomial: $X^4 + X^3 + 1 = 25$

$Z_i = \log_\alpha(1 + \alpha^i)$

Subfields:

### Table 10: The field $\mathbb{F}_{16}$

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Polynomial</th>
<th>Numerical Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_4$</td>
<td>$X^2 + X + 1$</td>
<td>7</td>
</tr>
</tbody>
</table>

### Table 11: The subfields of $\mathbb{F}_{64}$

<table>
<thead>
<tr>
<th>Subfield</th>
<th>Polynomial</th>
<th>Numerical rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{F}_4$</td>
<td>$X^2 + X + 1$</td>
<td>7</td>
</tr>
<tr>
<td>$\mathbb{F}_8$</td>
<td>$X^3 + X + 1$</td>
<td>11</td>
</tr>
</tbody>
</table>
Figure 7: Addition and multiplication table of $F_3$ and $F_9$ using the lexicographic ordering of elements

The lexicographic ordering has an interesting side-effect for the tabled of extension fields. The lexicographic ordering starts with the elements of the prime field. For this reason, the tables of the prime subfield can be seen in the upper left corner in the table of the extension field. For instance, for $F_9$ we see the tables for $F_3$ in the upper left corners, as illustrated in Figure 7. Recall that we do not draw the row and column associated with the zero element in the multiplication tables.

Orbiter uses primitive polynomials for creating extension fields. Because of this, the element $\alpha$ is always primitive. Since the numerical rank of $\alpha$ is $p$, this means that the rank $p$ always represents a primitive element in an extension field. For the addition and multiplication tables of $F_9$ arranged with respect to powers of a primitive element, see Figure 8.

A normal basis for a field extension $F_{q^d}$ over $F_q$ is a basis of $F_{q^d}$ as vector space over $F_q$ which consists of one cycle of the Frobenius automorphism of $F_{q^d}$ over $F_q$. For instance, the command

```
orbiter.out -v 2 -define F -finite_field -q 2 -end -end \n-define F \n-with F -do -finite_field_activity -normal_basis 3 -end
```

computes a normal basis of $F_8$ over $F_2$. Using the polynomial $X^3 + X^2 + 1$, the normal basis in terms of the standard polynomial basis $1, X, X^2, \ldots$ is given by the columns of the matrix

\[
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
\end{bmatrix}
\]
Reading the columns as coefficient vectors with respect to the standard basis, the normal basis is

\[ b_1 = 1 + X + X^2, \quad b_2 = X, \quad b_3 = X^2. \]

Let us apply the Frobenius mapping \( \Phi \) to the elements of the normal bases:

\begin{align*}
\Phi b_1 &= (1 + X + X^2)^2 = 1 + X^2 + X^4 = 1 + X^2 + X^3 + X = 1 + X + X^2 + X^2 + 1 = X = b_2, \\
\Phi b_2 &= X^2 = b_3, \\
\Phi b_3 &= X^4 = X^3 + X = X^2 + X + 1 = b_1.
\end{align*}

Thus,

\[ b_1 \mapsto b_2 \mapsto b_3 \mapsto b_1 \]

under \( \Phi \), as required.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-RREF</td>
<td>$m$ $n$ $L$</td>
<td>Compute the RREF of the $m \times n$ matrix $L$ over $\mathbb{F}_q$</td>
</tr>
<tr>
<td>-nullspace</td>
<td>$m$ $n$ $L$</td>
<td>Compute a basis for the right nullspace of the $m \times n$ matrix $L$</td>
</tr>
<tr>
<td>-normalize_from_the_right</td>
<td></td>
<td>Normalizes the result of -RREF or nullspace from the right</td>
</tr>
<tr>
<td>-normalize_from_the_left</td>
<td></td>
<td>Normalizes the result of -RREF or nullspace from the left</td>
</tr>
<tr>
<td>-eigenstuff</td>
<td>$d$ $M$</td>
<td>Computes the eigenvalues and eigenvectors of the given $d \times d$ matrix $M$ over $\mathbb{F}_q$</td>
</tr>
<tr>
<td>-eigenstuff_from_file</td>
<td>$d$ $\text{fname}$</td>
<td>Computes the eigenvalues and eigenvectors of the $d \times d$ matrix $M$ over $\mathbb{F}_q$. The matrix $M$ is read from a csv file.</td>
</tr>
<tr>
<td>-all_rational_normal_forms</td>
<td>$d$</td>
<td>Produces a report of all rational normal forms of endomorphisms of $\mathbb{F}_q^d$</td>
</tr>
</tbody>
</table>

Table 12: Finite Field Activities for Linear Algebra

11 Linear Algebra Over Finite Fields

In Table 12, some finite field activities regarding linear algebra are shown. For instance, the command

```
orbiter.out -v 2 -define F -finite_field -q 2 -end -end \n  -with F -do -finite_field_activity \n  -RREF 2 5 "1,1,1,1,0,1,1,0,0,1" -normalize_from_the_right \n  -end
```

computes the RREF form of the matrix

$$
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
$$

over $\mathbb{F}_2$. The output is the matrix

$$
\begin{bmatrix}
1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 \\
\end{bmatrix}.
$$

The command
orbiter.out -v 2 -define F2 -finite_field -q 2 -end -end \ 
   -with F2 -do -finite_field_activity \ 
   -nullspace 2 5 "1,1,1,1,0,1,1,0,0,1" \ 
   -normalize_from_the_right \ 
   -end

computes the nullspace of the same matrix. The output is the matrix

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0
\end{bmatrix}
\]

Orbiter can compute eigenvalues and eigenvectors of matrices over finite fields. For instance, the command

\[
\text{orbiter.out} -v 6 -finite_field_activity -q 5 \ 
   -eigenstuff 4 "0,1,0,2,0,1,2,1,4,2,3,1,2,0,4,3" \ 
   -end
\]

computes all eigenvectors and eigenvalues of the matrix

\[
\begin{bmatrix}
0 & 1 & 0 & 2 \\
0 & 1 & 2 & 1 \\
4 & 2 & 3 & 1 \\
2 & 0 & 4 & 3
\end{bmatrix}
\]

over \( \mathbb{F}_5 \).

Orbiter can produce a list of all conjugacy classes of endomorphisms of \( \mathbb{F}_q^d \) by means of their rational normal forms. For instance

\[
\text{orbiter.out} -v 6 -finite_field_activity -q 2 \ 
   -all_rational_normal_forms 3 -end
\]

produces a list of all 6 conjugacy classes of \( \text{GL}(3, 2) \). The report includes the order of the centralizer and the order of the conjugacy class. The order of the centralizer is computed using Kung’s formula [31]. This command relies on the Orbiter catalogue of irreducible polynomials. For an introduction to the rational normal form of endomorphisms, see [37].
Conjugacy Classes of $\text{GL}(3, 2)$

The number of conjugacy classes of $\text{GL}(3, 2)$ is 6:

\[
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 1 \\
0 & 1 & 0 \
\end{bmatrix},
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \
\end{bmatrix},
\begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1 \
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
0 & 1 & 1 \
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1 \
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \
\end{bmatrix}
\]

Class 0 / 6
3, 1, 0

\[
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 1 \\
0 & 1 & 0 \
\end{bmatrix}
\]
centralizer order 7
class size 24
Class 1 / 6
2, 1, 0

\[
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \
\end{bmatrix}
\]
centralizer order 7
class size 24
Class 2 / 6
0, 1, 0; 1, 1, 0

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1 \
\end{bmatrix}
\]
centralizer order 3
class size 56
Class 3 / 6
<table>
<thead>
<tr>
<th>Class</th>
<th>Centralizer Order</th>
<th>Class Size</th>
<th>Matrix</th>
</tr>
</thead>
</table>
| 4 / 6 | 4                 | 42         | \[
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
0 & 1 & 1
\end{bmatrix}
\] |
| 5 / 6 | 8                 | 21         | \[
\begin{bmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\] |
|          | 168               | 1          | \[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\] |
<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-character_table_symmetric_group</td>
<td>n</td>
<td>Computes the character table of $\text{Sym}_n$.</td>
</tr>
<tr>
<td>-orbits_on_polynomials</td>
<td>d</td>
<td>Computes the representation of the group $G$ on homogeneous polynomials of degree $d$. This is a group theoretic activity as described in Section 25. The group $G$ must be constructed first.</td>
</tr>
</tbody>
</table>

Table 13: Representation Theory Commands

12 Representation Theory

Orbiter has some commands for representations of finite groups. Table 13 list the commands available to classify arcs. For instance, the command

```
orbiter.out -v 2 -character_table_symmetric_group 4
```

computes the character table of the symmetric group $\text{Sym}(4)$ using the algorithm of Burnside.

The character table of $\text{Sym}(4)$ is the matrix

$$
\begin{bmatrix}
1 & -1 & 1 & 1 & -1 \\
3 & 1 & 0 & -1 & -1 \\
2 & 0 & -1 & 2 & 0 \\
3 & -1 & 0 & -1 & 1 \\
1 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

The command

```
orbiter.out -v 4 \n  -define G \n  -linear_group -PGL 4 3 \n  -end -end \n  -with G -do \n  -group_theoretic_activities \n  -representation_on_polynomials 3 \n  -end
```

creates $G = \text{PGL}(4,3)$ and computes the representation on polynomials of degree 3 in 4 variables. The representation has degree 20. Figure 9 shows the representing matrices for a generating set of size 9.

37
Figure 9: Representation of PGL(4, 3) on cubic polynomials
Table 14: Finite Field Activities

<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cheat_sheet_PG</td>
<td>n</td>
<td>Produce a cheat sheet for PG(n,q)</td>
</tr>
<tr>
<td>-decomposition_by_element</td>
<td>e D fname</td>
<td>Let s be the group element represented by S. A report about the action of s^i is produced and written to the file fname. This option requires -cheat_sheet_PG.</td>
</tr>
<tr>
<td>-transversal</td>
<td>L1 L2</td>
<td></td>
</tr>
<tr>
<td>-intersection_of_two_lines</td>
<td>L1 L2</td>
<td>Computes the intersection of two lines.</td>
</tr>
<tr>
<td>-move_two_lines_in_hyperplane_stabilizer</td>
<td>M1 M2 N1 N2</td>
<td>Computes a projectivity of PG(3,q) fixing the hyperplane v(X3) moving line m_i to n_i for i = 1,2. It assumes that both m_1, m_2 and n_1, n_2 are skew. The lines m_i and n_i are given through Orbiter line numbers.</td>
</tr>
<tr>
<td>-move_two_lines_in_hyperplane_stabilizer_text</td>
<td>M1 M2 N1 N2</td>
<td>Like -move_two_lines_in_hyperplane_stabilizer, but now each line is given by a 2 x 4 generator matrix for the subspace.</td>
</tr>
<tr>
<td>-study_surface</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>-inverse_isomorphism_klein_quadric</td>
<td>L36</td>
<td></td>
</tr>
<tr>
<td>-rank_point_in_PG</td>
<td>n L</td>
<td>Computes the orbiter point rank of the vector L in PG(n,q).</td>
</tr>
</tbody>
</table>

13 Finite Projective Spaces

The orbiter commands related to finite projective spaces are grouped into three classes: finite field activities, projective space activities and otherwise. In Table 14, some commands regarding projective spaces over finite fields are shown. In Table 15, the commands associated with a projective space over a finite field are shown. Table 16 lists Orbiter commands related to projective geometries which are not tied to a finite field activity.

Finite field activities rely on a finite field object. The finite field object must be created first. The finite field activity can be invoked for the finite field object. Likewise, a projective space activity relies on a projective space object, which must be created first.

The permutation representation of various groups acting on projective space is based on set of points in the projective geometry PG(n,q). The purpose of this enumerator is to establish
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-canonical_form_PG</td>
<td>Descr</td>
<td>Computes the canonical form of objects in PG($n,q$). See Table 38 for details.</td>
</tr>
</tbody>
</table>

Table 15: Projective Space Activities

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-classify_cubic_curves</td>
<td>$q$</td>
<td>Classifies cubic curves in PG(2,$q$). Requires -control_arcs. See Section 35.</td>
</tr>
<tr>
<td>-control_arcs</td>
<td>description</td>
<td>Poset classification control for arcs used during the classification of cubic curves. See Table 31.</td>
</tr>
<tr>
<td>-create_points_on_quartic</td>
<td>$\epsilon$</td>
<td>Creates a table of points on a specific quartic curve. Consecutive points are no more than $\epsilon$ apart.</td>
</tr>
<tr>
<td>-create_points_on_parabola</td>
<td>$\epsilon$ $a$ $b$ $c$</td>
<td>Creates a table of points on the parabola $y = ax^2 + bx + c$. Consecutive points are no more than $\epsilon$ apart.</td>
</tr>
<tr>
<td>-smooth_curve</td>
<td>$\epsilon$ $N$ $b$ $t_{\text{min}}$ $t_{\text{max}}$ $\text{function}$</td>
<td>Creates at least $N$ points on a continuous curve given by “function”. Consecutive points are no more than $\epsilon$ apart. The function must be in terms of a parameter $t$. The values of $t$ are taken from the interval $[t_{\text{min}}, t_{\text{max}}]$.</td>
</tr>
<tr>
<td>-create_BLT_set</td>
<td>description</td>
<td>Creates a BLT-set according to the description.</td>
</tr>
<tr>
<td>-create_spread</td>
<td>description</td>
<td>Creates a spread according to the description.</td>
</tr>
<tr>
<td>-make_table_of_surfaces</td>
<td></td>
<td>Produces a latex table summarizing the surfaces in the Orbiter catalogue.</td>
</tr>
</tbody>
</table>

Table 16: Orbiter commands related to projective geometries
a bijection between the set of points and the integers on the interval \([0, \theta_n(q) - 1]\), where

\[
\theta_n(q) = \frac{q^{n+1} - 1}{q - 1}.
\]

In order to facilitate the bijection, Orbiter enumerates representative vectors for the one-dimensional subspaces. The conditions on the vectors are summarized below:

1. The vector is not the zero vector.
2. The rightmost nonzero entry in the vector is one. If it is not, we normalize the vector so that the rightmost nonzero vector is indeed one. This operation does not change the projective point which is associated with the vector.

The second condition ensures that we list each projective point exactly once. We require two functions, \textsc{Rank} and \textsc{Unrank}. The function \textsc{Rank} takes a vector \(x \in \mathbb{F}_q^n\), not zero, and maps it to the element in \(\mathbb{Z}_N\) representing the projective point \(P(x)\). A frame in \(\text{PG}(n, q)\) is a set of \(n + 2\) points, no \(n + 1\) in a hyperplane. We assume that the coordinates of a vector are indexed by the elements of \(\mathbb{Z}_n\). Also, we let \(e_i\) be the \(i\)-th unit vector. A frame for \(\text{PG}(n, q)\) is

\[
e_0, \ldots, e_{n-1}, e_0 + \cdots + e_{n-1}.
\]

This is the \textit{standard frame}. We start the labeling of points with the standard frame. After these \(n + 2\) points, we list the remaining points in lexicographic ordering (utilizing right-normalized representative). Thus, for \(\text{PG}(2, 2)\) the ordering is

\[(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 1), (1, 1, 0), (1, 0, 1), (0, 1, 1).
\]

Let us describe the two functions rank and unrank to perform the actual mappings between \(\text{PG}(n, q)\) and \(\mathbb{Z}_N\), where \(N = \theta_n(q)\). For this, assume that ranking and unranking functions have already been defined for the elements of the finite field \(\mathbb{F}_q\). Thus, we assume that for \(x \in \mathbb{F}_q\), \(\text{RANK}(\mathbb{F}_q, x)\) is a number \(b\) in \(\mathbb{Z}_q\). Also, for \(b \in \mathbb{Z}_q\), we assume that \(\text{UNRANK}(\mathbb{F}_q, b)\) is the corresponding \(x \in \mathbb{F}_q\). So, we assume that \(\text{RANK}\) and \(\text{UNRANK}\) are mutually inverse functions. Consider the group \(\text{PGL}(3, 2)\) acting on \(\text{PG}(2, 2)\), for instance. The points of \(\text{PG}(2, 2)\) are listed in 17.

For instance, the command

\[
\text{orbiter.out} \ \
-\text{draw_options} \ -\text{end} \ 
-\text{define} \ F \ -\text{finite_field} \ -q \ 2 \ -\text{end} \ -\text{end} \ 
-\text{with} \ F \ -\text{do} \ -\text{finite_field_activity} \ -\text{cheat_sheet_PG} \ 3 \ -\text{end}
\]

creates a report for the projective geometry \(\text{PG}(3, 2)\). The report includes a list of the enumerated objects:
Algorithm 1 Rank

1: procedure Rank(vector : $x$, field : $\mathbb{F}_q$, int : $n$)
2:  assert $x$ is a nonzero vector in $\mathbb{F}_q^n$.
3:  if $x = e_i$ then
4:      return $i$
5:  if $x = \text{one}$ then
6:      return $n$
7:  $i \leftarrow \max\{j \in \mathbb{Z}_n \mid x_j \neq 0\}$
8:  $x \leftarrow \frac{1}{x_i} x$
9:  $a := 0$
10:  for $j = i - 1, \ldots, 1, 0$ do
11:     $a \leftarrow a + \text{Rank}(\mathbb{F}_q, x_j)$
12:     if $j > 0$ then
13:        $a \leftarrow a \cdot q$
14:  if $i = n - 1$ and $a \geq \sum_{j=0}^{i-1} q^j$ then
15:     $a \leftarrow a - 1$
16:     $a \leftarrow a + n - i + \sum_{j=0}^{i-1} q^j$
17:  return $a$

<table>
<thead>
<tr>
<th>$a = \text{Rank}(x)$</th>
<th>$x = \text{Unrank}(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>(0, 1, 0)</td>
</tr>
<tr>
<td>2</td>
<td>(0, 0, 1)</td>
</tr>
<tr>
<td>3</td>
<td>(1, 1, 1)</td>
</tr>
<tr>
<td>4</td>
<td>(1, 1, 0)</td>
</tr>
<tr>
<td>5</td>
<td>(1, 0, 1)</td>
</tr>
<tr>
<td>6</td>
<td>(0, 1, 1)</td>
</tr>
</tbody>
</table>

Table 17: Representatives of the points of PG(2, 2)
Algorithm 2 Unrank

1: procedure Unrank(int : $a$, field : $\mathbb{F}_q$, int : $n$)
2:   assert $a \in \mathbb{Z}_N$ where $N = \theta_{n-1}(q)$.
3:   if $a < n$ then
4:     return $e_a$
5:   $a \leftarrow a - n$
6:   if $a = 0$ then
7:     return one
8:   $a \leftarrow a - 1$
9:   $x \leftarrow 0$
10:  for $i = 1, \ldots, n - 1$ do
11:     if $a \geq \sum_{j=1}^{i-1} q^j$ then
12:       $a \leftarrow a - \sum_{j=1}^{i-1} q^j$
13:     else
14:       $x_i \leftarrow 1$
15:       break
16:  for $k = i + 1, \ldots, n - 1$ do
17:    $x_k \leftarrow 0$
18:  $a \leftarrow a + 1$
19:  if $i = n - 1$ and $a \geq \sum_{j=0}^{i-1} q^j$ then
20:    $a \leftarrow a + 1$
21:  $j \leftarrow 0$
22:  while $a > 0$ do
23:    $r \leftarrow a \mod q$
24:    $x_j \leftarrow \text{Unrank}(\mathbb{F}_q, r)$
25:    $j \leftarrow j + 1$
26:    $a \leftarrow (a - r)/q$
27:  for $h = j, \ldots, i - 1$ do
28:    $x_h \leftarrow 0$
29:  return $x$
The projective space $\text{PG}(3, 2)$

$q = 2$
$p = 2$
$e = 1$
$n = 3$
Number of points = 15
Number of lines = 35
Number of lines on a point = 7
Number of points on a line = 3

The points of $\text{PG}(3, 2)$

$\text{PG}(3, 2)$ has 15 points:

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

Normalized from the left:

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

The lines of $\text{PG}(3, 2)$

$\text{PG}(3, 2)$ has 35 1-subspaces:

$L_0 = \begin{bmatrix} 1000 \\ 0100 \end{bmatrix} = \text{Pl}(1, 0, 0, 0, 0, 0)$
$L_1 = \begin{bmatrix} 1000 \\ 0110 \end{bmatrix} = \text{Pl}(1, 0, 1, 0, 0, 0)$
$L_2 = \begin{bmatrix} 1000 \\ 0101 \end{bmatrix} = \text{Pl}(1, 0, 0, 0, 1, 0)$
$L_3 = \begin{bmatrix} 1000 \\ 0111 \end{bmatrix} = \text{Pl}(1,0,1,0,1,0)$

$L_4 = \begin{bmatrix} 1000 \\ 0010 \end{bmatrix} = \text{Pl}(0,0,1,0,0,0)$

$L_5 = \begin{bmatrix} 1000 \\ 0011 \end{bmatrix} = \text{Pl}(0,0,1,0,1,0)$

$: \ 

L_{34} = \begin{bmatrix} 0010 \\ 0001 \end{bmatrix} = \text{Pl}(0,1,0,0,0,0)$

Lines sorted by Pluecker coordinates

$0 = \text{Pl}(1,0,0,0,0,0) = L_0 = \begin{bmatrix} 1000 \\ 0100 \end{bmatrix}$

$1 = \text{Pl}(0,1,0,0,0,0) = L_{34} = \begin{bmatrix} 0010 \\ 0001 \end{bmatrix}$

$2 = \text{Pl}(0,0,1,0,0,0) = L_4 = \begin{bmatrix} 1000 \\ 0010 \end{bmatrix}$

$3 = \text{Pl}(0,0,0,1,0,0) = L_{30} = \begin{bmatrix} 0100 \\ 0001 \end{bmatrix}$

$4 = \text{Pl}(0,0,0,0,1,0) = L_6 = \begin{bmatrix} 1000 \\ 0001 \end{bmatrix}$

$5 = \text{Pl}(0,0,0,0,0,1) = L_{28} = \begin{bmatrix} 0100 \\ 0010 \end{bmatrix}$

$: \ 

34 = \text{Pl}(0,1,1,1,1,1) = L_{26} = \begin{bmatrix} 1101 \\ 0011 \end{bmatrix}$

PG$(3,2)$ has the following low weight Pluecker lines:

$L_0 = \begin{bmatrix} 1000 \\ 0100 \end{bmatrix} = \text{Pl}(1,0,0,0,0,0)$

$L_4 = \begin{bmatrix} 1000 \\ 0010 \end{bmatrix} = \text{Pl}(0,0,1,0,0,0)$

$L_6 = \begin{bmatrix} 1000 \\ 0001 \end{bmatrix} = \text{Pl}(0,0,0,0,1,0)$

$L_{28} = \begin{bmatrix} 0100 \\ 0010 \end{bmatrix} = \text{Pl}(0,0,0,0,0,1)$
The planes of PG(3, 2)

PG(3, 2) has 15 2-subspaces:

\[ L_0 = \begin{bmatrix} 1000 \\ 0100 \\ 0010 \end{bmatrix} = \mathbf{P}(0, 0, 0, 1, 0, 0) \]

\[ L_1 = \begin{bmatrix} 1000 \\ 0100 \\ 0011 \end{bmatrix} = \mathbf{P}(0, 1, 0, 0, 0, 0) \]

\[ \vdots \]

\[ L_{14} = \begin{bmatrix} 0100 \\ 0010 \\ 0001 \end{bmatrix} \]

The polynomial rings associated with PG(3, 2)

<table>
<thead>
<tr>
<th>h monomial</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( X_0 ) (1, 0, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>( X_1 ) (0, 1, 0, 0)</td>
</tr>
<tr>
<td>2</td>
<td>( X_2 ) (0, 0, 1, 0)</td>
</tr>
<tr>
<td>3</td>
<td>( X_3 ) (0, 0, 0, 1)</td>
</tr>
</tbody>
</table>

Suppose we are looking for a projectivity of PG(3, 16) fixing the plane \( v(X_3) \) pointwise and mapping a pair of skew lines not in that plane to another pair of skew lines not in that plane. For instance, we want to map

\[ M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mapsto N_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[ M_2 = \begin{bmatrix} 1 & 1 & 0 & \delta \\ 0 & 0 & 1 & 0 \end{bmatrix} \mapsto N_2 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \]

The command
orbiter.out -v 5 -define F -finite_field -q 16 -end -end \
-define F -finite_field_activity \
-move_two_lines_in_hyperplane_stabilizer_text \
"1,0,0,0, 0,0,0,1" "1,1,0,2, 0,0,1,0" \
"1,0,0,0, 0,0,0,1" "0,1,0,1, 0,0,1,0" \
-end

computes a projectivity which does so:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\delta^{14} & 0 & 0 & \delta^{14}
\end{bmatrix}
\]

Here, \(\delta\) is the primitive element in the built-in field \(\mathbb{F}_{16}\), satisfying \(\delta^4 = \delta^3 + 1\).
**Projective Planes**

An important case of projective geometries are the projective planes. The projective spaces \( \text{PG}(2, q) \) deserve special attention. They are called the desarguesian projective planes. There are projective planes which are not desarguesian.

The points in a projective plane have the coordinates \( P(x, y, z) \). We can distinguish one line, for instance \( z = 0 \), and call it the line at infinity. The points not on that line form an affine plane \( \text{AG}(2, q) \).

Orbiter cheat sheets of projective geometries provide extra information for projective planes. The command

\[
\text{orbiter.out -draw_options -rad 200 -end \}
-\text{define F -finite_field -q 4 -end -end \}
-\text{with F -do -finite_field_activity -cheat_sheet_PG 2 -end}
\]

creates a cheat sheet for \( \text{PG}(2, 4) \). It contains a drawing of the plane as shown in Figure 10. The affine plane is shown in the usual cartesian decomposition, while the line at infinity is wrapped around the top right corner. The points are listed next, then the canonical Baer subgeometry \( \text{PG}(2, 2) \), and then the points again, but with left-normalized vectors. Finally, the lines are shown.

Figure 10: The plane \( \text{PG}(2, 4) \)
PG(2, 4) has 21 points:

\[
\begin{align*}
P_0 &= (1, 0, 0) = (1, 0, 0) & P_{11} &= (2, 1, 1) = (\alpha, 1, 1) \\
P_1 &= (0, 1, 0) = (0, 1, 0) & P_{12} &= (3, 1, 1) = (\alpha^2, 1, 1) \\
P_2 &= (0, 0, 1) = (0, 0, 1) & P_{13} &= (0, 2, 1) = (0, \alpha, 1) \\
P_3 &= (1, 1, 1) = (1, 1, 1) & P_{14} &= (1, 2, 1) = (1, \alpha, 1) \\
P_4 &= (1, 1, 0) = (1, 1, 0) & P_{15} &= (2, 2, 1) = (\alpha, \alpha, 1) \\
P_5 &= (2, 1, 0) = (\alpha, 1, 0) & P_{16} &= (3, 2, 1) = (\alpha^2, \alpha, 1) \\
P_6 &= (3, 1, 0) = (\alpha^2, 1, 0) & P_{17} &= (0, 3, 1) = (0, \alpha^2, 1) \\
P_7 &= (1, 0, 1) = (1, 0, 1) & P_{18} &= (1, 3, 1) = (1, \alpha^2, 1) \\
P_8 &= (2, 0, 1) = (\alpha, 0, 1) & P_{19} &= (2, 3, 1) = (\alpha, \alpha^2, 1) \\
P_9 &= (3, 0, 1) = (\alpha^2, 0, 1) & P_{20} &= (3, 3, 1) = (\alpha^2, \alpha^2, 1) \\
P_{10} &= (0, 1, 1) = (0, 1, 1)
\end{align*}
\]

Baer subgeometry:

\[
\begin{align*}
P_0 &= (1, 0, 0) & P_2 &= (0, 0, 1) & P_4 &= (1, 1, 0) & P_{10} &= (0, 1, 1) \\
P_1 &= (0, 1, 0) & P_3 &= (1, 1, 1) & P_7 &= (1, 0, 1)
\end{align*}
\]

There are 7 elements in the Baer subgeometry.

Normalized from the left:

\[
\begin{align*}
P_0 &= (1, 0, 0) & P_6 &= (1, 2, 0) & P_{12} &= (1, 2, 2) & P_{18} &= (1, 3, 1) \\
P_1 &= (0, 1, 0) & P_7 &= (1, 0, 1) & P_{13} &= (0, 1, 3) & P_{19} &= (1, 2, 3) \\
P_2 &= (0, 0, 1) & P_8 &= (1, 0, 3) & P_{14} &= (1, 2, 1) & P_{20} &= (1, 1, 2) \\
P_3 &= (1, 1, 1) & P_9 &= (1, 0, 2) & P_{15} &= (1, 1, 3) \\
P_4 &= (1, 1, 0) & P_{10} &= (0, 1, 1) & P_{16} &= (1, 3, 2) \\
P_5 &= (1, 3, 0) & P_{11} &= (1, 3, 3) & P_{17} &= (0, 1, 2)
\end{align*}
\]

The Lines of PG(2, 4). PG(2, 4) has 21 1-subspaces:
The command

```
orbiter.out -draw_options -xin 20000 -yin 20000 -rad 200 \
-line_width 0.3 -nodes_empty -end \n\-define F \-finite_field -q 16 -end -end \n-with F \-do \-finite_field_activity \-cheat_sheet_PG 2 -end
```

produces the drawing of PG(2, 16) shown in Figure 11. The \texttt{-nodes_empty} command is used to suppress the drawing of the nodes. The \texttt{-xin 20000} and \texttt{-yin 20000} options double the input coordinate system (recall from Table 63 that the default values are 10000), which has the effect that the text appears smaller relative to the grid.
Figure 11: The plane $\mathrm{PG}(2, 16)$
The Grassmannian

Let $V$ be a finite dimensional vector space and let $\mathfrak{Sr}_k(V)$ be the Grassmannian of $k$-dimensional subspaces of $V$. If $\dim(V) = n$, the notation $\mathfrak{Sr}_{n,k}$ is used for $\mathfrak{Sr}_k(V)$. If $V = \mathbb{F}^n_q$, the notation $\mathfrak{Sr}_{n,k,q}$ is used for $\mathfrak{Sr}_k(V)$. The order of the set $\mathfrak{Sr}_{n,k,q}$ can be computed as

$$\begin{bmatrix} n \\ k \end{bmatrix}_q = \prod_{i=0}^{k-1} \frac{q^{n-i} - 1}{q^{k-i} - 1},$$

using the $q$-binomial coefficient.

Orbiter has an enumerator for the Grassmannian. The purpose of this enumerator is to establish a bijection between the Grassmannian and the integers in the interval $[0, N - 1]$, where $N = \begin{bmatrix} n \\ k \end{bmatrix}_q$. In order to do so, Orbiter picks a basis for each subspace. By writing the elements of the basis in the rows of a matrix, a $k \times n$ matrix is obtained. In order to make the matrix unique, we assume it to be in RREF. In coding theory, such a matrix is called a generator matrix.

The Orbiter cheat sheets for PG($n,q$) (see Section 13) contain lists of all Grassmannians, provided they are not too big. It is also possible to create cheat sheets specifically for one Grassmannian. For instance, the command

```
orbiter.out -define F -finite_field -q 2 -end -end \
-\with F -do -finite_field_activity -cheat_sheet_Gr 3 2 -end
```

produces a list of 2-dimensional subspaces of $\mathbb{F}_2^3$, i.e. the lines of PG($2,2$):

\[
\begin{align*}
L_0 &= \begin{bmatrix} 100 \\ 010 \\ 001 \end{bmatrix} & L_3 &= \begin{bmatrix} 101 \\ 010 \\ 011 \end{bmatrix} & L_6 &= \begin{bmatrix} 010 \\ 001 \end{bmatrix} \\
L_1 &= \begin{bmatrix} 100 \\ 011 \\ 001 \end{bmatrix} & L_4 &= \begin{bmatrix} 101 \\ 010 \\ 011 \end{bmatrix} & L_5 &= \begin{bmatrix} 110 \\ 001 \end{bmatrix}
\end{align*}
\]
16 Algebraic Sets

A set of points \( V \) in \( \mathsf{PG}(n,q) \) is algebraic if there is a set of homogeneous polynomials \( p_1, \ldots, p_r \) whose roots over \( \mathbb{F}_q \) are the given set. In this case, we write \( V = \mathbf{v}(p_1, \ldots, p_r) \). The set \( V \) is often called the variety of \( p_1, \ldots, p_r \).

Conversely, given a set of points \( V \) in \( \mathsf{PG}(n,q) \), the ideal \( I(V) \) is the set of homogeneous polynomials in \( \mathbb{F}_q[X_0, \ldots, X_n] \) which vanish on all of \( V \). This set is an ideal in the polynomial ring. In fact, it is a principal ideal, meaning that it is generated by one element only. Orbiter has ways to compute the variety of a polynomial ideal and to compute a generator for the ideal of a set.

Interestingly, in \( \mathsf{PG}(n,q) \), every set is algebraic of degree at most \((n+1)(q-1)\) [21]. The associated polynomial is unique and known as the algebraic normal form of the set.

In order to work with algebraic sets, polynomial rings are required. Orbiter offers homogeneous polynomials in any (finite) number of variables. There are two orderings of the monomials which can be chosen. The partition ordering (use option \texttt{-monomial\_type\_PART}) is grouping terms according to the partition that results from the degrees of the variables first, and then uses the lexicographic ordering as a tie breaker. The lexicographic ordering (use option \texttt{-monomial\_type\_LEX}) orders the monomials lexicographically. Table 18 shows the monomials in the partition ordering for degrees 2, 3, and 4 in a plane. Suppose we are interested in \( \mathbb{F}_{11} \) rational points of the elliptic curve \( y^2 = x^3 + x + 3 \). We write \( x^3 + 3 - y^2 + x = 0 \). Homogenizing yields \( X^3 + 3Z^3 - Y^2Z + XZ = 0 \). Using \( X_0, X_1, X_2 \) instead of \( X, Y, Z \) yields:

\[
X_0^3 + 3X_2^3 + 10X_1^2X_2 + X_0X_2^2 = 0.
\]

Using the indexing of monomials from Table 18, we record the following pairs \((a, i)\) where \( a \) is the coefficient and \( i \) is the index of the monomial:

\[(1,0), (3,2), (10,6), (1,7).\]

This is concatenated to the sequence 1, 0, 3, 2, 10, 6, 1, 7. The Orbiter command

\[
\text{orbiter.out -v 2 -create_combinatorial_object -q 11 -n 2} \ \\
\text{-projective_variety "EC" 3 "1,0,3,2,10,6,1,7"} \ \\
\text{-monomial\_type\_PART -end}
\]

creates the algebraic set associated to the cubic curve \( y^2 = x^3 + x + 3 \) in \( \mathsf{PG}(2,11) \). It turns out that there are exactly 18 points over \( \mathbb{F}_{11} \) (cf. Figure 12). Suppose we want to create the Hirschfeld surface over \( \mathbb{F}_4 \) with equation:

\[
X_0^2X_3 + X_1^2X_2 + X_1X_2^2 + X_0X_3^2 = 0.
\]

Table 19 shows the Orbiter monomial orderings for degrees 2 and 3 in \( \mathsf{PG}(3,q) \). Based on the partition ordering, the monomials appearing in the equation

\[
X_0^2X_3 + X_1^2X_2 + X_1X_2^2 + X_0X_3^2 = 0
\]
Table 18: The partition ordering of monomials of degree 1, 2, 3 and 4 in a plane

<table>
<thead>
<tr>
<th>$h$</th>
<th>mon</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$X_0^4$</td>
<td>(4, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>$X_1^4$</td>
<td>(0, 4, 0)</td>
</tr>
<tr>
<td>2</td>
<td>$X_2^4$</td>
<td>(0, 0, 4)</td>
</tr>
<tr>
<td>3</td>
<td>$X_0^3 X_1$</td>
<td>(3, 1, 0)</td>
</tr>
<tr>
<td>4</td>
<td>$X_0^3 X_2$</td>
<td>(3, 0, 1)</td>
</tr>
<tr>
<td>5</td>
<td>$X_0 X_1^3$</td>
<td>(1, 3, 0)</td>
</tr>
<tr>
<td>6</td>
<td>$X_1^3 X_2$</td>
<td>(0, 3, 1)</td>
</tr>
<tr>
<td>7</td>
<td>$X_0 X_2^3$</td>
<td>(1, 0, 3)</td>
</tr>
<tr>
<td>8</td>
<td>$X_1 X_2^3$</td>
<td>(0, 1, 3)</td>
</tr>
<tr>
<td>9</td>
<td>$X_0^2 X_1^2$</td>
<td>(2, 2, 0)</td>
</tr>
<tr>
<td>10</td>
<td>$X_0^2 X_2^2$</td>
<td>(2, 0, 2)</td>
</tr>
<tr>
<td>11</td>
<td>$X_1^2 X_2^2$</td>
<td>(0, 2, 2)</td>
</tr>
<tr>
<td>12</td>
<td>$X_0^2 X_1 X_2$</td>
<td>(2, 1, 1)</td>
</tr>
<tr>
<td>13</td>
<td>$X_1 X_2 X_2$</td>
<td>(1, 2, 1)</td>
</tr>
<tr>
<td>14</td>
<td>$X_0 X_1 X_2^2$</td>
<td>(1, 1, 2)</td>
</tr>
</tbody>
</table>

Figure 12: Elliptic curve $y^2 \equiv x^3 + x + 3 \mod 11$
<table>
<thead>
<tr>
<th>$h$</th>
<th>mon</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$X_0^3$</td>
<td>(3, 0, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>$X_1^3$</td>
<td>(0, 3, 0, 0)</td>
</tr>
<tr>
<td>2</td>
<td>$X_2^3$</td>
<td>(0, 0, 3, 0)</td>
</tr>
<tr>
<td>3</td>
<td>$X_3^3$</td>
<td>(0, 0, 0, 3)</td>
</tr>
<tr>
<td>4</td>
<td>$X_0^2X_1$</td>
<td>(2, 1, 0, 0)</td>
</tr>
<tr>
<td>5</td>
<td>$X_0^2X_2$</td>
<td>(2, 0, 1, 0)</td>
</tr>
<tr>
<td>6</td>
<td>$X_0^2X_3$</td>
<td>(2, 0, 0, 1)</td>
</tr>
<tr>
<td>7</td>
<td>$X_0X_1^2$</td>
<td>(1, 2, 0, 0)</td>
</tr>
<tr>
<td>8</td>
<td>$X_0X_2^2$</td>
<td>(0, 2, 1, 0)</td>
</tr>
<tr>
<td>9</td>
<td>$X_0X_3^2$</td>
<td>(0, 2, 0, 1)</td>
</tr>
<tr>
<td>10</td>
<td>$X_1X_2^2$</td>
<td>(1, 0, 2, 0)</td>
</tr>
<tr>
<td>11</td>
<td>$X_1X_3^2$</td>
<td>(0, 1, 2, 0)</td>
</tr>
<tr>
<td>12</td>
<td>$X_2X_3^2$</td>
<td>(0, 0, 2, 1)</td>
</tr>
<tr>
<td>13</td>
<td>$X_0X_2X_3$</td>
<td>(1, 0, 0, 2)</td>
</tr>
<tr>
<td>14</td>
<td>$X_1X_2X_3$</td>
<td>(1, 0, 1, 2)</td>
</tr>
<tr>
<td>15</td>
<td>$X_2X_3X_0$</td>
<td>(0, 0, 1, 2)</td>
</tr>
<tr>
<td>16</td>
<td>$X_0X_1X_2$</td>
<td>(1, 1, 1, 0)</td>
</tr>
<tr>
<td>17</td>
<td>$X_0X_1X_3$</td>
<td>(1, 1, 0, 1)</td>
</tr>
<tr>
<td>18</td>
<td>$X_0X_2X_3$</td>
<td>(1, 0, 1, 1)</td>
</tr>
<tr>
<td>19</td>
<td>$X_1X_2X_3$</td>
<td>(0, 1, 1, 1)</td>
</tr>
</tbody>
</table>

Table 19: The Orbiter ordering of monomials of degree 1, 2 and 3 in PG(3, q)
have rank 6, 8, 11 and 13. The coefficients are all one. Orbiter needs the equation in a
list of pairs, the first number representing the finite field element, and the second number
specifying the monomial rank in the chosen ordering. So, for this example, the sequence is
1, 6, 1, 8, 1, 11, 1, 13. The following command can be used to create the variety:

```
orbiter.out -v 2 -create_combinatorial_object -n 3 -q 4 \ 
 -projective_variety "Endrass_F7" 8 \ 
 -long_string "6,0,4,4,2,7,5,9,6,20,6,23,1,25,3,30,1,32,3,34,4,56,6,59,1,61,6,66,\ " 
 "2,68,6,70,3,77,2,79,6,83,6,120,2,123,5,125,3,130,1,132,3,134,3,141," 
```

A file called `Hirschfeld_surface_q4.txt` is created. The file contains the Orbiter ranks of
the 45 points on the surface.

Suppose we want to create the Endrass octic surface over a finite field. The real equation is

\[
64(-w^2 + x^2)(-w^2 + y^2)((x + y)^2 - 2w^2)((x - y)^2 - 2w^2) \\
-(-4(1 + \sqrt{2})(x^2 + y^2)^2 + (8(2 + \sqrt{2})z^2 + 2(2 + 7\sqrt{2})w^2)(x^2 + y^2) \\
-16z^4 + 8(1 - 2\sqrt{2})z^2w^2 - (1 + 12\sqrt{2})w^4)^2 = 0.
\]

Some precomputations in Maple and Orbiter are necessary to obtain the coefficient list of
this surface over a finite field. After all, there are 165 degree 8 monomials in 4 variables.
The command sequence

```
orbiter.out -v 4 -diophant -label octic_monomials \ 
 -coefficient_matrix 1 4 "1,1,1,1" -RHS "8,8,1" \ 
 -x_min_global 0 -x_max_global 8
```

```
orbiter.out -v 3 -diophant_activity \ 
 -input_file octic_monomials.diophant -solve_mckay
```

```
sort -r octic_monomials.sol >octic_monomials_sorted.txt
```

can be used to create a file `octic_monomials_sorted.txt` which lists the monomials in the
lexicographical ordering. These commands will be explained in Section 41. After reducing
modulo a prime \( p \), the equation can be considered with Orbiter. This means we consider the
equation in \( \mathbb{F}_p \). The only requirement is that 2 is a square in the field \( \mathbb{F}_p \). For instance, over
\( \mathbb{F}_7 \) we have \( \sqrt{2} = 3 \) because \( 3^2 = 9 \equiv 2 \mod 7 \), and the equation turns out to be

\[
6X_0^8 + 4X_0^6X_1^2 + 2X_0^5X_2^2 + 5X_0^4X_3^2 + 6X_0^3X_1X_4 + 6X_0^2X_2X_3 + X_0X_1X_2X_3 + 3X_1^4X_4 \\
+X_0^4X_2X_3 + 3X_0^4X_4 + 4X_0^3X_1^2 + 6X_0^2X_2^2 + X_0X_1X_2X_3 + 6X_0^2X_1X_2^2 + 2X_0X_1X_2X_3 + 2X_0X_1X_2X_3 \\
+6X_0^2X_1^2X_2^2 + 3X_0X_1^2X_3^2 + 2X_0X_1X_3 + 3X_0^2X_1^2X_2^2 + 6X_0^2X_1^2X_3^2 + 6X_1 + 2X_1^2X_2^2 + 5X_0X_2^2 + 3X_1X_3 \\
+X_1^2X_2^2 + 3X_1^2X_3 + 2X_0X_1^2X_2 + 6X_0X_1X_2^2 + 6X_0X_1X_3^2 + 3X_0X_2^2 = 0
\]

The following command

```
orbiter.out -v 2 -create_combinatorial_object -n 3 -q 7 \ 
 -projective_variety "Endrass_F7" 8 \ 
 -long_string "6,0,4,4,2,7,5,9,6,20,6,23,1,25,3,30,1,32,3,34,4,56,6,59,1,61,6,66,\ " 
 "2,68,6,70,3,77,2,79,6,83,6,120,2,123,5,125,3,130,1,132,3,134,3,141," 
```

56
shows that the surface has 33 points over \( \mathbb{F}_7 \). It is also possible to create the intersection of Zariski open sets. For instance, the command

\[
\text{orbiter.out -v 2 -create_combinatorial_object -q 11 -n 2 -}
\text{-intersection_of_zariski_open_sets "frame_complement" 1 4 -}
\text{"1,0" "1,1" "1,2" "1,0,1,1,1,2" -monomial_type_PART -end}
\]

creates the points off the fundamental quadrangle

\[
X_0 = 0, \quad X_1 = 0, \quad X_2 = 0, \quad X_0 + X_1 + X_2 = 0
\]

in PG(2,11).
17 The Klein Quadric and the Plücker Map

The Grassmannian is an algebraic variety. This means it can be described as the common zero-locus of a set of polynomials with rational coefficients. Amongst all Grassmannians, \( \mathfrak{G}_{r_4}(V) \) stands out. This variety \( \mathfrak{G}_{r_4}(V) \) is described by one single equation, which happens to be quadratic. The variety is known as the Klein quadric (cf. [45]).

Orbiter cheat sheets for Grassmannians as discussed in Section 15 behave slightly differently for the Klein quadric in that they will include information regarding this additional structure. For instance, the command

```
orbiter.out -define F -finite_field -q 2 -end -end \\
-define F -do -finite_field_activity -cheat_sheet_Gr 4 2 -end
```

creates the elements of \( \mathfrak{G}_{r_4,2} \) and lists them together with their Plücker coordinates. The following output is shortened:

There are 35 lines:

\[
L_0 = \begin{bmatrix} 1000 \\ 0100 \end{bmatrix} = \text{Pl}(1,0,0,0,0,0) \\
L_1 = \begin{bmatrix} 1000 \\ 0110 \end{bmatrix} = \text{Pl}(1,0,1,0,0,0)
\]

and hence belong to the quadric \( Q^+(5,q) \). This quadric is also known as the Klein quadric. Orthogonal spaces and quadrics will be discussed in Section 18. Orbiter has a labeling of points of quadrics that can be used to enumerate the points of \( Q^+(5,q) \). Using the inverse Plücker map, this gives a second way to label the lines of \( \text{PG}(3,q) \). In the example of \( \text{PG}(3,2) \) this yields the following list (output shortened):

The Plücker coordinates satisfy

\[
p_{12}p_{34} - p_{13}p_{24} + p_{14}p_{23} = 0
\]

and hence belong to the quadric \( Q^+(5,q) \). This quadric is also known as the Klein quadric. Orthogonal spaces and quadrics will be discussed in Section 18. Orbiter has a labeling of points of quadrics that can be used to enumerate the points of \( Q^+(5,q) \). Using the inverse Plücker map, this gives a second way to label the lines of \( \text{PG}(3,q) \). In the example of \( \text{PG}(3,2) \) this yields the following list (output shortened):
0 = \textbf{Pl}(1, 0, 0, 0, 0, 0) = L_0 = \begin{bmatrix} 1000 \\ 0100 \end{bmatrix} \quad 2 = \textbf{Pl}(0, 0, 1, 0, 0, 0) = L_4 = \begin{bmatrix} 1000 \\ 0010 \end{bmatrix} \\
1 = \textbf{Pl}(0, 1, 0, 0, 0, 0) = L_{34} = \begin{bmatrix} 0010 \\ 0001 \end{bmatrix} \quad : \\
34 = \textbf{Pl}(0, 1, 1, 1, 1, 1) = L_{26} = \begin{bmatrix} 1101 \\ 0011 \end{bmatrix}
Type | Quadratic Form | # Points
--- | --- | ---
$Q^+(n, q)$ Hyperbolic ($n$ is odd) | $\sum_{i=0}^{\frac{n-1}{2}} X_{2i}X_{2i+1}$ | $(q^{(n+1)/2} - 1)(q^{(n-1)/2} + 1)$
$Q^-(n, q)$ Elliptic ($n$ is odd) | $p(X_{n-1}, X_n) + \sum_{i=0}^{\frac{n-1}{2}} X_{2i}X_{2i+1}$ | $(q^{(n+1)/2} + 1)(q^{(n-1)/2} - 1)$
$Q(n, q)$ Parabolic ($n$ is even) | $X_n^2 + \sum_{i=0}^{\frac{n}{2}-1} X_{2i}X_{2i+1}$ | $\frac{q^n - 1}{q - 1}$

Table 20: Nondegenerate Quadrics in PG($n, q$) and the canonical form adopted in Orbiter

18 Orthogonal Spaces

Orbiter can create and work with orthogonal spaces and their groups. An orthogonal space is created by a quadratic form. We assume that the form is nondegenerate. There are three types of nondegenerate quadratic forms in PG($n, q$). Two when $n$ is odd (hyperbolic and elliptic) and one if $n$ is even (parabolic). Basic information about these quadrics and their representative quadratic forms in Orbiter is given in Table 20. Here, $p(X, Y) = c_1X^2 + c_2XY + c_3Y^2 \in \mathbb{F}_q[X, Y]$ is irreducible over $\mathbb{F}_q$. To create an orthogonal space, the

```
-orthogonal_space \epsilon \ d \ q \ -end
```

command can be used. Here, $d = n + 1$, $q$ is the order of the finite field, and

$$
\epsilon = \begin{cases} 
1 & \text{hyperbolic type } Q^+(d-1, q), \ d \ even \\
0 & \text{elliptic type } \ Q(d-1, q), \ d \ odd \\
-1 & \text{hyperbolic type } \ Q^-(d-1, q), \ d \ even 
\end{cases}
$$

In order to create an object of type orthogonal space, the `-orthogonal_space` command is used inside a `-definition .. -end` command sequence. In Table 21, Orbiter command options for creating orthogonal spaces are shown. By default, the orthogonal space is created together with the orthogonal group PGO($n+1, q$). When $q$ is prime, the group PGO($n+1, q$) is created instead (the groups are isomorphic in this case, and PGO($n+1, q$) is a bit more efficient). For large orthogonal spaces, creating the group is expensive in terms of time and memory. The `a` command `-without_group` can be used to prevent the group from being created. For instance

```
(define O -orthogonal_space 1 6 2 -end
```

creates an object $O$ of type $Q^+(5, 2)$. The field can be tailored by using a finite field object. In order to do so, the parameter $q$ is replaced by a reference to the symbolic object associated with the finite field. For instance,
Table 21: Command options to create an orthogonal space

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-label_txt</code></td>
<td>L</td>
<td>Set the ascii-label of the space. The label is used for things like file names etc. A default label will be used if this option is not given.</td>
</tr>
<tr>
<td><code>-label_tex</code></td>
<td>L</td>
<td>Set the tex-label of the space. The label is used within latex reports. A default label will be used if this option is not given.</td>
</tr>
<tr>
<td><code>-without_group</code></td>
<td></td>
<td>Do not create the orthogonal group.</td>
</tr>
</tbody>
</table>

Table 22: Activities releated to orthogonal spaces

-define F -finite_field -q 8 -override_polynomial "11" -end -end
-define O -orthogonal_space 1 4 F -end

creates the orthogonal space associated with the quadric $Q^+(3, 8)$, where $\mathbb{F}_8$ is defined using the polynomial $X^3 + X + 1$ (of numerical rank $8 + 2 + 1 = 11$). In Table 22, Orbiter activities for orthogonal spaces are shown. The command

```
orbiter.out -v 2 \
   -define F -finite_field -q 2 -end -end \ 
   -define O -orthogonal_space 1 6 F -without_group -end -end \ 
   -with O -do -orthogonal_space_activity -cheat_sheet_orthogonal -end
```

produces a cheat sheet for the quadric $Q^+(5, 2)$. This is the Klein quadric from Section 17. Orbiter produces the following output. At the top is the tactical decomposition of the incidence matrix between points and lines. After that, the points and lines are listed (output shortened):
The number of points is 35 points:

- $P_0 = (1, 0, 0, 0, 0, 0)$
- $P_1 = (0, 1, 0, 0, 0, 0)$
- $P_2 = (0, 0, 1, 0, 0, 0)$
- $P_3 = (1, 0, 1, 0, 0, 0)$
- $P_4 = (0, 1, 1, 0, 0, 0)$
- $P_5 = (0, 0, 0, 1, 0, 0)$
- $P_6 = (1, 0, 0, 1, 0, 0)$
- $P_7 = (0, 1, 0, 1, 0, 0)$
- $P_8 = (1, 1, 1, 0, 0, 0)$
- $P_9 = (0, 0, 0, 0, 1, 0)$
- $P_{10} = (1, 0, 0, 0, 1, 0)$
- $P_{11} = (0, 1, 0, 0, 1, 0)$
- $P_{12} = (0, 0, 1, 0, 1, 0)$
- $P_{13} = (1, 0, 1, 0, 1, 0)$
- $P_{14} = (0, 1, 1, 0, 1, 0)$
- $P_{15} = (0, 0, 0, 1, 1, 0)$
- $P_{16} = (1, 0, 0, 1, 1, 0)$
- $P_{17} = (0, 1, 0, 1, 1, 0)$
- $P_{18} = (1, 1, 1, 1, 1, 0)$
- $P_{19} = (0, 0, 0, 0, 0, 1)$
- $P_{20} = (1, 0, 0, 0, 0, 1)$
- $P_{21} = (0, 1, 0, 0, 0, 1)$
- $P_{22} = (0, 0, 1, 0, 0, 1)$
\[ P_{23} = (1, 0, 1, 0, 0, 1) \]
\[ P_{24} = (0, 1, 1, 0, 0, 1) \]
\[ P_{25} = (0, 0, 0, 1, 0, 1) \]
\[ P_{26} = (1, 0, 0, 1, 0, 1) \]
\[ P_{27} = (0, 1, 0, 1, 0, 1) \]
\[ P_{28} = (1, 1, 1, 0, 0, 1) \]
\[ P_{29} = (1, 1, 0, 0, 1, 1) \]
\[ P_{30} = (1, 1, 1, 0, 1, 1) \]
\[ P_{31} = (1, 1, 0, 1, 1, 1) \]
\[ P_{32} = (0, 0, 1, 1, 1, 1) \]
\[ P_{33} = (1, 0, 1, 1, 1, 1) \]
\[ P_{34} = (0, 1, 1, 1, 1, 1) \]

The number of lines is 105

\[ L_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \{P_0, P_{32}, P_{33}\} \]

\[ L_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \{P_1, P_{32}, P_{34}\} \]

\[ L_{104} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \{P_8, P_9, P_{18}\} \]

Orbiter has enumerators for points and lines. For spaces that are small enough, the cheat sheet lists points and lines in the Orbiter ordering. The command

```
orbiter.out -v 20 \
-define F -finite_field -q 64 -end -end \
-define O -orthogonal_space 1 6 F -without_group -end -end \
-with O -do -orthogonal_space_activity \
-unrank_line_through_two_points 15447347 15225451 \
-end
```

computes the Orbiter rank of the line through the points with rank 15447347 and 15225451, respectively. The rank of the line is 16767254. Note that all of these ranks are ranks in the orthogonal geometry. They are not ranks in the projective geometry or in the grassmannian of lines.
19 Hermitian Varieties

Orbiter has enumerators for points of the hermitian variety $H(k, Q)$. Here, $Q$ is a square, and so $q = \sqrt{Q}$ is an integer. The equation of the variety is

$$\sum_{i=0}^{k} X_i^{q+1} = 0.$$ 

The command

```
orbiter.out -define F -finite_field -q 4 -end -end \
    -with F -do -finite_field_activity -cheat_sheet_hermitian 2 -end
```

produces a cheat sheet for the variety $H(2, 4)$:

The Hermitian variety $H(2, 4)$ contains 9 points:

- $P_0 = (1, 1, 0) = 4$
- $P_1 = (2, 1, 0) = 5$
- $P_2 = (3, 1, 0) = 6$
- $P_3 = (1, 0, 1) = 7$
- $P_4 = (2, 0, 1) = 8$
- $P_5 = (3, 0, 1) = 9$
- $P_6 = (0, 1, 1) = 10$
- $P_7 = (0, 2, 1) = 13$
- $P_8 = (0, 3, 1) = 17$

All points: (4, 5, 6, 7, 8, 9, 10, 13, 17)

The command

```
orbiter.out -define F -finite_field -q 4 -end -end \
    -with F -do -finite_field_activity -cheat_sheet_hermitian 3 -end
```

produces a cheat sheet for the variety $H(3, 4)$:

The Hermitian variety $H(3, 4)$ contains 45 points:
$P_0 = (1, 1, 0, 0) = 5 \quad P_{23} = (3, 3, 1, 1) = 52$
$P_1 = (2, 1, 0, 0) = 6 \quad P_{24} = (0, 0, 1, 1) = 38$
$P_2 = (3, 1, 0, 0) = 7 \quad P_{25} = (1, 1, 2, 1) = 58$
$P_3 = (1, 0, 1, 0) = 8 \quad P_{26} = (2, 1, 2, 1) = 59$
$P_4 = (2, 0, 1, 0) = 9 \quad P_{27} = (3, 1, 2, 1) = 60$
$P_5 = (3, 0, 1, 0) = 10 \quad P_{28} = (1, 2, 2, 1) = 62$
$P_6 = (0, 1, 1, 0) = 11 \quad P_{29} = (2, 2, 2, 1) = 63$
$P_7 = (0, 2, 1, 0) = 15 \quad P_{30} = (3, 2, 2, 1) = 64$
$P_8 = (0, 3, 1, 0) = 19 \quad P_{31} = (1, 3, 2, 1) = 66$
$P_9 = (1, 0, 0, 1) = 23 \quad P_{32} = (2, 3, 2, 1) = 67$
$P_{10} = (2, 0, 0, 1) = 24 \quad P_{33} = (3, 3, 2, 1) = 68$
$P_{11} = (3, 0, 0, 1) = 25 \quad P_{34} = (0, 0, 2, 1) = 53$
$P_{12} = (0, 1, 0, 1) = 26 \quad P_{35} = (1, 1, 3, 1) = 74$
$P_{13} = (0, 2, 0, 1) = 30 \quad P_{36} = (2, 1, 3, 1) = 75$
$P_{14} = (0, 3, 0, 1) = 34 \quad P_{37} = (3, 1, 3, 1) = 76$
$P_{15} = (1, 1, 1, 1) = 4 \quad P_{38} = (1, 2, 3, 1) = 78$
$P_{16} = (2, 1, 1, 1) = 43 \quad P_{39} = (2, 2, 3, 1) = 79$
$P_{17} = (3, 1, 1, 1) = 44 \quad P_{40} = (3, 2, 3, 1) = 80$
$P_{18} = (1, 2, 1, 1) = 46 \quad P_{41} = (1, 3, 3, 1) = 82$
$P_{19} = (2, 2, 1, 1) = 47 \quad P_{42} = (2, 3, 3, 1) = 83$
$P_{20} = (3, 2, 1, 1) = 48 \quad P_{43} = (3, 3, 3, 1) = 84$
$P_{21} = (1, 3, 1, 1) = 50 \quad P_{44} = (0, 0, 3, 1) = 69$
$P_{22} = (2, 3, 1, 1) = 51$

All points: ( 5, 6, 7, 8, 9, 10, 11, 15, 19, 23, 24, 25, 26, 30, 34, 4, 43, 44, 46, 47, 48, 50, 51, 52, 38, 58, 59, 60, 62, 63, 64, 66, 67, 68, 53, 74, 75, 76, 78, 79, 80, 82, 83, 84, 69 )

Coincidentally, this Hermitian variety is the Hirschfeld cubic surface over $\mathbb{F}_4$. 
20 Permutation Groups

Orbiter provides many kinds of permutation groups. A permutation group is a group with a fixed action on a set, called the permutation domain. One group can have many actions. A given action can induce a new action on some combinatorial, geometrical or algebraic structure. These induced actions create new permutation groups, homomorphic to the original one.

Permutation domains are often large, so special methods are used to avoid listing all elements explicitly. Instead, the method of ranking and unranking is used to establish a fixed bijection between the permutation domain and an integer interval. Specifically, if \( P \) is a permutation domain of size \( N = |P| \), we establish a bijection between \( P \) and the integer interval \([0, N - 1]\). Whenever we need to compute the image of an element \( i \in [0, N - 1] \) under a group element \( g \), we first unrank the element in the permutation domain associated with \( i \) in the permutation domain. We then apply \( g \) to obtain the image, which in turn is converted back into an integer \( j \in [0, N - 1] \). We can say that \( i \) maps to \( j \) under \( g \). The process of converting integers to elements of the permutation domain and vice-versa is called unranking and ranking. We have seen ranking and unranking for projective points in Section 13. In Section 21, we will discuss matrix groups over finite fields. The enumerator for projective points from Section 13 is used to realize the permutation domain implicitly. This enumerator uses the bijection between field elements of \( \mathbb{F}_q \) and the integers in the interval \([0, q - 1]\) that we discussed in Sections 8 and 10. For an affine group, a different enumerator is used to describe a different permutation domain. This enumerator is based on the base-\( q \) representation of integers, which associates a vector over \( \mathbb{F}_q \) of length \( n \) with an integer between 0 and \( q^n \).

A permutation group is represented using a base and stabilizer chain (cf. [26, 50]). Strong generating sets are used to generate the groups. For the projective linear groups, the standard frame can be used as base. Orbiter uses this base and the associated strong generating set described in [9].

Group elements can be specified using the coded form. This is a vector of integers which describes a group element. For instance, for an element of a linear group, the coding consists of the entries of associated matrix (for projective matrix groups, the coding is not unique as scalar multiples of the matrix describe the same group element). For semilinear matrix groups, an extra integer is used to describe the associated field automorphism as a power of the generator of the group of field automorphisms (the Frobenius endomorphism as transformation). For affine groups, the coding consists of a matrix, a vector and possible a integer describing a field automorphism. Generating sets of groups can be specified by listing a sequence group elements in coded form. Whenever a generating set is expected, the user is asked to provide the group order. This allows Orbiter to use (fast) randomized algorithms to generate the group with a check that the answer is correct.
21 Linear Groups

Orbiter provides support for matrix groups and their various permutation representationes. For background information about the classical groups of matrices over finite fields, see cf. [55]. Any group in Orbiter is associated with a permutation action. There can be multiple actions for the same group though. Using homomorphisms of permutation groups, new actions can be formed from old actions. Basic group actions are projective, affine, and general linear, as well as orthogonal, unitary and tensor product. Product actions can be defined also. In order to establish a permutation representation, the elements (aka points) of the permutation domain need to be made available. One way would be to make a table of all elements in the permutation domain. However, this would be time and memory intensive. For this reason, a different technique is used that creates points only when needed. The way this works is that the permutation domain is encoded implicitly, using a fixed bijection to a suitable integer interval (zero based), called the domain. Whenever we want the $i$th point in the domain, we can call a function that produces it. Conversely, whenever we have a point, we can call a function that tells us what the associated index in the domain. This is facilitated by two mutually inverse functions. The rank function turns a point into an index. The unrank function turns an index in the domain into a point. Rank and unrank functions are helpful because they eliminate the need for tables of all objects. The ranks lead to rather compact storage of objects in files. The objects can be reconstructed from the ranks.

Let $V \simeq \mathbb{F}_q^n$ be a finite dimensional vector space over $\mathbb{F}_q$. The set of subspaces of $V$ form the projective geometry $\text{PG}(n-1, q)$.

Let $\pi$ be a projective space. A collineation of a projective space $\pi$ is a bijective mapping from the points of $\pi$ to themselves which preserves collinearity. That is, a collineation $\varphi$ maps any three collinear points $P, Q, R$ to another collinear triple $\varphi(P), \varphi(Q), \varphi(R)$. The collineations form a group with respect to composition, the collineation group. If $M$ is the matrix of an endomorphism, then $\Psi_M$ is the induced map on projective space. By considering the homomorphism $M \mapsto \Psi_M$, the group $\text{GL}(n+1, q)$ of invertible endomorphisms becomes a subgroup of the group of collineations of $\text{PG}(n, q)$. This is the projectivity group $\text{PGL}(n+1, q)$. It is isomorphic to $\text{GL}(n+1, q)/\mathbb{F}_q^\times$. Another source of collineations is this: Let $\Phi \in \text{Aut}(\mathbb{F}_q)$ be a field automorphism. Then $\Phi$ acts on projective space by sending $P(x)$ to $P(x\Phi)$. This map is another type of collineation, called automorphic collineation. This way, $\text{Aut}(\mathbb{F}_q)$ can be considered another subgroup of the group of collineations. If $q = p^h$ for some prime $p$ and some integer $h$ then

$$\Phi_0 : \mathbb{F}_q \to \mathbb{F}_q, \ x \mapsto x^p$$

is a generator for the cyclic group $C_h \simeq \text{Aut}(\mathbb{F}_q)$. The collineation group of $\text{PG}(n, q)$ ($n \geq 2$) is isomorphic to the semidirect product of the projectivity group and the automorphism group of the field. The collineation group is $\text{PGL}(n+1, q) = \text{PGL}(n+1, q) \rtimes \text{Aut}(\mathbb{F}_q)$. We use the following notation for elements of $\text{PGL}(n+1, q)$. Let $\Phi_0$ be a generator for $\text{Aut}(\mathbb{F}_q)$ and let $M \in \text{GL}(n+1, q)$. The map

$$(\Psi_M, \Phi_0^k) : \text{PG}(n, q) \to \text{PG}(n, q), \ P(x) \mapsto P(y), \ y = (x \cdot M)^{\Phi_0^k}$$
is denoted as
\[ M_k. \] (1)
The identity element is \( I_0 \), where \( I \) is the identity matrix and 0 is the residue class modulo \( h \). The rules for multiplication and inversion in the collineation group are given as
\[ M_k \cdot N_l = \left( M \cdot N^{\Phi^{-k}} \right)_{k+l}, \] (2)
\[ (M_k)^{-1} = \left( \left( M^{-1} \right)^{\Phi^k} \right)_{-k}. \] (3)
The affine group \( AGL(n, q) \) is the semidirect product of \( GL(n, q) \) with \( \mathbb{F}_q^n \). The affine semilinear group \( AΓL(n, q) \) is the semidirect product of \( AGL(n, q) \) with \( \text{Aut}(\mathbb{F}_q) \). The elements of \( AΓL(n, q) \) are triples
\[ M_{a,k} := (M, a, k) \in \text{GL}(n, q) \times \mathbb{F}_q^n \times \text{Aut}(\mathbb{F}_q), \]
which act on \( \mathbb{F}_q^n \):
\[ \left( x, (M, a, k) \right) \mapsto \left( x \cdot M + a \right)^{\Phi^k}. \]
The multiplication in \( AΓL(n, q) \) is
\[ M_{a,k} \cdot N_{b,l} = (MN)_{aN^\Phi^{-k} + b^\Phi^{-k}, k+l}. \]
The inverse of an element is
\[ \left( M_{a,k} \right)^{-1} = \left( M^{-1} \right)_{a^\Phi^k M^{-1}, -k}. \]
A correlation is a one-to-one mapping between the set of points and the set of hyperplanes which reverses incidence. So, if \( \rho \) is a correlation and \( P \) is a point and \( \ell \) is a hyperplane then \( P^\rho \) is a hyperplane and \( \ell^\rho \) is a point and
\[ \ell^\rho \in P^\rho \iff P \in \ell. \]
A correlation of order two is called polarity. The standard polarity is the map
\[ \rho : \mathcal{P} \leftrightarrow \mathcal{L}, \ P(x) \leftrightarrow [x]. \]
A group \( G \) can act on \( V \) in one of the types listed in Table 23. The elements of finite fields are represented as integers as described in Sections 8 and 10. The elements of the various sets on which the group acts are encoded as integers also. For instance,
\[ \text{orbiter.out -define G -linear_group -PGL 4 2 -end -end} \]
creates the group \( G = \text{PGL}(4, 2) \) acting on the 15 elements of \( \mathbb{S}r_1(\mathbb{F}_2^3) \). In this command, \( G \) is a label that can be chosen freely. The main purpose of labeling the group is to be able to refer to it later by its label. The basic types of groups are listed in Table 24. The command
<table>
<thead>
<tr>
<th>Type</th>
<th>Perm. Domain</th>
<th>Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>General linear GL$(n, q)$</td>
<td>all vectors of $V$</td>
<td>$q^n$</td>
</tr>
<tr>
<td>Affine AGL$(n, q)$</td>
<td>all vectors of $V$</td>
<td>$q^n$</td>
</tr>
<tr>
<td>Projective PGL$(n, q)$</td>
<td>$\mathfrak{S}_r(V)$</td>
<td>$\frac{q^n-1}{q-1}$</td>
</tr>
<tr>
<td>Wreath product GL$(d, q) \wr \text{Sym}(n)$</td>
<td>$\mathfrak{S}_r((\mathbb{F}_q^d)\otimes^n)$ extended</td>
<td>$n + nq^d + \frac{q^n-1}{q-1}$</td>
</tr>
<tr>
<td>Orthogonal PGO$(n, q)$</td>
<td>$Q(V)$</td>
<td>$\frac{q^{n-1} - 1}{q - 1}$</td>
</tr>
<tr>
<td>Orthogonal PGO$^+(n, q)$</td>
<td>$Q^+(V)$</td>
<td>$\frac{(q^{n/2} - 1)(q^{(n-2)/2} + 1)}{q - 1}$</td>
</tr>
<tr>
<td>Orthogonal PGO$^-(n, q)$</td>
<td>$Q^-(V)$</td>
<td>$\frac{(q^{n/2} + 1)(q^{(n-2)/2} - 1)}{q - 1}$</td>
</tr>
</tbody>
</table>

Table 23: Basic actions

```
orbiter.out -v 2 \
   -draw_options -rad 200 -end \n   -define F2 -finite_field -q 2 -end -end \n   -define G -linear_group -PGL 4 F2 -end -end \n   -with G -do \n   -group_theoretic_activities \n   -report \n   -end

pdflatex PGL_4_2_report.tex
```

creates a latex report for the group which includes the base and strong generating set, as well as the Schreier trees associated with the stabilizer chain. The report also gives information about the underlying field and the projective geometry.

**The Group PGL$(4, 2)$**

The order of the group PGL$(4, 2)$ is 20160
The group acts on a set of size 15
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>-GL</td>
<td>$n \ q$</td>
<td>GL($n, q$)</td>
</tr>
<tr>
<td>-GGL</td>
<td>$n \ q$</td>
<td>ΓL($n, q$)</td>
</tr>
<tr>
<td>-SL</td>
<td>$n \ q$</td>
<td>SL($n, q$)</td>
</tr>
<tr>
<td>-SSL</td>
<td>$n \ q$</td>
<td>ΣL($n, q$)</td>
</tr>
<tr>
<td>-PGL</td>
<td>$n \ q$</td>
<td>PGL($n, q$)</td>
</tr>
<tr>
<td>-PGGL</td>
<td>$n \ q$</td>
<td>PΓL($n, q$)</td>
</tr>
<tr>
<td>-PSL</td>
<td>$n \ q$</td>
<td>PSL($n, q$)</td>
</tr>
<tr>
<td>-PSSL</td>
<td>$n \ q$</td>
<td>PΣL($n, q$)</td>
</tr>
<tr>
<td>-AGL</td>
<td>$n \ q$</td>
<td>AGL($n, q$)</td>
</tr>
<tr>
<td>-AGGL</td>
<td>$n \ q$</td>
<td>AΓL($n, q$)</td>
</tr>
<tr>
<td>-ASL</td>
<td>$n \ q$</td>
<td>ASL($n, q$)</td>
</tr>
<tr>
<td>-ASSL</td>
<td>$n \ q$</td>
<td>AΣL($n, q$)</td>
</tr>
<tr>
<td>-PGO</td>
<td>$n \ q$</td>
<td>PGO($n, q$)</td>
</tr>
<tr>
<td>-PGOp</td>
<td>$n \ q$</td>
<td>PGO^+(n, q)</td>
</tr>
<tr>
<td>-PGOm</td>
<td>$n \ q$</td>
<td>PGO^-(n, q)</td>
</tr>
<tr>
<td>-PGGO</td>
<td>$n \ q$</td>
<td>PΓO($n, q$)</td>
</tr>
<tr>
<td>-PGGOp</td>
<td>$n \ q$</td>
<td>PΓO^+(n, q)</td>
</tr>
<tr>
<td>-PGGOm</td>
<td>$n \ q$</td>
<td>PΓO^-(n, q)</td>
</tr>
<tr>
<td>-GL_{d \ q \ wr \ Sym \ n}</td>
<td>$d \ q \ n$</td>
<td>GL($d, q$) \ ⋊ \ Sym(n)</td>
</tr>
</tbody>
</table>

Table 24: Basic types of Orbiter matrix groups
Strong generators for a group of order 20160:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
\end{pmatrix},
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
\end{pmatrix},
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 \\
\end{pmatrix},
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
\end{pmatrix},
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix},
\begin{pmatrix}
1,0,0,0,1,0,0,0,0,1,0,1,0,0,1, \\
1,0,0,0,1,0,0,0,0,1,0,0,1,0,1, \\
1,0,0,0,1,0,0,0,0,1,0,0,0,1,1, \\
1,0,0,0,1,0,0,0,0,1,0,0,1,0,1, \\
1,0,0,0,1,0,0,0,0,1,0,0,0,1,0, \\
0,1,0,0,1,0,0,0,0,1,0,0,0,1,0,1, \\
1,0,0,0,1,0,0,0,0,1,0,0,0,1,0, \\
0,1,0,0,1,0,0,0,0,1,0,0,0,1,0,1, \\
1,0,0,0,1,0,0,0,0,1,0,0,0,1,0, \\
1,0,0,0,1,0,0,0,0,1,0,0,0,1,0, \\
1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1, \\
0,1,0,0,1,0,0,0,0,1,0,0,0,1,0,1,
\end{pmatrix}
\]

The Action

Group action $\text{PGL}(4,2)$ of degree 15
We act on the following set:

\[
\begin{align*}
0 &= (1, 0, 0, 0) \\
1 &= (0, 1, 0, 0) \\
2 &= (0, 0, 1, 0) \\
3 &= (0, 0, 0, 1) \\
4 &= (1, 1, 1, 1) \\
5 &= (1, 1, 0, 0) \\
6 &= (1, 0, 1, 0) \\
7 &= (0, 1, 0, 0) \\
8 &= (1, 1, 1, 0) \\
9 &= (1, 0, 0, 1) \\
10 &= (0, 1, 0, 1) \\
11 &= (1, 1, 0, 1) \\
12 &= (0, 0, 1, 1) \\
13 &= (1, 0, 1, 1) \\
14 &= (0, 1, 1, 1)
\end{align*}
\]

The group is a matrix group.
The group acts on projective space $\text{PG}(3, 2)$
\[
q = 2 \\
p = 2 \\
e = 1 \\
n = 3
\]
Number of points = 15
Number of lines = 35
Number of lines on a point = 7
Number of points on a line = 3

**The finite field** $\mathbb{F}_2$

$Z_i = \log_\alpha (1 + \alpha^i)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\gamma_i$</th>
<th>$-\gamma_i$</th>
<th>$\gamma_i^{-1}$</th>
<th>$\log_\alpha (\gamma_i)$</th>
<th>$\alpha^i$</th>
<th>$Z_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DNE</td>
<td>DNE</td>
<td>1</td>
<td>DNE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>DNE</td>
</tr>
</tbody>
</table>

$+$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$\cdot$

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

$1^0 \equiv 1$

$1^1 \equiv 1$

**Base and Stabilizer Chain**

Group order 20160
tl=15, 14, 12, 8,
Base: (0,1,2,3)
Strong generators for a group of order 20160:

$$\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1
\end{pmatrix}
, 
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}
, 
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{pmatrix}
, 
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

1,0,0,0,1,0,0,0,0,1,0,1,0,0,1,0,0,0,0,1,0,1,0,0,1,0,0,0,1,0,1,0,0,0,1,0,0,0,1,1,
Stabilizer chain

<table>
<thead>
<tr>
<th>Level</th>
<th>Base pt</th>
<th>Orbit length</th>
<th>Subgroup order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>15</td>
<td>20160</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>1344</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>12</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Basic Orbit 0

Basic orbit 0 has size 15
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
Basic Orbit 1

Basic orbit 1 has size 14
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

Basic Orbit 2

Basic orbit 2 has size 12
2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14
Basic orbit 3 has size 8
3, 4, 9, 10, 11, 12, 13, 14
GAP export:

Generators in GAP format are:
\[
G := \text{Group}([\(4, 10\)(5, 15)(11, 12)(13, 14),
(4, 11)(5, 14)(10, 12)(13, 15),
(4, 13)(5, 12)(10, 14)(11, 15),
(3, 4)(7, 10)(8, 11)(9, 12),
(2, 3)(6, 7)(11, 13)(12, 14),
(1, 2)(7, 8)(10, 11)(14, 15)]);
\]

Magma export:

\[
G := \text{GeneralLinearGroup}(4, \text{GF}(2));
H := \text{sub}< G | [1,0,0,0, 0,1,0,0, 0,0,1,0, 1,0,0,1],
[1,0,0,0, 0,1,0,0, 0,0,1,0, 0,1,0,1],
[1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,1,1],
[1,0,0,0, 0,1,0,0, 0,0,0,1, 0,0,1,0],
[1,0,0,0, 0,0,1,0, 0,1,0,0, 0,0,0,1],
[0,1,0,0, 1,0,0,0, 0,0,1,0, 0,0,0,1] >;
\]

Compact form:

Generators in compact permutation form are:
\[
6 15
0 1 2 9 14 5 6 7 8 3 11 10 13 12 4
0 1 2 10 13 5 6 7 8 11 3 9 14 4 12
0 1 2 12 11 5 6 7 8 13 14 4 3 9 10
0 1 3 2 4 5 9 10 11 6 7 8 12 13 14
\]
The base has length 4
The basic orbits are:
Basic orbit 0 is orbit of 0 of length 15
Basic orbit 1 is orbit of 1 of length 14
Basic orbit 2 is orbit of 2 of length 12
Basic orbit 3 is orbit of 3 of length 8

The command

```
orbiter.out -v 2 \
  -draw_options -rad 200 -end \ 
  -define F2 -finite_field -q 2 -end -end \ 
  -define G -linear_group -PGO 5 F2 -end -end \ 
  -with G -do \ 
  -group_theoretic_activities \ 
  -report \ 
  -end
```

`pdflatex PGO_5_2_report.tex`

creates the group PGO(5, 2) acting on the 15 points of the $Q(4, 2)$ quadric. The following latex report is produced:

**The Group PGO(5, 2)**

The order of the group PGO(5, 2) is 720
The group acts on a set of size 15
Strong generators for a group of order 720:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]
The Action

Group action PGO(5,2) of degree 15
We act on the following set:

0 = (0, 1, 0, 0, 0) 8 = (0, 1, 1, 1, 1)
1 = (0, 0, 1, 0, 0) 9 = (1, 1, 1, 0, 0)
2 = (0, 0, 0, 1, 0) 10 = (1, 1, 1, 1, 0)
3 = (0, 1, 0, 1, 0) 11 = (1, 1, 1, 0, 1)
4 = (0, 0, 1, 1, 0) 12 = (1, 0, 0, 1, 1)
5 = (0, 0, 0, 0, 1) 13 = (1, 1, 1, 0, 1)
6 = (0, 1, 0, 0, 1) 14 = (1, 0, 1, 1, 1)
7 = (0, 0, 1, 0, 1)

The group is a matrix group.
The base action is on projective space PG(4,2)
q = 2
p = 2
e = 1
n = 4
Number of points = 31
Number of lines = 155
Number of lines on a point = 15
Number of points on a line = 3
The finite field $\mathbb{F}_2$

$Z_i = \log_\alpha (1 + \alpha^i)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\gamma_i$</th>
<th>$-\gamma_i$</th>
<th>$\gamma_i^{-1}$</th>
<th>$\log_\alpha(\gamma_i)$</th>
<th>$\alpha^i$</th>
<th>$Z_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DNE</td>
<td>DNE</td>
<td>1</td>
<td>DNE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$+$

| 0   | 0   | 1   |
| 0   | 0   | 1   |
| 1   | 1   | 0   |

$\cdot$

| 1   |
| 1   |

$1^0 \equiv 1$

$1^1 \equiv 1$

Base and Stabilizer Chain

Group order 720
$tl=15, 8, 3, 1, 1, 2,$
Base: $(0, 1, 2, 3, 4, 5)$
Strong generators for a group of order 720:

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

1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,1,0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,
Stabilizer chain

<table>
<thead>
<tr>
<th>Level</th>
<th>Base pt</th>
<th>Orbit length</th>
<th>Subgroup order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>15</td>
<td>720</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Basic Orbit 0

Basic orbit 0 has size 15
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
Basic Orbit 1

Basic orbit 1 has size 8
1, 4, 7, 8, 9, 10, 11, 14

Basic Orbit 2

Basic orbit 2 has size 3
2, 5, 12

Basic Orbit 3

Basic orbit 3 has size 1
3
Basic Orbit 4

Basic orbit 4 has size 1

Basic Orbit 5

Basic orbit 5 has size 2

5, 12

GAP export:

Generators in GAP format are:
G := Group([[(6, 13)(7, 14)(8, 15)(9, 12),
(3, 13)(4, 14)(5, 15)(9, 11),
(2, 12)(3, 14)(4, 13)(8, 10),
(2, 8, 9, 10, 12, 15)(3, 14, 7)(4, 13, 6)(5, 11),
(1, 10)(4, 11)(7, 12)(9, 14),
(1, 7)(3, 5)(4, 9)(10, 12)(11, 14)(13, 15)]);

Magma export:

Compact form:

Generators in compact permutation form are:
6 15
0 1 2 3 4 12 13 14 11 9 10 8 5 6 7
0 1 12 13 14 5 6 7 10 9 8 11 2 3 4
0 11 13 12 4 5 6 9 8 7 10 1 3 2 14
0 7 13 12 10 3 2 8 9 11 4 14 5 6 1
9 1 2 10 4 5 11 7 13 0 3 6 12 8 14
The base has length 6
The basic orbits are:
Basic orbit 0 is orbit of 0 of length 15
Basic orbit 1 is orbit of 1 of length 8
Basic orbit 2 is orbit of 2 of length 3
Basic orbit 3 is orbit of 3 of length 1
Basic orbit 4 is orbit of 4 of length 1
Basic orbit 5 is orbit of 5 of length 2

The command

```
orbiter.out -v 3 -define G -linear_group -GL_d_q_wr_Sym_n 2 2 4 -end -end
```

creates the group $GL(2, 2) \rtimes Sym(4)$ acting on $PG(15, 2)$ extended by a set of 20 extra points.
The extra points are associated with the actions of the components of the wreath product:
Four points form a permutation domain for the permutation part $Sym(4)$. An additional
$16 = 4 \times 4$ points form four permutation domains of $GL(2, 2)$, one for each factor.
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Janko1</td>
<td></td>
<td>first Janko group, needs PGL(7, 11)</td>
</tr>
<tr>
<td>-monomial</td>
<td></td>
<td>subgroup of monomial matrices</td>
</tr>
<tr>
<td>-diagonal</td>
<td></td>
<td>subgroup of diagonal matrices</td>
</tr>
<tr>
<td>-null_polarity_group</td>
<td></td>
<td>null polarity group</td>
</tr>
<tr>
<td>-symplectic_group</td>
<td></td>
<td>symplectic group</td>
</tr>
<tr>
<td>-singer</td>
<td>$k$</td>
<td>subgroup of index $k$ in the Singer cycle</td>
</tr>
<tr>
<td>-singer_and_frobenius</td>
<td>$k$</td>
<td>subgroup of index $k$ in the Singer cycle, extended by the Frobenius automorphism of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$</td>
</tr>
<tr>
<td>-borel_upper</td>
<td></td>
<td>Borel subgroup of upper triangular matrices</td>
</tr>
<tr>
<td>-borel_lower</td>
<td></td>
<td>Borel subgroup of lower triangular matrices</td>
</tr>
<tr>
<td>-identity_group</td>
<td></td>
<td>identity subgroup</td>
</tr>
<tr>
<td>-subgroup_from_file</td>
<td>$f \ l$</td>
<td>read subgroup from file $f$ and give it the label $l$</td>
</tr>
<tr>
<td>-orthogonal</td>
<td>$\epsilon$</td>
<td>orthogonal group $O^\epsilon(n, q)$, with $\epsilon \in {\pm 1}$ when $n$ is even</td>
</tr>
<tr>
<td>-subgroup_by_generators</td>
<td>$l \ o \ n \ s_1 \ldots s_n$</td>
<td>Generate a subgroup from generators. The label “$l$” is used to denote the subgroup; $o$ is the order of the subgroup; $n$ is the number of generators and $s_1, \ldots, s_n$ are the generators for the subgroup in string representation.</td>
</tr>
</tbody>
</table>

**Table 25: Commands for creating subgroups**

### 22 Subgroups

There are many ways to create subgroups of a group. Table 25 lists some commands to do so. For instance, the command

```
orbiter.out -v 3 -define G -linear_group -PGL 7 11 -Janko1 -end -end
```

creates the first Janko group as a subgroup of PGL(7, 11).

The command

```
orbiter.out -v 3 -define G -linear_group -PGL 3 11 -singer 1 -end -end
```

can be used to create the Singer cycle. The Singer cycle in $\text{GL}(d, q)$ is a generator for a subgroup of order $q^d - 1$. It induces an element of order $\frac{q^d - 1}{q - 1}$ on the associated projective geometry $\text{PG}(d - 1, q)$. The additional integer parameter $k$ after the `-singer` command can be used to create the subgroup of index $k$ of the Singer cycle.
The command

```
orbiter.out -v 3 -define G -linear_group -PGL 3 11 \
   -singer_and_frobenius 19 -end -end
```

can be used to create a subgroup of index 19 of the Singer cycle of PG(2, 11), extended by a group of order 3 that arises from the field extension \( \mathbb{F}_{2}^{11} \) over \( \mathbb{F}_{11} \). The group created by this command has order 21.

The quaternion group is a group of order 8 generated by the following matrices over \( \mathbb{R} \):

\[
i = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad j = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}, \quad k = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.
\]

It is isomorphic to a subgroup of SL(2,3). To perform the embedding, we need to replace the real number \(-1\) by the corresponding field element in \( \mathbb{F}_3 \). Because Orbiter field elements are integers in the interval \([0, \ldots, q-1]\), we replace \(-1\) by 2. The Orbiter command

```
orbiter.out -v 3 -define G \
   -linear_group -SL 2 3 \ 
   -subgroup_by_generators "quaternion" "8" 3 \ 
   "1,1,1,2" \ 
   "2,1,1,1" \ 
   "0,2,1,0" \ 
   -end -end \ 
   -with G -do \ 
   -group_theoretic_activities \ 
   -print_elements_tex \ 
   -group_table \ 
   -report \ 
   -end
```

creates the group. The command produces the list of group elements shown below.

```
Element 0 / 8 of order 1:

\[
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]

(0)(1)(2)(3)(4)(5)(6)(7)(8)

Element 1 / 8 of order 4:

\[
\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}
\]

(0)(1, 5, 2, 7)(3, 4, 6, 8)

84
Element 2 / 8 of order 2:
\[
\begin{bmatrix}
2 & 0 \\
0 & 2
\end{bmatrix}
\]
(0)(1, 2)(3, 6)(4, 8)(5, 7)

Element 3 / 8 of order 4:
\[
\begin{bmatrix}
1 & 2 \\
2 & 2
\end{bmatrix}
\]
(0)(1, 7, 2, 5)(3, 8, 6, 4)

Element 4 / 8 of order 4:
\[
\begin{bmatrix}
1 & 1 \\
1 & 2
\end{bmatrix}
\]
(0)(1, 4, 2, 8)(3, 7, 6, 5)

Element 5 / 8 of order 4:
\[
\begin{bmatrix}
0 & 1 \\
2 & 0
\end{bmatrix}
\]
(0)(1, 3, 2, 6)(4, 5, 8, 7)

Element 6 / 8 of order 4:
\[
\begin{bmatrix}
2 & 2 \\
2 & 1
\end{bmatrix}
\]
(0)(1, 8, 2, 4)(3, 5, 6, 7)

Element 7 / 8 of order 4:
\[
\begin{bmatrix}
0 & 2 \\
1 & 0
\end{bmatrix}
\]
(0)(1, 6, 2, 3)(4, 7, 8, 5)

The group table is created as csv file:

```
Row,C0,C1,C2,C3,C4,C5,C6,C7
0,0,1,2,3,4,5,6,7
1,1,2,3,0,5,6,7,4
```
The group of the cube can be created over the field $\mathbb{F}_3$:

```
orbiter.out -v 3 -define G \
  -linear_group -GL 3 3 \n    -subgroup_by_generators "cube" "48" 3 \n      "0,1,0,2,0,0,0,0,1" \n      "0,0,1,0,1,0,2,0,0" \n      "2,0,0,0,1,0,0,0,1" \n    -end -end \n  -with G -do \n    -group_theoretic_activities \n    -print_elements_tex \n    -report \n  -end
```

The tetrahedral subgroup can be created as well:

```
orbiter.out -v 3 -define G \
  -linear_group -GL 3 3 \n    -subgroup_by_generators "tetra" "12" 2 \n      "0,1,0,0,0,1,1,0,0" \n      "0,0,1,2,0,0,0,2,0" \n    -end -end \n  -with G -do \n    -group_theoretic_activities \n    -print_elements_tex \n    -report \n  -end
```
23 Linear Groups, Advanced Topics

It is sometimes desired to be able to control the finite field that is used in the construction of a matrix group. For prime fields, this is not an issue. For extension fields, the choice of polynomial does matter, as the generators depend on specific choices made for the finite field. This is particularly relevant when collaborating with other researcher or when the polynomial has been chosen otherwise. Magma and GAP use Conway polynomials. However, Conway polynomials are difficult to compute. Orbiter has a built-in table of primitive polynomials, but they are not necessarily Conway polynomials. As explained in Section 10, Orbiter allows to specify the polynomial that should be used to create the finite field. The next example shows an instance where choosing the polynomial is important. We are recreating a group from the electronic Atlas on finite simple groups [58].

The electronic Atlas of finite simple groups [58] lists generators for $U_3(3)$ as $3 \times 3$ matrices over the field $\mathbb{F}_9$ using the following short Magma [12] program:

```magma
F<w>:=GF(9);
x:=CambridgeMatrix(1,F,3,[
"164",
"506",
"851"]);
y:=CambridgeMatrix(1,F,3,[
"621",
"784",
"066"]);
G<x,y>:=MatrixGroup<3,F|x,y>;
```

The generators are given using the Magma command `CambridgeMatrix`, which allows for more efficient coding of field elements. The field elements are coded as base-3 integers (like in Orbiter) with respect to the Magma version of $\mathbb{F}_9$. The polynomial for $\mathbb{F}_9$ can be determined using the following Magma command (which can be typed into the Magma online calculator at [53])

```magma
F<w>:=GF(9);
print DefiningPolynomial(F);
```

which results in

$.1^2 + 2*$.1 + 2

which is the Magma way of printing the polynomial $X^2 + 2X + 2$. If $\alpha$ is a root of the polynomial over $\mathbb{F}_3$, then

$\alpha^2 = \alpha + 1$.

Regarding the coefficient vector of the polynomial $(1, 2, 2)$ as in integer written in base-3, we obtain

$1 \cdot 3^2 + 2 \cdot 3 + 2 = 17$.

The command
can be used to create $F_9$ using this polynomial. The command

```
-define F -finite_field -q 9 -override_polynomial "17" -end -end
```

creates a symbolic variable $F$ for this specific field $F_9$. In order to create the linear group over this field, the command

```
-linear_group -PGL 3 F -end
```

can be used, where the second argument after the `-PGL` command references the field $F_9$ that we just created through its symbolic name. The desired subgroup can now be created using the command

```
-linear_group -PGL 3 F \
-subgroup_by_generators "U_3_3" "6048" 2 \
"1,6,4, 5,0,6, 8,5,1" \
"6,2,1, 7,8,4, 0,6,6" \
-end
```

Finally, we create a symbolic variable $G$ for the group and invoke a group theoretic activity to produce a report about the group. Here is the full command sequence:

```
orbiter.out -v 3 \n-draw_options -end \n(define F -finite_field -q 9 -override_polynomial "17" -end -end \n(define G -linear_group -PGL 3 F \n-subgroup_by Generators "U_3_3" "6048" 2 \n"1,6,4, 5,0,6, 8,5,1" \n"6,2,1, 7,8,4, 0,6,6" \n-end -end \n-with G -do \n-group_theoretic_activities \n-report -end
```

Group theoretic activities will be discussed in Section 25.

As an example of a large group, consider the Conway group $Co_3$. Following [52], the group can be generated using two matrices of dimension 22 over $F_2$. We use the `-long_string`..`-end_string` commands to enclose the string argument for each generator. This is so that we can spread each matrix over multiple lines, one row at a time. The following command creates a report for the group.

```
orbiter.out -v 6 \n(define G \n-linear_group -PGL 22 2 \n-subgroup_by_generators "Co3" "495766656000" 2 \n-long_string
```

88
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-wedge</td>
<td></td>
<td>action on the exterior square</td>
</tr>
<tr>
<td>-wedge_detached</td>
<td></td>
<td>action on the exterior square. Unlike -wedge, this command does not establish the homomorphism to the original group. Instead, the group is created as subgroup of the larger general linear group.</td>
</tr>
<tr>
<td>-PGL2OnConic</td>
<td></td>
<td>induced action of PGL(2, q) on the conic in the plane PG(2, q)</td>
</tr>
<tr>
<td>-subfield_structure_action</td>
<td>s</td>
<td>action by field reduction to the subfield of index s</td>
</tr>
<tr>
<td>-on_k_subspaces</td>
<td>k</td>
<td>induced action on k dimensional subspaces</td>
</tr>
<tr>
<td>-on_tensors</td>
<td></td>
<td>induced action of GL(d, q) ⋊ Sym(n) on the tensor space</td>
</tr>
<tr>
<td>-on_rank_one_tensors</td>
<td></td>
<td>induced action of GL(d, q) ⋊ Sym(n) on the tensor space</td>
</tr>
<tr>
<td>-restricted_action</td>
<td>s</td>
<td>restricted action on the set s</td>
</tr>
</tbody>
</table>

Table 26: Commands for creating induced or restricted group actions

### 24 Induced Actions

It is possible to create new group actions from old. Table 26 lists some commands to do so. For instance, the command

```
orbiter.out -v 4 \
-define G \
-linear_group -GL_d_q_wr_Sym_n 2 2 3 -on_tensors -end -end
```

creates the group GL(2, 2) ⋊ Sym(3) acting on the 255 elements of PG(7, 2) which are identified with the tensors of type (2, 2, 2) over \( \mathbb{F}_2 \). A look at the report shows how elements of this group are denoted in Orbiter. It is also important to observe that the stabilizer chain is very long and has short subgroup indices. This is because GL(2, 2) ⋊ Sym(3) is originally created on a larger permutation domain. This larger permutation domain allows for short orbits, making the implementation more efficient.

**The Group** GL(2, 2) ⋊ Sym(3)

The order of the group GL(2, 2) ⋊ Sym(3) is 1296
The group acts on a set of size 255
The Action

Group action $\text{GL}(2, 2) \wr \text{Sym}(3)\text{res}255$ of degree 255

Base and Stabilizer Chain

Group order 1296
$tl=3, 2, 1, 3, 2, 3, 2, 3, 2$.

Strong generators for a group of order 1296.

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}; \text{id},$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \text{id},$$

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}; (1, 2),$$

$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; (0, 1)$$

0,1,2,1,0,0,1,1,0,0,1,1,0,1,1,
0,1,2,1,0,0,1,1,0,0,1,0,1,1,0,
0,1,2,1,0,0,1,1,0,1,1,0,0,1,
0,1,2,1,0,0,1,1,0,1,1,0,0,1,
0,1,2,1,0,0,1,1,0,1,1,0,0,1,
0,2,1,1,0,0,1,1,0,0,1,1,0,0,1,
1,0,2,1,0,0,1,1,0,1,0,1,0,0,1,

Stabilizer chain

<table>
<thead>
<tr>
<th>Level</th>
<th>Base pt</th>
<th>Orbit length</th>
<th>Subgroup order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1296</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>432</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>216</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>216</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
It is also possible to restrict the action on all rank-one tensors, as the following example shows:

```
orbiter.out -v 4 \
  -draw_options -end \n  -define G \n  -linear_group -GL_d_q_wr_Sym_n 2 2 3 -on_rank_one_tensors -end -end \n  -with G -do \n  -group_theoretic_activities \n  -report \n  -end
```

This creates an action of degree 27. Because the degree is small, the Orbiter report shows all points in the permutation domain.

### The Group \( GL(2,2) \wr Sym(3) \)

The order of the group \( GL(2,2) \wr Sym(3) \) is 1296
The group acts on a set of size 27

### The Action

Group action \( GL(2,2) \wr Sym(3)_{res27} \) of degree 27
We act on the following set:

\[
\begin{align*}
0 &= (1,0,0,0,0,0,0,0) & 14 &= (0,0,0,0,0,0,0,1,1) \\
1 &= (0,1,0,0,0,0,0,0) & 15 &= (0,0,0,0,1,0,1,0) \\
2 &= (1,1,0,0,0,0,0,0) & 16 &= (0,0,0,0,1,0,0,1) \\
3 &= (0,0,1,0,0,0,0,0) & 17 &= (0,0,0,0,1,1,1,1) \\
4 &= (0,0,0,1,0,0,0,0) & 18 &= (1,0,0,0,1,0,0,0) \\
5 &= (0,0,1,1,0,0,0,0) & 19 &= (0,1,0,0,0,1,0,0) \\
6 &= (1,0,1,0,0,0,0,0) & 20 &= (1,1,0,0,1,1,0,0) \\
7 &= (0,1,0,1,0,0,0,0) & 21 &= (0,0,1,0,0,0,1,0) \\
8 &= (1,1,1,0,0,0,0,0) & 22 &= (0,0,0,1,0,0,0,1) \\
9 &= (0,0,0,0,1,0,0,0) & 23 &= (0,0,1,1,0,0,1,1) \\
10 &= (0,0,0,0,1,0,0,0) & 24 &= (1,0,1,0,1,0,0,1) \\
11 &= (0,0,0,0,1,1,0,0) & 25 &= (0,1,0,1,0,1,0,1) \\
12 &= (0,0,0,0,0,0,1,0) & 26 &= (1,1,1,1,1,1,1,1) \\
13 &= (0,0,0,0,0,0,1,1) \end{align*}
\]
The group of a conic is the group of the projective line. This isomorphism can be realized using an the induced action command \texttt{PGL2OnConic}. The group elements are stored and multiplied in their original form as elements of \texttt{PGL(2,q)}. The action is changed using the induced action on the Veronese variety. Here is an example. We create the collineation group \texttt{PGL(2,8)} of \texttt{PG(1,8)} and act on \texttt{PG(2,8)}:

\begin{verbatim}
onerter.out -v 4 \
 -draw_options -rad 250 -end \
 -define G \
 -linear_group -PGGL 2 8 -PGL2OnConic -end -end \
 -with G -do \
 -group_theoretic_activities \ 
 -report \ 
 -end
\end{verbatim}

This produces the following report. We note how the generators are elements of \texttt{PGL(2,8)}. We note that the action is induced on all of \texttt{PG(2,8)}, not just the conic itself. Note further that the first basic orbit is the conic itself and all other basic orbits are subsets of the conic.
The Group $\text{PGL}(2,8) \text{OnConic}(2, 8)$

The order of the group $\text{PGL}(2,8) \text{OnConic}(2, 8)$ is 1512.

The group acts on a set of size 73.

Strong generators for a group of order 1512:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix},
\begin{bmatrix}
\gamma & 0 \\
0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 \\
\gamma & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 \\
\gamma^2 & 1
\end{bmatrix},
\begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix},
\begin{bmatrix}
1, 0, 0, 1, 1 \\
1, 0, 0, 6, 0 \\
1, 0, 1, 1, 0 \\
1, 0, 2, 1, 0 \\
1, 0, 4, 1, 0 \\
0, 1, 1, 0, 0
\end{bmatrix}
\]

The Action

Group action $\text{PGL}(2,8) \text{OnConic}$ of degree 73

We act on the following set:

\[
0 = (1, 0, 0) \\
1 = (0, 1, 0) \\
2 = (0, 0, 1) \\
3 = (1, 1, 1) \\
4 = (1, 1, 0) \\
5 = (2, 1, 0) \\
6 = (3, 1, 1) \\
7 = (3, 1, 0) \\
8 = (3, 0, 0)
\]

The group is a matrix group.

The base action is on projective space $\text{PG}(1, 8)$

$q = 8$
$p = 2$
$e = 3$
$n = 1$

Number of points $= 9$
Number of lines $= 1$
The finite field $\mathbb{F}_8$

polynomial: $X^3 + X^2 + 1 = 13$

$Z_i = \log_\alpha (1 + \alpha^i)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\gamma_i$</th>
<th>$\overline{-\gamma_i}$</th>
<th>$\overline{-\gamma_i}$</th>
<th>$\log_\alpha (\gamma_i)$</th>
<th>$\alpha^i$</th>
<th>$Z_i$</th>
<th>$\phi(\gamma_i)$</th>
<th>$T(\gamma_i)$</th>
<th>$N(\gamma_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 = 0</td>
<td>DNE</td>
<td>DNE</td>
<td>1</td>
<td>DNE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 = 1</td>
<td>1</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$\alpha = \gamma$</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$\alpha + 1 = \gamma^5$</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$\alpha^2 = \gamma^2$</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>$\alpha^2 + 1 = \gamma^3$</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>$\alpha^2 + \alpha = \gamma^6$</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>$\alpha^2 + \alpha + 1 = \gamma^4$</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>DNE</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ + \]

\[
\begin{array}{cccccccc}
+ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\
2 & 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\
3 & 3 & 2 & 1 & 0 & 7 & 6 & 5 & 4 \\
4 & 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\
5 & 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \\
6 & 6 & 7 & 4 & 5 & 2 & 3 & 0 & 1 \\
7 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

\[ \cdot \]

\[
\begin{array}{cccccccc}
\cdot & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
2 & 2 & 4 & 6 & 5 & 7 & 1 & 3 \\
3 & 3 & 6 & 5 & 1 & 2 & 7 & 4 \\
4 & 4 & 5 & 1 & 7 & 3 & 2 & 6 \\
5 & 5 & 7 & 2 & 3 & 6 & 4 & 1 \\
6 & 6 & 1 & 7 & 2 & 4 & 3 & 5 \\
7 & 7 & 3 & 4 & 6 & 1 & 5 & 2 \\
\end{array}
\]
$2^0 = 1$  \hspace{1cm}  $2^5 = 3$
$2^1 = 2$  \hspace{1cm}  $2^6 = 6$
$2^2 = 4$  \hspace{1cm}  $2^7 = 1$
$2^3 = 5$
$2^4 = 7$

**Base and Stabilizer Chain**

Group order 1512
$tl=9,8,7,3,$

**Stabilizer chain**

<table>
<thead>
<tr>
<th>Level</th>
<th>Base pt</th>
<th>Orbit length</th>
<th>Subgroup order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>9</td>
<td>1512</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>168</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Basic Orbit 0**

Basic orbit 0 has size 9
0, 1, 2, 3, 4, 5, 6, 7, 8

![Diagram of Basic Orbit 0]
Basic Orbit 1

1, 2, 3, 4, 5, 6, 7, 8

Basic orbit 1 has size 8

Basic Orbit 2

2, 3, 4, 5, 6, 7, 8

Basic orbit 2 has size 7
Basic Orbit 3

Basic orbit 3 has size 3
4, 6, 7
25 Group Theoretic Activities

Once a group has been created as in Section 21, a group theoretic activity can be performed. For this purpose, Orbiter provides the -\texttt{group_theoretic_activities} option. Tables 27 and 28 list the possible commands that can come after it. As usual, the -\texttt{group_theoretic_activities} command line block must be terminated with the -\texttt{end} option.

The command

\begin{verbatim}
orbiter.out -v 5 -define G -linear_group -PGL 3 4 -end -end \
    -with G -do \ 
    -group_theoretic_activities -find_singer_cycle -end
\end{verbatim}

finds all Singer cycles in PGL(3, 4) whose matrix is the companion matrix of a polynomial. The first one found is

\[
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
2 & 3 & 2
\end{bmatrix}
\]
whose projective order is 21. Here, we are using the numeric form of field elements, so 2 is $\omega$ and 3 is $\omega + 1$.

Suppose we want to multiply two elements in a group. The following command shows an example in GL(2, 8). We multiply the elements coded by 0,1,2,3 and 4,5,6,7:

\begin{verbatim}
orbiter.out -v 5 -define G -linear_group -GL 2 8 -end -end \
    -with G -do \ 
    -group_theoretic_activities -multiply "0,1,2,3" "4,5,6,7" 
pdflatex GL_2_8_mult.tex
\end{verbatim}

The output is

\[
\begin{bmatrix}
0 & 1 \\
\gamma & \gamma^5
\end{bmatrix} \begin{bmatrix}
\gamma^2 & \gamma^3 \\
\gamma^6 & \gamma^4
\end{bmatrix} = \begin{bmatrix}
\gamma^6 & \gamma^4 \\
\gamma & \gamma^5
\end{bmatrix}
\]

0,1,2,3,
4,5,6,7,
6,7,2,3,
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-multiply</td>
<td>$s_1 \ s_2$</td>
<td>Multiplies group elements $s_1$ and $s_2$, assuming the elements are given in coded form. Produces a latex report.</td>
</tr>
<tr>
<td>-inverse</td>
<td>$s$</td>
<td>Computes the inverse of $s$, which is given in coded form. Produces a latex report.</td>
</tr>
<tr>
<td>-raise_to_the_power</td>
<td>$s \ n$</td>
<td>Computes the $n$-th power of of $s$, which is given in coded form. Produces a latex report.</td>
</tr>
<tr>
<td>-search_element_of_order</td>
<td>$i$</td>
<td>Finds all elements of order $i$ in the group $(i \in \mathbb{N})$.</td>
</tr>
<tr>
<td>-element_rank</td>
<td>$s$</td>
<td>Determines the rank of the group element $s$ in the given group. $s$ is given in coded form.</td>
</tr>
<tr>
<td>-element_unrank</td>
<td>$r$</td>
<td>Produces the group element whose rank is $r$.</td>
</tr>
<tr>
<td>-find_singer_cycle</td>
<td></td>
<td>Finds all Singer cycles whose matrix is a companion matrix.</td>
</tr>
<tr>
<td>-export_gap</td>
<td></td>
<td>Exports the group to GAP [20].</td>
</tr>
<tr>
<td>-export_magma</td>
<td></td>
<td>Exports the group to Magma [12].</td>
</tr>
<tr>
<td>-poset_classification_control</td>
<td>see Table 31</td>
<td>Poset classification options. The argument list must be terminated with -end</td>
</tr>
<tr>
<td>-classes_based_on_normal_form</td>
<td></td>
<td>Stores the group table as csv-file.</td>
</tr>
<tr>
<td>-group_table</td>
<td></td>
<td>Produce a latex report about the group.</td>
</tr>
<tr>
<td>-report</td>
<td></td>
<td>Include Sylow subgroups in the report (requires -report).</td>
</tr>
<tr>
<td>-sylow</td>
<td></td>
<td>Produces a printout of all group elements.</td>
</tr>
<tr>
<td>-print_elements</td>
<td></td>
<td>Produces a latex report of all group elements.</td>
</tr>
<tr>
<td>-order_of_products</td>
<td>$g_1 \ldots g_n$</td>
<td>Creates a table of the orders of all products $g_i g_j$, $1 \leq i, j \leq n$.</td>
</tr>
</tbody>
</table>

Table 27: Group theoretic activities (Part 1)
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-classify_arcs</td>
<td>description</td>
<td>Classify arcs in geometries. See Section 34.</td>
</tr>
<tr>
<td>-linear_codes</td>
<td>$d \ n_{\text{max}}$</td>
<td>Classify linear codes with prescribed minimum distance $d$. Assumes that the group is $\text{PGL}(r,q)$ or $\text{P}^\Gamma\text{L}(r,q)$. For each $n \leq n_{\text{max}}$, the $[n,k,\geq d]$ codes are classified with $n-k=r$. See Section 40.</td>
</tr>
<tr>
<td>-spread_classify</td>
<td>$k$</td>
<td>Classifies spreads of $\text{PG}(k-1,q)$ in $\text{PG}(n-1,q)$. The group must be $\text{P}^\Gamma\text{L}(n,q)$ or any subgroup. See Section 45.</td>
</tr>
<tr>
<td>-spread_table_init</td>
<td>$k \ L \ \text{path}$</td>
<td>Creates a table of all spreads in $\text{PG}(3,q)$ whose isomorphism type belongs to the list of orbiter isomorphism types $L$. The spreads are stored in files with prefix $\text{path}$. See Section 46.</td>
</tr>
<tr>
<td>-packing_with_</td>
<td>description</td>
<td>Classifies packings in $\text{PG}(3,q)$ consisting of spreads whose isomorphism type belongs to the given list. A group of symmetries $H$ is assumed. The normalizer $N$ of $H$ is used to classify the packings. See Section 46.</td>
</tr>
<tr>
<td>assumed_symmetry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-tensor_classify</td>
<td>$d$</td>
<td>Classifies tensors of tensor-rank at most $d$.</td>
</tr>
<tr>
<td>-tensor_permutations</td>
<td></td>
<td>Computes the permutation representation of generators of wreath product.</td>
</tr>
<tr>
<td>-reverse_iso</td>
<td></td>
<td>Given a set of generators of a subgroup of $\text{PGO}^+(6,q)$ as $6 \times 6$ matrixes, compute the inverse image of the generators in $\text{PGL}(4,q)$ (if possible).</td>
</tr>
<tr>
<td>morphism_exterior_square</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-classify_cubic_curves</td>
<td>descr</td>
<td>Classifies cubic curves. Expects an arc description options as in Table 39.</td>
</tr>
</tbody>
</table>

Table 28: Group theoretic activities (Part 2)
Note that the output shows the codings of the three group elements. This way, the result of this computation can be processed further easily. The same example over $\mathbb{F}_7$, noting that $7 \equiv 0 \mod 7$ is:

\[
\text{orbiter.out -v 5 -define G -linear_group -GL 2 7 -end -end \-with G -do \-group_theoretic_activities -multiply "0,1,2,3" "4,5,6,0"}
\]
\[
pdflatex GL_2_7_mult.tex
\]
The output is

\[
\begin{bmatrix}
0 & 1 \\
2 & 3
\end{bmatrix}
\begin{bmatrix}
4 & 5 \\
6 & 0
\end{bmatrix}
= 
\begin{bmatrix}
6 & 0 \\
5 & 3
\end{bmatrix}
\]

0,1,2,3,
4,5,6,0,
6,0,5,3,

We can compute the inverse of a group element:

\[
\text{orbiter.out -v 5 -define G -linear_group -GL 2 7 -end -end \-with G -do \-group_theoretic_activities -inverse "0,1,2,3"}
\]
\[
pdflatex GL_2_7_inv.tex
\]
The output is

\[
\begin{bmatrix}
0 & 1 \\
2 & 3
\end{bmatrix}
^{-1}
= 
\begin{bmatrix}
2 & 4 \\
1 & 0
\end{bmatrix}
\]

0,1,2,3,
2,4,1,0,

We can raise a group element to a power:

\[
\text{orbiter.out -v 5 -define G -linear_group -GL 2 7 -end -end \-with G -do \-group_theoretic_activities -raise_to_the_power "0,1,2,3" 2}
\]
\[
pdflatex GL_2_7_inv.tex
\]
The output is
\[
\begin{bmatrix}
0 & 1 \\
2 & 3
\end{bmatrix}^2 =
\begin{bmatrix}
2 & 3 \\
6 & 4
\end{bmatrix}
\]
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-classes</td>
<td></td>
<td>Compute a report of the conjugacy classes of elements.</td>
</tr>
<tr>
<td>-centralizer_of_element</td>
<td>label coding</td>
<td>Compute the centralizer of the coded group element, using label to create file names.</td>
</tr>
<tr>
<td>-normalizer_of_cyclic_subgroup</td>
<td>label s</td>
<td>Compute the normalizer of the cyclic subgroup generated by the element s.</td>
</tr>
<tr>
<td>-normalizer</td>
<td></td>
<td>Compute the normalizer of a subgroup in the larger group.</td>
</tr>
</tbody>
</table>

Table 29: Group theoretic activities based on Magma

26 Group Theoretic Activities Based on Magma

Through its interface to Magma [12], Orbiter can perform group theoretic computations. Table 29 list the group theoretic commands that rely on Magma. The communication to and from magma happens through files. This is a three step process: An Orbiter session receives a command to compute the conjugacy classes of a group. The Orbiter session writes a magma file. This file is read and executed by Magma. Magma writes a second file containing the conjugacy classes in coded form. Another Orbiter session reads the magma output file, decodes the information and produces the desired list of conjugacy classes. A latex report is written containing the classes, as well as related information regarding centralizers and normalizers.

For instance, the three-step command sequence

```plaintext
orbiter.out -v 6 \
  -define G \
  -linear_group -PGGL 2 4 -end -end \
  -with G -do \ 
  -group_theoretic_activities \ 
  -classes -end \
/usr/local/magma/magma PGGL_2_4_classes.magma\norbiter.out -v 6 \
  -define G \
  -linear_group -PGGL 2 4 -end -end \
  -with G -do \ 
  -group_theoretic_activities \ 
  -classes -end \npdflatex PGGL_2_4_classes_out.tex
```

computes the classes of elements in PΓL(2,4) using Orbiter-Magma-Orbiter. The first Orbiter command produces the file `PGGL_2_4_classes.magma`. The magma command
reads this file and produces the file PGGL_2_4_classes_out.txt. The second Orbiter command reads the file PGGL_2_4_classes_out.txt and produces the latex report PGGL_2_4_classes_out.tex.

The report produced by Orbiter is quite long (4 pages). Let us look at just one conjugacy class. Here is the output for class 1 / 7 (numbering starts from 0, so this is the second class):

Order of element = 2  
Class size = 10  
Centralizer order = 12  
Normalizer order = 12  
Representing element is  

\[
c_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}_1
\]

do order 2 and with 3 fixed points. 0, 1, 1, 0, 1,
The normalizer is generated by:
Strong generators for a group of order 12:
\[
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_1, \begin{bmatrix} \omega^2 & 0 \\ 0 & 1 \end{bmatrix}_1, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}_1
\]

1,0,0,1,1,  
1,0,0,2,1,  
0,1,1,0,1,

The command sequence

```bash
orbiter.out -v 6 \ 
-define G \ 
-linear_group -PGGL 4 4 -end -end \ 
-with G -do \ 
-group_theoretic_activities \ 
-centralizer_of_element "2A" "1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1, 1" \ 
-end
/usr/local/magma/magma element_2A_centralizer.magma
orbiter.out -v 6 \ 
-define G \ 
-linear_group -PGGL 4 4 -end -end \ 
-with G -do \ 
-group_theoretic_activities \ 
-centralizer_of_element "2A" "1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1, 1" \ 
-end
```

106
computes the centralizer of the Baer involution

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

The centralizer is a group of order 40320, isomorphic to \( \text{PGL}(4, 2).\mathbb{Z}_2 \). Orbiter produces a list of strong generators, shown below:

Strong generators for a group of order 40320:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

The end of the report has a list of generators in coded form. This list can be used to create the centralizer in Orbiter. The following command illustrates this:

```
orbiter.out -v 6 \
-draw_options -rad 200 -end 
```
Orbiter can compute the normalizer of a cyclic subgroup. For instance, the element

$$\sigma = \begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 3 & 0 & 4
\end{bmatrix}$$

generates a cyclic subgroup of PGL(4,5) of order 31. The command

```
orbiter.out -v 6 -define G \
    -linear_group -PGL 4 5 -end -end \
    -with G -do \
    -group_theoretic_activities \
    -normalizer_of_cyclic_subgroup "31" "2,0,0,0, 0,0,1,0, 0,0,0,1, 0,3,0,4"
pdflatex normalizer_of_31_in_PGL_4_5.tex
```

computes the normalizer, which is a group of order 372. The following report is produced:

<table>
<thead>
<tr>
<th>The subgroup generated by</th>
</tr>
</thead>
</table>
| \[ \begin{bmatrix}
3 & 0 & 0 & 0 \\
0 & 4 & 0 \\
0 & 0 & 4 \\
2 & 0 & 1
\end{bmatrix} \] |

has order 31

The normalizer has order 372
Strong generators for a group of order 372:

\[
\begin{bmatrix}
4 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & 4 \\
0 & 2 & 3 & 1
\end{bmatrix},
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 0 & 0 & 2 \\
0 & 2 & 2 & 1
\end{bmatrix}
\]

1,0,0,0,0,4,0,0,0,0,4,0,0,0,0,4,
1,0,0,0,0,3,0,0,0,0,3,0,0,0,0,3,
1,0,0,0,0,4,0,0,0,0,2,1,0,3,2,4,
1,0,0,0,0,0,1,0,0,0,0,1,0,1,1,3,

For general normalizers, the group must be constructed as a subgroup $H$ of a bigger group $G$. Consider this example. The group

\[H = \langle \begin{bmatrix}
\alpha^4 & 0 \\
0 & 1
\end{bmatrix}, \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix} \rangle \simeq C_2 \times C_2\]

is a subgroup of $G = \text{PGL}(2, 9)$. To compute the normalizer of $H$ in $G$, the following command sequence can be used:

```
orbiter.out -v 2 \
  -define G -linear_group -PGL 2 9 -subgroup_by_generators \
    Z22 4 2 "2,0,0,1" "0,1,1,0" -end -end \
  -with G -do \
  -group_theoretic_activities \
  -normalizer \
  -end
```

pdflatex PGL_2_9_Subgroup_Z22_4_normalizer.tex

It produces a report showing the the normalizer is a group of order 24 (it is isomorphic to $\text{Sym}(4)$, though the report does not tell us this fact directly):
Inside the group of order 720, the normalizer has order 24:

Strong generators for a group of order 24:

\[
\begin{pmatrix}
\alpha^4 & 0 \\
0 & 1
\end{pmatrix}, \begin{pmatrix}
\alpha^2 & 0 \\
0 & 1
\end{pmatrix}, \begin{pmatrix}
\alpha^4 & \alpha^4 \\
\alpha^4 & 1
\end{pmatrix},
\begin{pmatrix}
\alpha^4 & \alpha^6 \\
\alpha^2 & 1
\end{pmatrix}
\]
Figure 13: The Schreier tree for the action on rank-one tensors

27 Orbit Algorithms

One of the main applications of Orbiter is computing orbits of various groups in various actions (after all, this is why it is named Orbiter). Orbiter provides a range of different orbit algorithms. A basic algorithm uses Schreier trees. It is explained in [14, 26, 50]. This Schreier algorithm has memory complexity proportional to the size of the orbit. This means that the algorithm is somewhat limited to small problems and does not scale well. For more fancy algorithms using posets, see Section 28.

All of the commands discussed in this section are group theoretic activities, so they must be part of a -group_theoretic_activities sequence.

The basic Schreier algorithm can be used as the default orbit algorithm for all problems. Consider the wreath product acting on rank-one tensors from Section 24. The following command sequence computes the orbits, exports the Schreier tree, and produces the drawing shown in Figure 13.

orbiter.out -v 4 \ 
-draw_options -end \ 
-define G \ 
-linear_group -GL_d_q_wr_Sym_n 2 2 3 -on_rank_one_tensors -end -end \
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-orbits_on_subsets</td>
<td>$k$</td>
<td>Compute orbits on $k$-subsets.</td>
</tr>
<tr>
<td>-orbits_on_points</td>
<td></td>
<td>Compute orbits in the action that was created.</td>
</tr>
<tr>
<td>-orbits_of</td>
<td>$i$</td>
<td>Compute orbit of point $i$ in the action that was created.</td>
</tr>
<tr>
<td>-stabilizer</td>
<td></td>
<td>Compute the stabilizer of the orbit representative (needs -orbits_on_points).</td>
</tr>
<tr>
<td>-orbits_on_set_system_from_file</td>
<td>fname $f$ $l$</td>
<td>Reads the csv file “fname” and extract sets from columns $[f,...,f+l-1]$.</td>
</tr>
<tr>
<td>-orbit_of_set_from_file</td>
<td>fname</td>
<td>Reads a set from the text file “fname” and computes orbits on the elements of the set.</td>
</tr>
<tr>
<td>-orbits_on_polynomials</td>
<td>$d$</td>
<td>Computes the orbits of the matrix group on homogeneous polynomials of degree $d$. The number of variables is determined by the degree of the matrix group.</td>
</tr>
<tr>
<td>-conjugacy_class_of</td>
<td>label $s$</td>
<td>Compute the conjugacy class of the group element encoded as $s$ using the given label for file-names. Write a file containing the ranks for all elements in the class. Writes a second file containing the transporter elements for each element in the class. A transporter element maps the class representative to the given element under conjugation.</td>
</tr>
<tr>
<td>-orbits_on_group_elements_under_conjugation</td>
<td>fname-C fname-T</td>
<td>Under the centralizer of the class representative, construct the orbits on the class. For each non-trivial orbit, test whether the group generated by it and the class representative is Klein-four and all nontrivial elements are from the given class. If so, classify these groups and compute the normalizers. The arguments fname-C and fname-T are the files containing the elements of the class and the transporter, respectively.</td>
</tr>
</tbody>
</table>

Table 30: Basic Orbit algorithms
In the next example, we compute the orbits of a linear group on homogeneous polynomials:

```
orbiter.out -v 4 \
  -define G \
  -linear_group -PGL 4 2 \
  -end -end \
  -with G -do \
  -group_theoretic_activities \
  -orbits_on_polynomials 3 \
  -end
```

```
pdflatex poly_orbits_d3_n3_q2.tex
```

This command computes the orbits of PGL(4, 2) on all cubic forms in 4 variables, confirming the work of Dickson [17] and an enumerative result of Cooley [16].

The next example computes orbits for an induced action. Recall from Section 24 that one group can have many actions. In particular, Orbiter can work with induced actions without changing the representation of the group elements. This has the advantage that the stabilizers are expressed in terms of the original action. To consider an example, suppose we want to consider the action of the stabilizer of a conic on the points of the plane (this continues an example from Section 24). The following command can be used:

```
orbiter.out -v 4 \
  -draw_options -rad 250 -end \
  -define G \
  -linear_group -PGGL 2 8 -PGL2OnConic -end -end \
  -with G -do \
  -group_theoretic_activities \
  -orbits_on_points\n  -report \
  -end
```

The output is another Orbiter report. First, the orbits are listed. Then for each orbit, the stabilizer is shown, together with the generators in the action on the plane. For the sake of saving space, some of the output has been shortened. The three orbits correspond to the conic, the nucleus and the remaining points of the plane.
**Group Orbits**

Orbits of the group $\text{PGL}(2,8)\text{OnConic}$:

Strong generators for a group of order 1512:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}_1, \begin{bmatrix}
\gamma & 0 \\
0 & 1
\end{bmatrix}_0, \begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix}_0,
\]

\[
\begin{bmatrix}
1 & 0 \\
\gamma & 1
\end{bmatrix}_0, \begin{bmatrix}
1 & 0 \\
\gamma^2 & 1
\end{bmatrix}_0, \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}_0.
\]

1,0,0,1,1,
1,0,0,6,0,
1,0,1,1,0,
1,0,2,1,0,
1,0,4,1,0,
0,1,1,0,0,

Considering the orbit length, there are 3 types of orbits:

\[(1, 9, 63)\]

\[i : \text{orbit length} : \text{number of orbits}\]

\[0 : 1 : 1\]

\[1 : 9 : 1\]

\[2 : 63 : 1\]

Orbits classified:

Set 0 has size 1 : \{1\}
Set 1 has size 1 : \{0\}
Set 2 has size 1 : \{2\}

Orbits of length 1:

Orbit 1: (1)

\[0 : 1 = (0, 1, 0)\]

Orbits of length 9:

Orbit 0: (0, 2, 3, 29, 48, 38, 55, 60, 67)
Orbits of length 63:
Orbit 2: (4, 5, 18, 7, 57, 25, 11, 37, 56, 10, 8, 33, 66, 45, 32, 41, 34, 14, 64, 9, 30, 47, 68, 52, 59, 71, 62, 6, 49, 65, 26, 21, 72, 54, 39, 13, 20, 43, 70, 50, 61, 17, 22, 44, 35, 23, 46, 40, 51, 28, ...12, 31, 16)

Orbits of length 1:
Orbit 1: (1)
Stabilizer of orbit representative 1:

Strong generators for a group of order 1512:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}, \begin{bmatrix}
\gamma & 0 \\
0 & 1
\end{bmatrix}, \begin{bmatrix}
1 & 0 \\
\gamma & 1
\end{bmatrix},
\]
\[
\begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\]

1,0,0,1,1,
1,0,0,6,0,
1,0,2,1,0,
0,1,1,0,0,
Generator 0 / 4 is:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]
Generator 1 / 4 is:
\[
\begin{bmatrix}
\gamma & 0 \\
0 & 1
\end{bmatrix}
\]

Generator 2 / 4 is:
\[
\begin{bmatrix}
1 & 0 \\
\gamma & 1
\end{bmatrix}
\]

Generator 3 / 4 is:
\[
\begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\]

Orbits of length 9:
Orbit 0: ( 0, 2, 3, 29, 48, 38, 55, 60, 67 )
Stabilizer of orbit representative 0:
Strong generators for a group of order 168:
\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}, \begin{bmatrix}
\gamma^6 & 0 \\
0 & 1
\end{bmatrix}, \begin{bmatrix}
\gamma^2 & 0 \\
0 & 1
\end{bmatrix}
\]
1,0,0,1,1,
1,0,0,2,0,
1,0,3,5,0,
Generator 0 / 3 is:
\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]
Generator 1 / 3 is:

\[
\begin{bmatrix}
\gamma^6 & 0 \\
0 & 1
\end{bmatrix}
\]


Generator 2 / 3 is:

\[
\begin{bmatrix}
\gamma^4 & 0 \\
\gamma^2 & 1
\end{bmatrix}
\]


Orbits of length 63:

Orbit 2: \( \langle 4, 5, 18, 7, 57, 25, 11, 37, 56, 10, 8, 33, 66, 45, 32, 41, 34, 14, 64, 9, 30, 47, 68, 52, 59, 71, 62, 6, 49, 65, 26, 21, 72, 54, 39, 13, 20, 43, 70, 50, 61, 17, 22, 44, 35, 23, 46, 51, 28, ...12, 31, 16 \rangle \)

Stabilizer of orbit representative 4:

Strong generators for a group of order 24:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}_1, \begin{bmatrix}
1 & 0 \\
\gamma^5 & 1
\end{bmatrix}_2, \begin{bmatrix}
1 & 0 \\
\gamma^3 & 1
\end{bmatrix}_0,
\begin{bmatrix}
1 & 0 \\
\gamma & 1
\end{bmatrix}_0
\]

1,0,0,1,1,
1,0,3,1,2,
1,0,5,1,0,
1,0,2,1,0,

Generator 0 / 4 is:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}_1
\]

Generator 1 / 4 is:

\[
\begin{bmatrix}
1 & 0 \\
\gamma^5 & 1
\end{bmatrix}
\]


Generator 2 / 4 is:

\[
\begin{bmatrix}
1 & 0 \\
\gamma^3 & 1
\end{bmatrix}
\]


Generator 3 / 4 is:

\[
\begin{bmatrix}
1 & 0 \\
\gamma & 1
\end{bmatrix}
\]

28 Poset Classification

A poset is a type of order structure as it arises frequently in Algebra and Combinatorics. For instance, the set of subsets of a set form an order structure with respect to the set-inclusion relation. The Hasse diagram is a diagram whose nodes represent the element. Nodes are arranged from top to bottom, and relations are indicated by lines. Transitivity is implied. For instance, Figure 14 shows the power set lattice of a four-element subset. For simplicity, the subsets are listed in a somewhat compressed form. Sets are related using the subset relation if there is a path between the associated nodes (the path is not allowed to change the direction, i.e. no up-down paths).

In algebraic combinatorics, posets often come with group actions on them. A group $G$ acts on a poset $\mathcal{P}$ if for all $x, y \in \mathcal{P}$ and all $g \in G$,

$$x \leq y \Rightarrow xg \leq yg.$$

For background on poset actions, see Plesken [44]. The orbits of $G$ on $\mathcal{P}$ form another poset, the poset of orbits. Many problem in Combinatorics reduce to poset classification problems. Orbiter can compute the orbits of groups on posets. The algorithm is based on work of Schmalz [49]. Two posets can be a subposet of either the subset lattice of some set or the subspace lattice of some vector space over a finite field. Figure 15 shows the subspace lattice of $V(3,2) = \mathbb{F}_2^3$. The basis elements are listed, using the enumerator for elements on the projective geometry $\text{PG}(2,2)$ explained in Section 13.

The commands shown in Tables 31-32 can be used to control the poset classification algorithm.
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-problem_label</td>
<td>str</td>
<td>Use str as a prefix for files that are created.</td>
</tr>
<tr>
<td>-path</td>
<td>p</td>
<td>Use path p for files that are created.</td>
</tr>
<tr>
<td>-depth</td>
<td>d</td>
<td>Set search depth to d.</td>
</tr>
<tr>
<td>-draw_options</td>
<td>options</td>
<td>Drawing options according to Table 63.</td>
</tr>
<tr>
<td>-v</td>
<td>v</td>
<td>Set verbosity to v. Larger numbers mean more output.</td>
</tr>
<tr>
<td>-gv</td>
<td>v</td>
<td>Set verbosity for group theoretic operations to v. Larger numbers mean more output.</td>
</tr>
<tr>
<td>-recover</td>
<td>fname</td>
<td>Recover from the given file.</td>
</tr>
<tr>
<td>-lex</td>
<td></td>
<td>Use the lexicographic ordering to speed up the search.</td>
</tr>
<tr>
<td>-w</td>
<td></td>
<td>Save orbits at level d only.</td>
</tr>
<tr>
<td>-W</td>
<td></td>
<td>Save orbits at all levels.</td>
</tr>
<tr>
<td>-write_data_files</td>
<td></td>
<td>Save data to files.</td>
</tr>
<tr>
<td>-t</td>
<td></td>
<td>Write a file containing the search tree at level d.</td>
</tr>
<tr>
<td>-T</td>
<td></td>
<td>Write a file containing the search tree at all levels.</td>
</tr>
<tr>
<td>-write_tree</td>
<td></td>
<td>Write the poset of orbits as a tree file.</td>
</tr>
<tr>
<td>-find_node_by_stabilizer_order</td>
<td>i</td>
<td>Find all nodes whose stabilizer has order i.</td>
</tr>
<tr>
<td>-draw_poset</td>
<td></td>
<td>Produce a drawing of the poset of orbits.</td>
</tr>
<tr>
<td>-draw_full_poset</td>
<td></td>
<td>Produce a drawing of the full poset with elements grouped by orbits.</td>
</tr>
<tr>
<td>-plesken</td>
<td></td>
<td>Compute Plesken matrices Asup and Ainf.</td>
</tr>
</tbody>
</table>

Table 31: Options to control the poset classification algorithm (Part 1)
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-print_data_structure</td>
<td></td>
<td>Print the data structure.</td>
</tr>
<tr>
<td>-list</td>
<td></td>
<td>List orbits at level ( d ).</td>
</tr>
<tr>
<td>-list_all</td>
<td></td>
<td>List orbits at all levels.</td>
</tr>
<tr>
<td>-table_of_nodes</td>
<td></td>
<td>Produce a spreadsheet of all orbits.</td>
</tr>
<tr>
<td>-make_relations_with_flag_orbits</td>
<td></td>
<td>Produce a bitmap drawing of the neighboring relations in the poset with flag orbits.</td>
</tr>
<tr>
<td>-Kramer_Mesner_matrix</td>
<td>( t \ k )</td>
<td>Compute the Kramer-Mesner matrix ( M_{t,k} ).</td>
</tr>
<tr>
<td>-level_summary_csv</td>
<td></td>
<td>Write a summary of number of orbits at each level to a csv file.</td>
</tr>
<tr>
<td>-orbit_reps_csv</td>
<td></td>
<td>Write orbit representatives to a csv file.</td>
</tr>
<tr>
<td>-report</td>
<td></td>
<td>Produce a latex report. Requires -orbiter_path option from Section 4.</td>
</tr>
<tr>
<td>-node_label_is_group_order</td>
<td></td>
<td>When drawing the poset of orbits, display the group order in the orbit nodes.</td>
</tr>
<tr>
<td>-node_label_is_element</td>
<td></td>
<td>When drawing the poset of orbits, display the element rank in the orbit nodes.</td>
</tr>
<tr>
<td>-show_orbit_decomposition</td>
<td></td>
<td>Show the orbits of the stabilizers.</td>
</tr>
<tr>
<td>-show_stab</td>
<td></td>
<td>Show the stabilizers.</td>
</tr>
<tr>
<td>-save_stab</td>
<td></td>
<td>Save the stabilizer generators.</td>
</tr>
<tr>
<td>-show_whole_orbits</td>
<td></td>
<td>Show the whole orbits.</td>
</tr>
<tr>
<td>-recognize</td>
<td>( L )</td>
<td>Recognize the given object in the classified list and compute a transporter that maps the given object to the canonical form. Here, ( L ) must be a list of integers (comma separated and enclosed in double quotes) encoding an object. This option can be repeated.</td>
</tr>
<tr>
<td>-export_schreier_trees</td>
<td></td>
<td>Export all Schreier trees.</td>
</tr>
<tr>
<td>-draw_schreier_trees</td>
<td>( \text{args} )</td>
<td>Draw all Schreier trees.</td>
</tr>
</tbody>
</table>

Table 32: Options to control the poset classification algorithm (Part 2)
Figure 15: Subspace lattice of $V(3, 2)$
29 Orbits on Subsets

The lattice of subsets of a set $X$ is $\mathcal{P}(X)$, the set of all subsets of $X$, ordered with respect to inclusion. Assume that a group $G$ acts on $X$, and hence on the lattice by means of the induced action on subsets. The orbits of $G$ on subsets form a new poset, the poset of orbits. Poset classification is the process of computing the poset of orbits. Orbiter has an algorithm to perform poset classification. In many cases, we are not interested in the full lattice of subsets $\mathcal{P}(X)$ but rather in a subposet of it. We require that the subposet is closed under the group action and that the following property holds:

$$x, y \in \mathcal{P}(X) \text{ and } x \leq y \Rightarrow (y \in \mathcal{P} \rightarrow x \in \mathcal{P}).$$

The join of two subsets in the poset may or may not belong to the poset. Let us consider the poset of subsets of the 4-element set under the action of a group of order 3. We take the 4 points to be the vectors of $X = \mathbb{F}_2^2$. Let $G$ be the group generated by the Singer cycle in $\text{GL}(2,2)$, so

$$G = \langle \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \rangle \simeq \langle (0)(1,3,2) \rangle,$$

the latter being the permutation representation on the set $X$. Thus, $G$ is a group of order 3 acting with one fixed point. The command

```
orbiter.out -v 3 -define G \
  -linear_group -GL 2 2 -singer 1 -end \n  -with G -do \n  -group_theoretic_activities \n  -orbits_on_subsets 4 \n  -draw_poset \n  -report \n  -end
```

computes the orbits of $G$ on the poset of subsets. The poset of orbits is shown in Figure 16. All nodes except for the root node are labeled by elements of $X = \{0,1,2,3\}$. In order to determine the set that is associated to a node, we follow the unique leftmost path to the root node and collect the node labels. This produces the set associated to the node. The orbit representatives are indicated next to the diagram. By convention, the order of the stabilizer is written as a subscript. Since this example is small enough, it is possible to show the complete orbits, as in Figure 17. Elements belonging to an orbit are grouped together. The leftmost element in each group is the orbit representative.

The lex-least spanning tree of a poset is obtained by joining each non-root node to its unique lex-least ancestor. Figure 18 shows the spanning tree for the poset of subsets of a 4 element set.
Figure 16: The poset of orbits under the Singer group

\{0\}_3
\{0,1\}_1
\{1\}_1
\{0,1,2\}_1
\{1,2,3\}_3
\{0,1,2,3\}_3

Figure 17: The poset with orbits indicated by grouping
Figure 18: The lex-least spanning tree for the poset of orbits
30 Orbits on Subspaces

Orbiter can compute the orbits of a group on the lattice of subspaces of a finite vector space.

The orthogonal group is the stabilizer of a non-degenerate quadric. Suppose we want to classify the subspaces in $\text{PG}(3,2)$ under the action of the orthogonal group. In $\text{PG}(3,2)$ there are two distinct nondegenerate quadrics, $Q^+(3,2)$ and $Q^-(3,2)$. The $Q^+(3,2)$ 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 19. $\text{PG}(3,2)$ has 15 points:

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

The $Q^+(3,2)$ quadric given by the equation above consists of the nine points

$$P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_{10}.$$

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

```
orbiter.out -v 5 \n-draw_options -end \n-define G -linear_group -PGL 4 2 -orthogonal 1 -end -end \n-with G -do \n-group_theoretic_activities \n```

is executed.
Figure 20: Hasse-diagram for the orbits of the orthogonal group $\text{PGO}^+(4, 2)$ on subspaces of $\text{PG}(3, 2)$

```
-\texttt{poset\_classification\_control} \ -\texttt{node\_label\_is\_element} \\
-\texttt{draw\_poset} \ -\texttt{draw\_options} \ -\texttt{end} \ -\texttt{problem\_label} O\texttt{p}_4_2 \ -W \ -\texttt{depth} 4 \ -\texttt{end} \\
-\texttt{orbits\_on\_subspaces} 4 \ -\texttt{end} \\
-\texttt{report} \ -\texttt{end} \\
\texttt{orbiter\_out} \ -v 5 \ -\texttt{end} \\
-\texttt{draw\_layered\_graph} \ \texttt{PGL}_4_2_\texttt{Orthogonal\_plus\_4_2\_poset\_lvl\_4\_layered\_graph} \ -\texttt{rad} 300 \ -\texttt{embedded} \ -\texttt{line\_width} 1.1 \ -\texttt{y\_stretch} 0.9 \ -\texttt{scale} 0.25 \ -\texttt{end} \\
pdflatex \texttt{PGL}_4_2_\texttt{Orthogonal\_plus\_4_2\_poset\_lvl\_4\_draw.tex}
```

produces a classification of all subspaces of $\text{PG}(3, 2)$ under $\text{PGO}^+(4, 2)$. The option `-\texttt{draw\_poset}` (inside `-\texttt{poset\_classification\_control}`) creates a Hasse diagram of the classification as shown in Figure 20. The nodes show the ranks of points in $\text{PG}(3, 2)$ as described in Section 13.
31 Creating Objects in Projective Geometries

Orbiter can create and process objects in projective space. In this section, we will focus on creating objects. In Section 32, we will discuss activities for objects in projective spaces.

Objects are represented as a set of integers and are frequently stored in files. The integers encode the elements of the object. For instance, an elliptic curve is represented as the list of ranks of all $\mathbb{F}_q$-rational points on it. A spread is represented by the list of ranks of the subspaces that define it. The `-create_combinatorial_object` command must be used before creating objects. After that, the commands shown in Tables 33 and 34 can be applied. Modifier options as shown in Table 35 can be given. For instance, the command sequence

```
orbiter.out -v 5 -create_combinatorial_object -q 11 \
    -elliptic_curve 1 3 -end -save "./"
```

creates the elliptic curve

$$y^2 \equiv x^3 + x + 3 \mod 11$$

over the field $\mathbb{F}_{11}$. The curve has 18 points, saved to the file `elliptic_curve_b1_c3_q11.txt`. 

<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-hyperoval</td>
<td></td>
<td>To create a hyperoval</td>
</tr>
<tr>
<td>-subiaco_oval</td>
<td>f_short</td>
<td>Create the Subiaco oval</td>
</tr>
<tr>
<td>-subiaco_hyperoval</td>
<td></td>
<td>Create the Subiaco hyperoval</td>
</tr>
<tr>
<td>-adelaide_hyperoval</td>
<td></td>
<td>Create the Adalaide hyperoval</td>
</tr>
<tr>
<td>-translation</td>
<td>i</td>
<td>Create the translation hyperoval with exponent i</td>
</tr>
<tr>
<td>-Segre</td>
<td></td>
<td>Create the Segre hyperoval</td>
</tr>
<tr>
<td>-Payne</td>
<td></td>
<td>Create the Payne hyperoval</td>
</tr>
<tr>
<td>-Cherowitzo</td>
<td></td>
<td>Create the Cherowitzo hyperoval</td>
</tr>
<tr>
<td>-OKeefe_Penttila</td>
<td></td>
<td>Create the O’Keefe, Penttila hyperoval</td>
</tr>
<tr>
<td>-BLT_database</td>
<td>k</td>
<td>Create the kth BLT-set of order q from the database (k = 0, 1,...)</td>
</tr>
<tr>
<td>-ovoid</td>
<td></td>
<td>Create an ovoid</td>
</tr>
<tr>
<td>-Baer</td>
<td></td>
<td>Create the (standard) Baer subgeometry</td>
</tr>
<tr>
<td>-orthogonal</td>
<td>ε</td>
<td>Create the <strong>Q</strong>^(ε)(n, q) quadric</td>
</tr>
<tr>
<td>-hermitian</td>
<td></td>
<td>Create the Hermitian variety given by (\sum_{i=0}^{n} X_i^{q+1} = 0)</td>
</tr>
<tr>
<td>-cuspidal_cubic</td>
<td></td>
<td>Create the cuspidal cubic ((s^3, ts^2, t^3)) in <strong>PG</strong>(2, q)</td>
</tr>
<tr>
<td>-twisted_cubic</td>
<td></td>
<td>Create a twisted cubic ((s^3, s^2 t, st^2, t^3)) in <strong>PG</strong>(3, q)</td>
</tr>
<tr>
<td>-elliptic_curve</td>
<td>a b</td>
<td>Create the elliptic curve (y^2 = x^3 + ax + b)</td>
</tr>
<tr>
<td>-ttp_construction_A</td>
<td></td>
<td>Create the twisted tensor product code of type A [6]</td>
</tr>
<tr>
<td>-ttp_construction_A_hyperoval</td>
<td></td>
<td>Create the twisted tensor product code of type A [6]</td>
</tr>
<tr>
<td>-ttp_construction_B</td>
<td></td>
<td>Create the twisted tensor product code of type B [6]</td>
</tr>
</tbody>
</table>

Table 33: Orbiter Objects (Part 1)
<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-unital_XXq_YZq_ZYq</td>
<td></td>
<td>Create the unital with equation $XX^q+YZ^q+ZY^q = 0$</td>
</tr>
<tr>
<td>-desarguesian_line_spread_in_PG_3_q</td>
<td></td>
<td>Create the desarguesian line spread in PG(3,q) as a set of 2-subspaces</td>
</tr>
<tr>
<td>-Buekenhout_Metz</td>
<td></td>
<td>Create the Buekenhout Metz unital</td>
</tr>
<tr>
<td>-Uab</td>
<td>$a\ b$</td>
<td>Create the Buekenhout Metz unital in the form of Barwick and Ebert [5]</td>
</tr>
<tr>
<td>-whole_space</td>
<td></td>
<td>Create the whole space</td>
</tr>
<tr>
<td>-hyperplane</td>
<td>pt</td>
<td>Create the hyperplane given by dual coordinates associated with the given point</td>
</tr>
<tr>
<td>-segre_variety</td>
<td>$a\ b$</td>
<td>Create the Segre variety</td>
</tr>
<tr>
<td>-Maruta_Hamada_arc</td>
<td></td>
<td>Create the Maruta Hamada arc</td>
</tr>
<tr>
<td>-projective_variety</td>
<td>$l\ d\ C$</td>
<td>Create the projective variety of degree $d$ with label $l$, with coefficient vector $C$, see Section 16.</td>
</tr>
<tr>
<td>-intersection_of_zariski_open_sets</td>
<td>$l\ d\ C_1\ldots\ C_n$</td>
<td>Create the intersection of the Zariski open sets given by equations $C_1,\ldots, C_n$ of degree $d$ with label $l$, see Section 16.</td>
</tr>
<tr>
<td>-projective_curve</td>
<td>$l\ r\ d\ C$</td>
<td>Create the projective curve of degree $d$ with label $l$, with coefficient vector $C$ in $r$ variables</td>
</tr>
</tbody>
</table>

Table 34: Orbiter Objects (Part 2)
<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>q</td>
<td>The size of the finite field $\mathbb{F}_q$</td>
</tr>
<tr>
<td>-Q</td>
<td>Q</td>
<td>The field size of the extension field $\mathbb{F}_Q$</td>
</tr>
<tr>
<td>-n</td>
<td>n</td>
<td>The projective dimension</td>
</tr>
<tr>
<td>-poly</td>
<td>r</td>
<td>Use polynomial with rank $r$ to create the field $\mathbb{F}_q$</td>
</tr>
<tr>
<td>-poly_Q</td>
<td>r</td>
<td>Use polynomial with rank $r$ to create the field $\mathbb{F}_Q$</td>
</tr>
<tr>
<td>-embedded_in_PG_4_q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-BLT_in_PG</td>
<td></td>
<td>Create the BLT-set with ranks in PG$(n, q)$ instead of orthogonal point ranks</td>
</tr>
<tr>
<td>-monomial_type_LEX</td>
<td></td>
<td>Monomials are in lexicographical ordering.</td>
</tr>
<tr>
<td>-monomial_type_PART</td>
<td></td>
<td>Monomials are in partition ordering.</td>
</tr>
</tbody>
</table>

Table 35: Orbiter Objects: Modifiers
<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fname_base_out</td>
<td>prefix</td>
<td>Use prefix for output files.</td>
</tr>
<tr>
<td>-ideal_LEX</td>
<td>$d$</td>
<td>Compute the ideal of a set of points, using lexicographic ordering of monomials</td>
</tr>
<tr>
<td>-ideal_PART</td>
<td>$d$</td>
<td>Compute the ideal of a set of points, using partition ordering of monomials</td>
</tr>
<tr>
<td>-homogeneous_polynomials_LEX</td>
<td>$d$</td>
<td>Prints the equation whose coefficient vector is the input vector using lexicographic ordering of monomials of degree $d$.</td>
</tr>
<tr>
<td>-homogeneous_polynomials_PART</td>
<td>$d$</td>
<td>Prints the equation whose coefficient vector is the input vector using partition ordering of monomials of degree $d$.</td>
</tr>
<tr>
<td>-canonical_form</td>
<td>prefix</td>
<td>Computes the canonical form of point sets, using prefix for the output file name. Requires -n option.</td>
</tr>
<tr>
<td>-n</td>
<td>$n$</td>
<td>Sets the projective dimension to $n$.</td>
</tr>
<tr>
<td>-draw_points_in_plane</td>
<td>prefix</td>
<td>Produces a drawing of a set of points in a projective plane. Uses the given prefix for the output file name.</td>
</tr>
<tr>
<td>-klein</td>
<td></td>
<td>Applies the Klein correspondence.</td>
</tr>
<tr>
<td>-line_type</td>
<td></td>
<td>Computes the line type.</td>
</tr>
<tr>
<td>-plane_type</td>
<td></td>
<td>Computes the plane type.</td>
</tr>
<tr>
<td>-conic_type</td>
<td></td>
<td>Computes the conic type.</td>
</tr>
<tr>
<td>-hyperplane_type</td>
<td></td>
<td>Computes the hyperplane type.</td>
</tr>
</tbody>
</table>

Table 36: Orbiter Activities for Objects in PG($n,q$) (Part 1)

32 Activities for Objects in Projective Geometries

Orbiter provides a range of activities for objects in projective geometries. Tables 36-37 lists the available commands. The objects must come from an input stream.
<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-intersect_with_set_from_file</td>
<td>fname</td>
<td>Computes the intersection with a set specified in the given file.</td>
</tr>
<tr>
<td>-arc_with_given_set_as_s_lines_after_dualizing</td>
<td>sz d d_{\min} s</td>
<td>Finds arcs with the given set as s-lines.</td>
</tr>
<tr>
<td>-arc_with_two_given_sets_of_lines_after_dualizing</td>
<td>sz d d_{\min} s t T</td>
<td>Finds arcs with the two given sets as s-lines and t-lines, respectively.</td>
</tr>
<tr>
<td>-arc_with_three_given_sets_of_lines_after_dualizing</td>
<td>sz d d_{\min} s t T u U</td>
<td>Finds arcs with the three given sets as s-lines and t-lines and u-lines, respectively.</td>
</tr>
<tr>
<td>-dualize_hyperplanes_to_points</td>
<td></td>
<td>Turns ranks of hyperplanes into ranks of points.</td>
</tr>
<tr>
<td>-dualize_points_to_hyperplanes</td>
<td></td>
<td>Turns ranks of points into ranks of hyperplanes.</td>
</tr>
</tbody>
</table>

Table 37: Orbiter Activities for Objects in PG(n, q) (Part 2)
33 Canonical Forms of Objects in Projective Geometries

In small projective spaces, objects in can be classified using canonical forms of graphs. The canonical forms can be computed using associated graphs. This means that each of the objects to be classified is converted to a graph. The canonical form of the graph can be used to classify the original objects in the projective geometry (cf. [2]). Two objects are isomorphic if and only if the associated graphs have the same canonical form. The method of canonical forms is very popular in graph theory. Orbiter uses Nauty [40] to perform these computations.

Table 38 list Orbiter commands related to canonical forms of objects in projective geometries.

Suppose we want to compute the stabilizer of an elliptic curve. In Section 31, we have created an elliptic curve over \( \mathbb{F}_{11} \) and stored the set of \( \mathbb{F}_q \)-points in the file `elliptic_curve_b1_c3_q11.txt`. The following example creates an input stream to read the elliptic curve from file and then computes the set stabilizer. Input streams have been discussed in Section 6.

```
orbiter.out -v 2
   -define F -finite_field -q 11 -end -end
   -define P -projective_space 2 F -end
   -with P -do
   -projective_space_activity
   -canonical_form_PG
   -input -file_of_points elliptic_curve_b1_c3_q11.txt
   -end
   -classification_prefix EC
   -report
   -end
   -end
pdflatex EC_classification.tex
```

Orbiter shows that the curve has a collineation stabilizer of order 6, generated by

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 1 & 8 \\
5 & 9 & 5 \\
8 & 1 & 1
\end{bmatrix}.
\]
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-input</td>
<td>description</td>
<td>Describe the set of input objects.</td>
</tr>
<tr>
<td>-save_classification</td>
<td>fname</td>
<td>Save the classified list of objects to the given file.</td>
</tr>
<tr>
<td>-fixed_structure_of_element_of_order</td>
<td>o</td>
<td>Report the fixed structure of elements of order o.</td>
</tr>
<tr>
<td>-report</td>
<td>fname</td>
<td>Produce a report of the classification to the given file.</td>
</tr>
<tr>
<td>-max_TDO_depth</td>
<td>d</td>
<td>Limit TDO depth to d in the report.</td>
</tr>
<tr>
<td>-classification_prefix</td>
<td>prefix</td>
<td>Use the given prefix when writing files related to the classification.</td>
</tr>
<tr>
<td>-saveCanonical_labeling</td>
<td></td>
<td>Save the canonical labelings to file.</td>
</tr>
<tr>
<td>-save_ago</td>
<td></td>
<td>Save the automorphism group orders to file.</td>
</tr>
<tr>
<td>-loadCanonical_labeling</td>
<td></td>
<td>Load the canonical labelings from file.</td>
</tr>
<tr>
<td>-load_ago</td>
<td></td>
<td>Load the automorphism group orders from file.</td>
</tr>
<tr>
<td>-save_cumulative_canonical_labeling</td>
<td></td>
<td>Save the canonical labelings of the collected set of objects to a file.</td>
</tr>
<tr>
<td>-save_cumulative_ago</td>
<td></td>
<td>Save the automorphism group orders of the collected set of objects to a file.</td>
</tr>
<tr>
<td>-save_cumulative_data</td>
<td></td>
<td>Save the data of the collected set of objects to a file.</td>
</tr>
<tr>
<td>-save_fibration</td>
<td></td>
<td>Record which objects are related to any given canonical form.</td>
</tr>
</tbody>
</table>

Table 38: Orbiter commands related to canonical forms of objects in projective geometries
### Table 39: Commands for Classifying Arcs

<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>q</td>
<td>Specify the size of the field $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-d</td>
<td>d</td>
<td>Require that no more than $d$ points lie on a line.</td>
</tr>
<tr>
<td>-n</td>
<td>n</td>
<td>The size of the matrix group.</td>
</tr>
<tr>
<td>-target_size</td>
<td>t</td>
<td>Specify the size of the arc to be $t$.</td>
</tr>
<tr>
<td>-conic_test</td>
<td></td>
<td>Require that no 6 points of the arc lie on a conic.</td>
</tr>
<tr>
<td>-affine</td>
<td></td>
<td>Classify arcs in the affine geometry, assuming that $x_0 = 0$ is the hyperplane at infinity. The condition that no more that $d$ point lie on a line applies to affine lines only.</td>
</tr>
<tr>
<td>-no_arc_testing</td>
<td></td>
<td>Do not test the at most $d$ points per line condition.</td>
</tr>
<tr>
<td>-forbidden_point_set</td>
<td>set</td>
<td>The arc must not contain any of the given points.</td>
</tr>
</tbody>
</table>

### 34 Arcs and Caps in Projective Spaces

In this section, we discuss arcs and caps. In $\text{PG}(n,q)$, an arc is a set of points, no $n + 1$ in a hyperplane. A cap is set of points, no three collinear. Here, we restrict our attention to arcs in $\text{PG}(2,q)$. Arcs in higher dimensional projective spaces are equivalent to MDS codes and will be treated in Section 40. Our main examples will be the construction of the Lunelli-Sce hyperoval in $\text{PG}(2,16)$ (cf. [38]) and the Pellegrino cap in $\text{AG}(4,3)$. The uniqueness of this cap was proven by Hill [22].

A $(k,d)$-arc in a projective plane $\pi$ is a set $S$ of $k$ points such that very line intersects $S$ in at most $d$ points. Arcs are related to linear codes and other structures. Two arcs $S_1$ and $S_2$ are equivalent if there is a projectivity $\Phi$ such that $\Phi(A) = B$. The problem of classifying arcs is the problem of determining the orbits of the projectivity group on arcs. At times, we consider the larger group of collineations. In that case, the problem of classifying arcs is the problem of determining the orbits of the collineation group on arcs. Orbiter can solve such classification problems, at least for small parameter cases. Table 39 list the commands available to classify arcs. Here is an example. A hyperoval in a plane $\text{PG}(2,2^e)$ is a $(2^e + 2,2)$-arc. It is interesting to classify the hyperovals up to collineation equivalence under the group $\text{PGL}(3,2^e)$. The command

```
orbiter.out -v 4 \
  -orbiter_path $(ORBITER_PATH) \
  -define G \n  -linear_group -PGGL 3 16 -end -end \
```
performs the classification of hyperovals in PG(2, 16). There are exactly two hyperovals in this plane. Orbiter also finds the stabilizers of these arcs. They have orders 16320 and 144, respectively. The two hyperovals are the regular hyperoval and the Lunelli-Sce hyperoval. Here is the relevant output from the Orbiter report (in the output, the Lunelli-Sce hyperoval is orbit 0, and the regular hyperoval is orbit 1):

### Orbits at Level 18

There are 2 orbits at level 18.

#### Orbit 0 / 2 at Level 18

Node number: 4212

\{0, 1, 2, 3, 52, 67, 89, 106, 126, 141, 159, 176, 184, 199, 220, 235, 245, 262\}_{144}

<table>
<thead>
<tr>
<th>Node</th>
<th>Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 = (1, 0, 0)</td>
</tr>
<tr>
<td>1</td>
<td>1 = (0, 1, 0)</td>
</tr>
<tr>
<td>2</td>
<td>2 = (0, 0, 1)</td>
</tr>
<tr>
<td>3</td>
<td>3 = (1, 1, 1)</td>
</tr>
<tr>
<td>4</td>
<td>52 = (3, 2, 1)</td>
</tr>
<tr>
<td>5</td>
<td>67 = (2, 3, 1)</td>
</tr>
<tr>
<td>6</td>
<td>89 = (8, 4, 1)</td>
</tr>
<tr>
<td>7</td>
<td>106 = (9, 5, 1)</td>
</tr>
<tr>
<td>8</td>
<td>126 = (13, 6, 1)</td>
</tr>
<tr>
<td>9</td>
<td>141 = (12, 7, 1)</td>
</tr>
<tr>
<td>10</td>
<td>159 = (14, 8, 1)</td>
</tr>
<tr>
<td>11</td>
<td>176 = (15, 9, 1)</td>
</tr>
<tr>
<td>12</td>
<td>184 = (7, 10, 1)</td>
</tr>
<tr>
<td>13</td>
<td>199 = (6, 11, 1)</td>
</tr>
<tr>
<td>14</td>
<td>220 = (11, 12, 1)</td>
</tr>
<tr>
<td>15</td>
<td>235 = (10, 13, 1)</td>
</tr>
<tr>
<td>16</td>
<td>245 = (4, 14, 1)</td>
</tr>
<tr>
<td>17</td>
<td>262 = (5, 15, 1)</td>
</tr>
</tbody>
</table>
Strong generators for a group of order 144:

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
\delta^4 & \delta^9 & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & \delta^7 & \delta^{13} \\
\delta^8 & \delta^9 & \delta^{10} \\
\delta & \delta^6 & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
\delta^5 & \delta^5 & \delta^5 \\
\delta^5 & \delta^2 & \delta^{11} \\
\delta^5 & \delta^{14} & 1
\end{bmatrix}
\]

1,0,0;0,1;0,9,5,1,1,
1,7,6,14,5,10,2,15,1,3,
1,1,1,3,15,1,5,10,0,
There are 0 extensions
Number of generators 3

**Orbit 1 / 2 at Level 18**

Node number: 4213

\{0, 1, 2, 3, 52, 70, 83, 109, 127, 139, 156, 174, 186, 199, 217, 229, 256, 264\}_{16320}

0 : 0 = ( 1, 0, 0 ) \quad 10 : 156 = ( 11, 8, 1 )
1 : 1 = ( 0, 1, 0 ) \quad 11 : 174 = ( 13, 9, 1 )
2 : 2 = ( 0, 0, 1 ) \quad 12 : 186 = ( 9, 10, 1 )
3 : 3 = ( 1, 1, 1 ) \quad 13 : 199 = ( 6, 11, 1 )
4 : 52 = ( 3, 2, 1 ) \quad 14 : 217 = ( 8, 12, 1 )
5 : 70 = ( 5, 3, 1 ) \quad 15 : 229 = ( 4, 13, 1 )
6 : 83 = ( 2, 4, 1 ) \quad 16 : 256 = ( 15, 14, 1 )
7 : 109 = ( 12, 5, 1 ) \quad 17 : 264 = ( 7, 15, 1 )
8 : 127 = ( 14, 6, 1 )
9 : 139 = ( 10, 7, 1 )

Strong generators for a group of order 16320:

\[
\begin{bmatrix}
\delta^6 & 0 & 0 \\
0 & \delta^3 & 0 \\
0 & 0 & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
\delta^9 & 0 & 0 \\
0 & \delta^7 & 0 \\
0 & 0 & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
\delta^2 & 0 & 0 \\
0 & \delta^{11} & 0 \\
\delta^4 & \delta^7 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
\delta^{10} & 0 & 0 \\
0 & \delta^3 & 0 \\
\delta & \delta^{11} & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
\delta & 0 & 0 \\
\delta^{12} & \delta^2 & \delta^{15} \\
\delta^{14} & \delta^6 & 1
\end{bmatrix}
, \quad
\begin{bmatrix}
\delta^5 & 0 & 0 \\
\delta^5 & \delta^3 & \delta^6 \\
\delta^6 & \delta^6 & 1
\end{bmatrix}
\]

138
There are 0 extensions
Number of generators 8

Next, we turn our attention to the Pellegrino cap in \( AG(4,3) \). In order to classify large arcs in \( AG(4,3) \), we consider the embedding of \( AG(4,3) \) in \( PG(4,3) \). Suppose we pick the hyperplane at infinity to be \( v(X_0) \). With respect to this choice of hyperplane, we need to determine the set of affine lines. These are the lines which do not lie on the hyperplane at infinity. To this end, we create a cheat sheet for \( PG(4,3) \) using

\[
\text{orbiter.out -cheat_sheet_PG 4 3}
\]
\[
pdflatex PG_4_3.tex
\]

We create the group as a subgroup of \( PGL(5,3) \) using the command

\[
\text{orbiter.out -v 5 \ -draw_options -end \ -define G \ -linear_group -PGL 5 3 -subgroup_by_generators "AGL_4_3" "1965150720" 14 \ "1,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1," \ "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,
\]

\end{verbatim}

\[ \text{report} \]

139
Next, we perform the search for the arcs in this space. Because of the embedding of AG(4, 3) in PG(4, 3), we need to exclude the points on the hyperplane \(v(X_0)\). For this, we use the -forbidden_point_set option of the -classify_arcs command sequence. The points that we exclude are those whose coordinates have \(X_0 = 0\). We can determine the ranks of these points from the cheat sheet. The following Orbiter command performs the search:

```
orbiter.out -v 5 \n  -define G \n  -linear_group -PGL 5 3 \n    -subgroup_by_generators "AGL_4_3" "1965150720" 14 \n      "1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1," \n      "1,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1," \n      "1,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1," \n      "1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1," \n      "1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1," \n      "1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,1," \n      "1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,2," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,2,0,0,0,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,2,0,0,0,0,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,2,0,0,0,0,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1," \n      "1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1," \n      "1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0,0,1," \n    -end -end \n  -with G -do \n    -group_theoretic_activities \n      -poset_classification_control -problem_label arcs_AG_4_3_d2 \n        -W -depth 25 -end \n      -classify_arcs \n        -affine \n        -target_size 25 \n        -q 3 \n        -n 5 \n        -d 2 \n        -forbidden_point_set -long_string \n          "1,2,3,4,10,13,18,21,24,27,30,33,36,39,44,47,50,53,56,59,62," \n          "65,68,71,74,77,80,82,85,88,91,94,97,100,103,106,109,112,115,118" \n          -end_string \n        -end \n      -end \n    -end
```

In order to find the largest arc, we specify a target search depth of 25 using the target_size
and depth commands. The program will show that no cap with 21 points exists. To investigate the Pellegrino cap we use the command

```
orbiter.out -v 2 -canonical_form_PG 4 3 \
   -input -set_of_points \ 
      "0,5,6,8,11,16,19,25,28,43,45,51,57,63,70,99,102,105,107,114" \ 
   -end \ 
   -classification_prefix arc_AG_4_3 \ 
   -report \ 
   -end
```

```
pdflatex arc_AG_4_3_classification.tex
```

It shows that the cap has a stabilizer of order 2880. This command also shows a tactical decomposition of the geometry and the arc within which gives structural information about the arc. The output contains a set of generators for the stabilizer. This group can be recreated and explored using the command

```
orbiter.out -v 5 \ 
   -define G \ 
   -linear_group -PGL 5 3 -subgroup_by_generators \ 
      "AGL_4_3_arc_aut" "2880" 6 \ 
      "1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,2,2,1,2," \ 
      "1,0,0,0,0,1,0,0,0,0,2,1,0,1,0,1,0,1,2,0,2,0,0,2," \ 
      "1,0,0,0,0,2,0,1,2,0,1,1,0,0,0,1,0,0,1,0,2,0,0,0," \ 
      "1,1,0,0,0,2,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,1," \ 
      "1,0,1,0,0,0,1,0,0,0,0,2,0,0,0,0,0,1,0,0,2,0,1,2," \ 
      "1,0,1,0,1,0,2,0,0,0,0,1,2,0,2,0,1,0,1,2,0,2,0,1,0," \ 
   -end -end \ 
   -with G -do \ 
   -group_theoretic_activities \ 
   -export_gap \ 
   -export_magma \ 
   -report
```

```
pdflatex PGL_5_3_Subgroup_AGL_4_3_arc_aut_2880_report.tex
```

This command also exports the group to GAP and Magma.
35 Cubic Curves

Orbiter can classify cubic curves in PG(2, q). To this end, the (9, 3)-arcs in PG(2, q) are classified first. From this classification, the classification of curves is computed. This classification only works for arcs which contain a (9, 3) arc. For very small fields, this is not always the case.

Here is an example. The command sequence

```
cubic_curves_PG_2_8:
  orbiter.out -v 3 -define G \n    -linear_group -PGGL 3 8 -end -end \n    -with G -do \n    -group_theoretic_activities \n    -classify_cubic_curves -q 8 -target_size 9 -n 3 -d 3 \n      -poset_classification_control -problem_label cc_8 -W -depth 9 \n      -draw_options -rad 200 -embedded -end \n    -draw_poset \n  -end \n  -end

pdflatex Cubic_curves_q8.tex
open Cubic_curves_q8.pdf
```

classifies the cubic curves in PG(2, 8).
36 Cubic Surfaces: Creation

Orbiter can create, classify and investigate cubic surfaces. To create a single cubic surface with 27 lines, the commands in Tables 40-41 can be used. Orbiter contains a built-in catalogue of cubic surfaces with 27 lines for small finite fields \( \mathbb{F}_q \) (all surfaces in fields \( \mathbb{F}_q, q \leq 97 \) are built-in, plus some for larger fields). For surfaces created with `-by_coefficients` or `-family_HCV` or `-family_G13` or `-family_F13` or `-catalogue` or `-arc_lifting`, a generating set for the collineation stabilizer will be created as well. Let us look at some examples. The command

```bash
orbiter.out -v 3 -define G -linear_group -PGGL 4 4 -wedge -end -end \
  -with G -do \ 
  -group_theoretic_activities \ 
  -create_surface -q 4 -catalogue 0 -end
```

creates the unique cubic surface with 27 lines over the field \( \mathbb{F}_4 \) which is stored under the index 0 in the catalogue. This is the Hirschfeld surface with 45 Eckardt points.

Another way of creating surfaces is as members of known infinite families. For instance,

```bash
orbiter.out -v 3 -define G -linear_group -PGL 4 13 -wedge -end -end \
  -with G -do \ 
  -group_theoretic_activities \ 
  -create_surface -family_HCV 3 1 -q 13 -end
```

creates the member of the Hilbert, Cohn-Vossen surface described in [10] with parameter \( a = 3 \) and \( b = 1 \) over the field \( \mathbb{F}_{13} \).

The command

```bash
orbiter.out -v 3 -define G -linear_group -PGGL 4 8 -end -end \ 
  -with G -do \ 
  -group_theoretic_activities \ 
  -control_six_arcs -end \ 
  -create_surface -q 8 -arc_lifting "0,1,2,3,28,35" \ 
    -end
```

creates the cubic surface associated with the non-conical 6-arc 0,1,2,3,28,35 over the field \( \mathbb{F}_8 \). The arc has been computed in Section 34. This command uses the algorithm of Karaoglu [28] to create the surface from the arc. The algorithm relies on trihedral pairs and their generalizations, double triplets. The automorphism group of the surface is created as well. A latex report about the surface is written.

The command

```bash
orbiter.out -v 3 -define G -linear_group -PGGL 4 8 -end -end \ 
  -with G -do \ 
  -group_theoretic_activities \ 
  -control_six_arcs -end \ 
```

creates the cubic surface associated with the non-conical 6-arc 0,1,2,3,28,35 over the field \( \mathbb{F}_8 \). The arc has been computed in Section 34. This command uses the algorithm of Karaoglu [28] to create the surface from the arc. The algorithm relies on trihedral pairs and their generalizations, double triplets. The automorphism group of the surface is created as well. A latex report about the surface is written.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>q</td>
<td>Specify the order of the field. The surface will be defined in PG(3, q).</td>
</tr>
<tr>
<td>-catalogue</td>
<td>i</td>
<td>Create the i-th surface in the catalogue. Here, i is an index variable used to index all surfaces in PG(3, q). The index i is zero-based.</td>
</tr>
<tr>
<td>-by_coefficients</td>
<td>list-of-coeff-pairs</td>
<td>Create a surface from a list of coefficient-monomial pairs.</td>
</tr>
<tr>
<td>-family_HCV</td>
<td>a b</td>
<td>Create the Hilbert, Cohn-Vossen surface with parameters (a, b) as in see [10]. The equation is $X_0^3 - b^2(X_0^2 + X_1^2 + X_2^2)X_3 + \frac{b^3}{a}(a^2 + 1)X_0X_1X_2 = 0$.</td>
</tr>
<tr>
<td>-family_G13</td>
<td>a</td>
<td>Create a member of the $G_{13}$ family with parameter a. The surface has 13 Eckardt points.</td>
</tr>
<tr>
<td>-family_F13</td>
<td>a</td>
<td>Create a member of the $F_{13}$ family with parameter a. The surface has 13 Eckardt points.</td>
</tr>
<tr>
<td>-family_bes</td>
<td>a c</td>
<td>Create a member of the bes family with parameter a. The surface has 5 Eckardt points. Bes is the word for five in Turkish.</td>
</tr>
<tr>
<td>-family_general_</td>
<td>a b c d</td>
<td>Create a member of the general family with parameters a, b, c, d.</td>
</tr>
<tr>
<td>abcd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-arc_lifting</td>
<td>A</td>
<td>Create the surface associated with the arc $A = a_1, \ldots, a_6$ in PG(2, q) by means of the Clebsch map. Each of the $a_i$ is the rank of a point in PG(2, q). Use the trihedral pair algorithm. Here, A is a comma-separated string containing the numerical ranks of the $P_i$ in PG(2, q).</td>
</tr>
</tbody>
</table>

Table 40: Commands to create a known cubic surface (Part 1)
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-arc_lifting_with_two_lines</td>
<td>( A \ L )</td>
<td>Create the surface associated with the arc ( a_1, \ldots, a_6 ) in ( \text{PG}(2,q) ) by means of the Clebsch map. Each of the ( a_i ) is the rank of a point in ( \text{PG}(2,q) ). Use the two-lines algorithm. Here, ( A ) is a comma-separated string containing the numerical ranks of the ( P_i ) in ( \text{PG}(3,q) ) and ( L ) is a comma-separated string of the numerical ranks of two lines in ( \text{PG}(3,q) ). If both of the lines are given as 0, the program will pick a suitable set of lines automatically.</td>
</tr>
<tr>
<td>-select_double_six</td>
<td>( L )</td>
<td>Relabel the lines by choosing the 12 lines in ( L ) as new double six. The entries in ( L ) are line indices with respect to the old double six. They are integers in the interval ([0, 26]). This command can be repeated. In each application, the double six refers to the previous labeling.</td>
</tr>
<tr>
<td>-transform</td>
<td>( A )</td>
<td>Transform the surface by the projectivity (or collineation) defined by ( A ). This option can be repeated.</td>
</tr>
<tr>
<td>-transform_inverse</td>
<td>( A )</td>
<td>Transform the surface by the inverse projectivity (or collineation) defined by ( A ). This option can be repeated.</td>
</tr>
</tbody>
</table>

Table 41: Commands to create a known cubic surface (Part 2)
-create_surface -q 8 -arc_lifting_with_two_lines \
"0,1,2,20,30,37" "24,1898" \
-end

creates the same cubic surface, using a different algorithm. The arc is now given as elements in PG(3, 8), and the ranks of two lines in PG(3, 8) are given as well. The two lines \( \ell_1 \) and \( \ell_2 \) are skew and pass through the first two points of the arc. They must not lie in the plane containing the arc. This command does not create the automorphism group of the surface.

It is possible to apply a transformation to the surface. Suppose we are interested in the surface over \( \mathbb{F}_8 \) created in (4). The command

```
orbiter.out -v 3 -define G -linear_group -PGGL 4 8 -wedge -end -end \
-with G -do \n-group_theoretic_activities \n-create_surface -q 8 -catalogue 0 \n-transform_inverse "1,4,4,0,6,0,0,0,6,2,0,1,7,0,4,0,0" \n-end 
```

creates surface 0 over \( \mathbb{F}_8 \) and applies the inverse transformation to recover the surface whose equation was given in (4). The surface number 0 over \( \mathbb{F}_8 \) is created, and the transformation (5) is applied in inverse. The commands `-transform` and `-transform_inverse` accept the transformation matrix in row-major ordering, with the field automorphism as additional element. It is possible to give a sequence of transformations. In this case, the transformations are applied in the order in which the commands are given on the command line. The command

```
orbiter.out -v 3 -define G -linear_group -PGGL 4 8 -wedge -end -end \
-with G -do \n-group_theoretic_activities \n-create_surface -q 8 -catalogue 0 \n-select_double_six "15,11,22,19,24,5,16,10,23,20,25,4" \n-select_double_six "3,2,1,0,5,4,9,8,7,6,11,10" \n-transform_inverse "1,4,4,0,6,0,0,0,6,2,0,1,7,0,4,0,0" \n-transform "4,4,0,0, 0,0,1,0, 1,0,0,0, 0,0,0,1, 0"
-end 
```

can be used to create the surface number 0 over \( \mathbb{F}_8 \), so that the 13 Eckardt points lie in the plane \( \pi = [0, 0, 0, 1]^\perp \). The double six has been relabeled in such a way that the Clebsch map \( \Phi_{b_1,b_2,\pi} \) sends the lines \( a_i \) to \( P_i \), for \( i = 1, \ldots, 6 \). A latex report is written. A closer inspection of the latex report shows that

\[
P_1 = (1, 0, 0), \ P_2 = (0, 1, 0), \ P_3 = (0, 0, 1), \ P_4 = (1, 1, 1), \ P_5 = (2, 3, 1), \ P_6 = (3, 2, 1).
\]
Table 42: Group theoretic activities related to the classification of cubic surfaces

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-surface_classify</td>
<td></td>
<td>Classifies cubic surfaces using double sixes. The group must be $\text{PGL}(4,q)$ (or any subgroup) in the wedge action.</td>
</tr>
<tr>
<td>-classify_surfaces_through_arcs_and_trihedral_pairs</td>
<td></td>
<td>Classifies the surfaces using the associated arcs using the algorithm of [28]. The group must be $\text{PGL}(4,q)$ (or any subgroup) in the standard action.</td>
</tr>
<tr>
<td>-classify_surfaces_through_arcs_and_two_lines</td>
<td></td>
<td>Classifies the surfaces using the associated arcs using the algorithm of [27]. The group must be $\text{PGL}(4,q)$ (or any subgroup) in the standard action.</td>
</tr>
<tr>
<td>-report</td>
<td></td>
<td>Produce a report (needs surface_classify).</td>
</tr>
<tr>
<td>-trihedral1_control</td>
<td></td>
<td>Poset classification control for classification of trihedra of type 1.</td>
</tr>
<tr>
<td>-trihedral2_control</td>
<td></td>
<td>Poset classification control for classification of trihedra of type 2.</td>
</tr>
<tr>
<td>-control_six_arcs</td>
<td></td>
<td>Poset classification control for classification of six-arcs.</td>
</tr>
</tbody>
</table>

Table 42: Group theoretic activities related to the classification of cubic surfaces

37 Cubic Surfaces: Classification and Recognition

Let us now address the problems of classification and recognition of surfaces. There are several different approaches to classify cubic surfaces over finite fields with 27 lines under the collineation group $\text{PGL}(4,q)$. One approach is described in [10] and relies on Schläfli’s notion of a double six as a substructure [48]. Another approach is through non-conical six-arcs in a plane, as described in [28]. A third approach is using the theory described in [27]. All three approaches are available in Orbiter.

Table 42 lists Orbiter commands related to the classification of cubic surfaces. These commands are group theoretic activities as described in Section 25.

The only cubic surface with 27 lines in $\text{PG}(3,4)$ is the Hirschfeld surface of [23]. The following Orbiter command can be used to prove this. The command

```
orbiter.out -v 3 -define G \n  -linear_group -PGGL 4 4 -wedge -end -end \n  -with G -do \n  -group_theoretic_activities \n  -poset_classification_control -W -end \n```
classifies all cubic surfaces with 27 lines over the field \( \mathbb{F}_4 \) under the collineation group \( \text{PGL}(4, 4) \) using the algorithm of [10]. The report option creates a latex report. After some redactions, the report contains the following elements.

### The semilinear group

#### The Action

Group action \( \text{PGL}(4, 4) \) of degree 85
The group is a matrix group.

The base action is on projective space \( \text{PG}(3, 4) \)
- \( q = 4 \)
- \( p = 2 \)
- \( e = 2 \)
- \( n = 3 \)
Number of points = 85
Number of lines = 357
Number of lines on a point = 21
Number of points on a line = 5

### The orthogonal group

#### The Action

Group action \( \text{PGL}(4, 4) \text{OnWedge} \) of degree 1365
The group is a matrix group.
The base action is on projective space \( \text{PG}(3, 4) \)
- \( q = 4 \)
- \( p = 2 \)
- \( e = 2 \)
- \( n = 3 \)
Number of points = 85
Number of lines = 357
Number of lines on a point = 21
Number of points on a line = 5
The group stabilizing the fixed line

The Action

Group action $\text{PGL}(4,4)\text{OnWedgeres100}$ of degree 100

Strong generators for a group of order 5529600:

The classification of five-plus-ones

Poset classification up to depth 5

The Orbits

Number of Orbits By Level

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nb of orbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Summary of Orbit Representatives

N = node
D = depth or level
O = orbit with a level
Rep = orbit representative
$(S,O)$ = (order of stabilizer, orbit length)
L = number of live points
F = number of flags
Gen = number of generators for the stabilizer of the orbit rep.
Table 43: Orbit Representatives

<table>
<thead>
<tr>
<th>N</th>
<th>D</th>
<th>O</th>
<th>Rep</th>
<th>(S,O)</th>
<th>L</th>
<th>F</th>
<th>Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{}</td>
<td>(5529600, 1)</td>
<td>100</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>{ 0 }</td>
<td>(55296, 100)</td>
<td>64</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>{ 0, 3 }</td>
<td>(1728, 3200)</td>
<td>36</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>{ 0, 3, 56 }</td>
<td>(144, 38400)</td>
<td>16</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>{ 0, 3, 56, 76 }</td>
<td>(288, 19200)</td>
<td>4</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>{ 0, 3, 56, 77 }</td>
<td>(96, 57600)</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>{ 0, 3, 56, 80 }</td>
<td>(72, 76800)</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>0</td>
<td>{ 0, 3, 56, 76, 96 }</td>
<td>(1440, 3840)</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1</td>
<td>{ 0, 3, 56, 76, 97 }</td>
<td>(96, 57600)</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>2</td>
<td>{ 0, 3, 56, 80, 92 }</td>
<td>(360, 15360)</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>3</td>
<td>{ 0, 3, 56, 80, 93 }</td>
<td>(120, 46080)</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>
Poset of Orbits in Detail

Classification of 5 + 1 Configurations in PG(3, 4)

The order of the group is 1974067200
The group has 4 orbits on five plus one configurations in PG(3, 4).

Of these, 1 impose 19 conditions.
Of these, 1 are associated with double sixes. They are:
0/1 is orbit 3/4 \{0, 3, 56, 80, 93\}_{120} orbit length 46080
The overall number of five plus one configurations associated with double sixes in PG(3, 4) is: 46080

Flag orbits for double sixes

The number of primary orbits below is 4
The number of primary orbits above is 1
The number of flag orbits is 1
The flag orbits are:

(1) Flag orbit 0 / 1 down=(3,0) up=(0,-1) is ( 0, 3, 56, 80, 93, 16, 340, 38, 61, 156, 0, 16, 340, 38, 61, 156, 165, 72, 54, 25, 356, 0 ) with a stabilizer of order 120
Strong generators for a group of order 120:

\[
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & \omega^2 & 0 & 0 \\
  0 & 0 & \omega^2 & 0 \\
  0 & 0 & 0 & 1 \\
\end{bmatrix}_{1},
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & \omega & 0 & 0 \\
  \omega^2 & 0 & 0 & 0 \\
  0 & 0 & \omega & 0 \\
\end{bmatrix}_{1},
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  \omega & 0 & 0 & 0 \\
  \omega^2 & \omega & 0 & 0 \\
  0 & 0 & 0 & \omega \\
\end{bmatrix}_{1},
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  \omega^2 & 0 & 0 & 1 \\
  \omega & 0 & 0 & 0 \\
  0 & 0 & \omega & 0 \\
\end{bmatrix}_{1},
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  \omega & \omega & 0 & 0 \\
  \omega^2 & 1 & \omega^2 & 1 \\
  0 & 0 & 1 & 0 \\
\end{bmatrix}_{1}
\]
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
\omega & \omega^1 & \omega^2 \\
0 & \omega^2 & 1
\end{bmatrix}
\]

1,0,0,0,0,3,0,0,0,0,3,0,0,0,0,1,1,
1,0,0,0,0,2,0,0,3,0,2,0,0,3,0,1,1,
1,0,0,0,3,2,0,0,0,2,0,0,0,3,1,1,
1,0,0,0,3,3,0,0,0,3,0,0,0,1,1,0,
1,0,0,0,3,2,0,0,2,0,2,0,3,1,3,1,0,
1,1,0,0,3,0,0,0,0,3,3,0,0,1,1,0,1,
1,2,0,0,1,0,0,2,2,1,3,0,3,0,1,1,

nb received = 0

**Double Sixes**

The order of the group is 1974067200

The group has 1 orbits:

The orbits are:

(1) 0/1 \{16, 340, 38, 61, 156, 165, 155, 72, 54, 25, 356, 0\}_{1440} orbit length 1370880

Strong generators for a group of order 1440:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
0 & 0 & \omega^2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}_1,
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
\omega^2 & 0 & \omega^2 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}_0,
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
\omega^2 & \omega^2 & 0 & 0 \\
\omega^2 & \omega^2 & 1 & 1
\end{bmatrix}_0
\]

1,0,0,0,0,3,0,0,0,0,3,0,0,0,0,1,1,
1,0,0,0,0,3,0,0,3,0,3,0,0,1,1,0,
1,0,0,0,3,0,2,0,2,2,0,0,3,3,1,1,0,
1,0,0,0,2,0,2,0,1,2,0,0,2,1,2,1,1,
0,0,1,0,0,0,2,1,1,0,3,0,3,1,3,2,0,
1,1,0,0,3,0,0,0,0,0,3,3,0,0,1,0,1,
The overall number of objects is: 1370880

Flag orbits for surfaces

The number of primary orbits below is 1
The number of primary orbits above is 1
The number of flag orbits is 1
The flag orbits are:

(1) Flag orbit 0 / 1 down=(0,0) up=(0,-1) is (16, 340, 38, 61, 156, 165, 72, 54, 25, 356, 0, 71, 55, 26, 100, 1, 138, 109, 345, 84, 85, 122, 110, 145, 139, 81) with a stabilizer of order 1440

Strong generators for a group of order 1440:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
0 & 0 & \omega^2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
, \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
\omega^2 & 0 & \omega^2 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
, \begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & 0 & \omega & 0 \\
\omega & \omega & 0 & 0 \\
\omega^2 & \omega^2 & 1 & 1
\end{bmatrix}
, \begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & 0 & \omega & 0 \\
1 & 0 & 0 & 0 \\
\omega & 1 & \omega & 1
\end{bmatrix}
, \begin{bmatrix}
0 & 0 & \omega^2 & 0 \\
0 & 0 & 1 & \omega^2 \\
\omega^2 & 0 & \omega & 0 \\
\omega & \omega & \omega & 1
\end{bmatrix}
, \begin{bmatrix}
1 & 1 & 0 & 0 \\
\omega^2 & 0 & 0 & 0 \\
0 & 0 & \omega^2 & \omega^2 \\
0 & 0 & 1 & 0
\end{bmatrix}
, \begin{bmatrix}
1,0,0,0,0,3,0,0,0,0,0,0,0,0,1,1, \\
1,0,0,0,3,0,0,3,0,0,0,1,0,1,0, \\
1,0,0,0,3,0,2,0,2,2,0,0,3,3,1,1,0, \\
1,0,0,0,2,0,2,0,1,2,0,0,2,1,2,1,1, \\
0,0,1,0,0,2,1,1,0,3,0,3,1,3,2,0, \\
1,1,0,0,3,0,0,0,0,0,0,3,3,0,0,1,0,1, \\
nb received = 0
\end{bmatrix}

Surfaces

The order of the group is 1974067200
The group has 1 orbits:

The orbits are:

(1) 0/1 \{16, 340, 38, 61, 156, 165, 155, 72, 54, 25, 356, 0, 71, 55, 26, 100, 1, 138, 109, 345, 84, 85, 122, 110, 145, 139, 81\}_{51840} orbit length 38080
Strong generators for a group of order 51840:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega & 0 & 0 \\
0 & 0 & \omega & 0 \\
0 & 0 & 0 & \omega^2
\end{bmatrix}, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
0 & 0 & \omega & 0 \\
0 & 0 & \omega^2 & 0
\end{bmatrix},
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1
\end{bmatrix}, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega^2 & 0 \\
0 & 0 & \omega & 0 \\
0 & 0 & \omega^2 & 1
\end{bmatrix}, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\omega & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}, \quad
\begin{bmatrix}
\omega & \omega & \omega^2 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & 1 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1, 1,0,0,0,0,2,0,0,0,0,2,0,0,0,0,1,0, 1,0,0,0,0,3,0,0,0,0,3,0,1,0,0,1,0, 1,0,0,0,0,1,0,0,1,1,1,0,1,1,0,1,0, 1,0,0,0,3,2,2,0,0,0,2,0,1,0,3,1,0, 1,0,0,0,1,0,1,0,0,1,1,1,0,1,0,1,0, 1,0,0,0,3,2,2,0,0,0,2,0,1,0,3,1,0, 1,0,0,0,1,0,2,0,2,2,0,0,2,2,1,1,0, 1,3,1,2,1,0,2,0,3,2,0,0,2,0,0,0,0, 1,1,3,3,0,3,0,1,1,2,0,1,0,3,0,0,0,

The overall number of objects is: 38080

**The Group** PGL(4, 4)

The order of the group is 1974067200

**Cubic Surfaces with 27 Lines in** PG(3, 4)

The order of the group is 1974067200

The group has 1 orbits:

The orbits are:
Strong generators for a group of order 51840:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega & 0 & 0 \\
0 & 0 & \omega & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
0 & 0 & \omega^2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega^2 & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & 0 & 0 \\
\omega & 0 & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega^2 & \omega & 0 & 0 \\
\omega & \omega & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 1 \\
\omega & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega & \omega & 0 & 0 \\
The automorphism group is the following group

Strong generators for a group of order 51840:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega & 0 & 0 \\
0 & 0 & \omega & 0 \\
0 & 0 & 0 & \omega^2 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
0 & 0 & \omega^2 & 0 \\
1 & 0 & 0 & 1 \\
\end{bmatrix}
, \\
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
\omega^2 & \omega & \omega & 0 \\
0 & 0 & \omega & 0 \\
\omega & \omega & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \\
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & \omega^2 \\
0 & 1 & 0 & \omega \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & \omega^2 \\
0 & 1 & 0 & \omega^2 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
, \\
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
, \quad
\begin{bmatrix}
\omega & \omega & 1 & 1 \\
\omega^2 & 0 & 1 & 0 \\
\omega & \omega & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
,
\]

General information

Points on lines:

\[5^{27}\]

Lines on points:

\[3^{45}\]

The 27 Lines

\[\ell_0 = a_1 = \begin{bmatrix}
1 & 0 & \omega^2 & 0 \\
0 & 1 & 1 & \omega \\
\end{bmatrix}_{72} = \begin{bmatrix}
1 & 0 & 3 & 0 \\
0 & 1 & 1 & 2 \\
\end{bmatrix}_{72} = \text{Pl}(3,2,3,0,3,1)_{308}\]

\[\ell_1 = a_2 = \begin{bmatrix}
1 & 0 & \omega & 0 \\
0 & 1 & 0 & \omega^2 \\
\end{bmatrix}_{54} = \begin{bmatrix}
1 & 0 & 2 & 0 \\
0 & 1 & 0 & 3 \\
\end{bmatrix}_{54} = \text{Pl}(2,3,0,2,1)_{238}\]
\ell_2 = a_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = P(1,1,0,0,1,1)_{177}

\ell_3 = a_4 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = P(0,1,0,0,0,0)_{1}

\ell_4 = a_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = P(1,0,0,0,0,0)_{0}

\ell_5 = a_6 = \begin{bmatrix} 1 & 0 & \omega^2 & 1 \\ 0 & 1 & 0 & \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 & 1 \\ 0 & 1 & 0 & 2 \end{bmatrix} = P(3,2,0,2,3,1)_{314}

\ell_6 = b_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = P(0,0,0,1,0,0)_{9}

\ell_7 = b_2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = P(0,0,1,1,1,1)_{198}

\ell_8 = b_3 = \begin{bmatrix} 1 & \omega & 0 & 0 \\ 0 & 0 & 1 & \omega^2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 3 \end{bmatrix} = P(0,0,2,3,2,1)_{265}

\ell_9 = b_4 = \begin{bmatrix} 1 & 0 & \omega^2 & 1 \\ 0 & 1 & 1 & \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix} = P(3,0,3,2,3,1)_{335}

\ell_{10} = b_5 = \begin{bmatrix} 1 & \omega & 0 & 1 \\ 0 & 0 & 1 & \omega \end{bmatrix} = \begin{bmatrix} 1 & 3 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix} = P(0,2,3,2,3,1)_{337}

\ell_{11} = b_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = P(0,0,1,0,0,0)_{2}

\ell_{12} = c_{12} = \begin{bmatrix} 1 & 0 & \omega & 1 \\ 0 & 1 & 0 & \omega^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 0 & 3 \end{bmatrix} = P(2,3,0,3,2,1)_{256}

\ell_{13} = c_{13} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} = P(1,1,0,1,1,1)_{189}

\ell_{14} = c_{14} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = P(0,1,0,1,0,0)_{13}

\ell_{15} = c_{15} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} = P(1,0,0,1,0,0)_{10}
\[ \ell_{16} = c_{16} = \begin{bmatrix} 1 & 0 & \omega^2 & 0 \\ 0 & 1 & 0 & \omega \end{bmatrix}_{71} = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}_{71} = \text{Pl}(3, 2, 0, 0, 3, 1)_{299} \]

\[ \ell_{17} = c_{23} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}_{85} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}_{85} = \text{Pl}(1, 1, 1, 0, 0, 0)_{16} \]

\[ \ell_{18} = c_{24} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}_{122} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}_{122} = \text{Pl}(0, 1, 1, 1, 1, 1)_{202} \]

\[ \ell_{19} = c_{25} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}_{110} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}_{110} = \text{Pl}(1, 0, 1, 0, 1, 1)_{199} \]

\[ \ell_{20} = c_{26} = \begin{bmatrix} 1 & 0 & \omega & 0 \\ 0 & 1 & 1 & \omega^2 \end{bmatrix}_{55} = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 1 & 3 \end{bmatrix}_{55} = \text{Pl}(2, 3, 2, 0, 2, 1)_{244} \]

\[ \ell_{21} = c_{34} = \begin{bmatrix} 1 & \omega & 0 & 1 \\ 0 & 0 & 1 & \omega^2 \end{bmatrix}_{145} = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix}_{145} = \text{Pl}(0, 3, 2, 3, 2, 1)_{271} \]

\[ \ell_{22} = c_{35} = \begin{bmatrix} 1 & 0 & \omega & 1 \\ 0 & 1 & 1 & \omega^2 \end{bmatrix}_{139} = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 3 \end{bmatrix}_{139} = \text{Pl}(2, 0, 2, 3, 2, 1)_{267} \]

\[ \ell_{23} = c_{36} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}_{26} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}_{26} = \text{Pl}(1, 1, 1, 0, 1, 1)_{180} \]

\[ \ell_{24} = c_{45} = \begin{bmatrix} 1 & \omega^2 & 0 & 0 \\ 0 & 0 & 1 & \omega \end{bmatrix}_{81} = \begin{bmatrix} 1 & 3 & 0 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}_{81} = \text{Pl}(0, 0, 3, 2, 3, 1)_{332} \]

\[ \ell_{25} = c_{46} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{100} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{100} = \text{Pl}(0, 1, 1, 0, 0, 0)_{6} \]

\[ \ell_{26} = c_{56} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}_{1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}_{1} = \text{Pl}(1, 0, 1, 0, 0, 0)_{3} \]

Rank of lines: (72, 54, 25, 356, 0, 155, 340, 38, 61, 156, 165, 16, 138, 109, 345, 84, 71, 85, 122, 110, 55, 145, 139, 26, 81, 100, 1)

Rank of points on Klein quadric: (308, 238, 177, 1, 0, 314, 9, 198, 265, 335, 337, 2, 256, 189, 13, 10, 299, 16, 202, 199, 244, 271, 267, 180, 332, 6, 3)

All Points on surface

The surface has 45 points
Eckardt Points
The surface has 45 Eckardt points:
0 : E56 = a5 ∩ b6 ∩ c56 = P0 = P0 = P(1, 0, 0, 0) = P(1, 0, 0, 0), T = 0
1 : E51 = a5 ∩ b1 ∩ c15 = P1 = P1 = P(0, 1, 0, 0) = P(0, 1, 0, 0), T = 4
2 : E46 = a4 ∩ b6 ∩ c46 = P2 = P2 = P(0, 0, 1, 0) = P(0, 0, 1, 0), T = 20
3 : E41 = a4 ∩ b1 ∩ c14 = P3 = P3 = P(0, 0, 0, 1) = P(0, 0, 0, 1), T = 84
4 : E32 = a3 ∩ b2 ∩ c23 = P4 = P4 = P(1, 1, 1, 1) = P(1, 1, 1, 1), T = 27
5 : E52 = a5 ∩ b2 ∩ c25 = P5 = P5 = P(1, 1, 0, 0) = P(1, 1, 0, 0), T = 1
6 : E54 = a5 ∩ b4 ∩ c45 = P6 = P6 = P(ω, 1, 0, 0) = P(2, 1, 0, 0), T = 2
7 : E53 = a5 ∩ b3 ∩ c35 = P7 = P7 = P(ω 2 , 1, 0, 0) = P(3, 1, 0, 0), T = 3
8 : E36 = a3 ∩ b6 ∩ c36 = P8 = P8 = P(1, 0, 1, 0) = P(1, 0, 1, 0), T = 5
9 : E16 = a1 ∩ b6 ∩ c16 = P9 = P9 = P(ω, 0, 1, 0) = P(2, 0, 1, 0), T = 10
10 : E26 = a2 ∩ b6 ∩ c26 = P10 = P10 = P(ω 2 , 0, 1, 0) = P(3, 0, 1, 0), T = 15
11 : E14,23,56 = c14 ∩ c23 ∩ c56 = P11 = P11 = P(0, 1, 1, 0) = P(0, 1, 1, 0), T = 9
12 : E13,24,56 = c13 ∩ c24 ∩ c56 = P12 = P12 = P(1, 1, 1, 0) = P(1, 1, 1, 0), T = 6
13 : E65 = a6 ∩ b5 ∩ c56 = P13 = P13 = P(ω, 1, 1, 0) = P(2, 1, 1, 0), T = 12
14 : E12,34,56 = c12 ∩ c34 ∩ c56 = P14 = P14 = P(ω 2 , 1, 1, 0) = P(3, 1, 1, 0), T = 18
15 : E15,23,46 = c15 ∩ c23 ∩ c46 = P15 = P23 = P(1, 0, 0, 1) = P(1, 0, 0, 1), T = 21
16 : E31 = a3 ∩ b1 ∩ c13 = P16 = P26 = P(0, 1, 0, 1) = P(0, 1, 0, 1), T = 25
17 : E15,24,36 = c15 ∩ c24 ∩ c36 = P17 = P27 = P(1, 1, 0, 1) = P(1, 1, 0, 1), T = 22
18 : E21 = a2 ∩ b1 ∩ c12 = P18 = P30 = P(0, ω, 0, 1) = P(0, 2, 0, 1), T = 46
19 : E15,26,34 = c15 ∩ c26 ∩ c34 = P19 = P31 = P(1, ω, 0, 1) = P(1, 2, 0, 1), T = 24
20 : E61 = a6 ∩ b1 ∩ c16 = P20 = P34 = P(0, ω 2 , 0, 1) = P(0, 3, 0, 1), T = 67
21 : E15 = a1 ∩ b5 ∩ c15 = P21 = P35 = P(1, ω 2 , 0, 1) = P(1, 3, 0, 1), T = 23
22 : E42 = a4 ∩ b2 ∩ c24 = P22 = P38 = P(0, 0, 1, 1) = P(0, 0, 1, 1), T = 41
23 : E13,25,46 = c13 ∩ c25 ∩ c46 = P23 = P39 = P(1, 0, 1, 1) = P(1, 0, 1, 1), T = 26
24 : E14,25,36 = c14 ∩ c25 ∩ c36 = P24 = P42 = P(0, 1, 1, 1) = P(0, 1, 1, 1), T = 30
25 : E62 = a6 ∩ b2 ∩ c26 = P25 = P47 = P(ω, ω, 1, 1) = P(2, 2, 1, 1), T = 53
26 : E25 = a2 ∩ b5 ∩ c25 = P26 = P48 = P(ω 2 , ω, 1, 1) = P(3, 2, 1, 1), T = 80
27 : E16,25,34 = c16 ∩ c25 ∩ c34 = P27 = P51 = P(ω, ω 2 , 1, 1) = P(2, 3, 1, 1), T = 55
28 : E12 = a1 ∩ b2 ∩ c12 = P28 = P52 = P(ω 2 , ω 2 , 1, 1) = P(3, 3, 1, 1), T = 79
29 : E43 = a4 ∩ b3 ∩ c34 = P29 = P53 = P(0, 0, ω, 1) = P(0, 0, 2, 1), T = 62
30 : E12,35,46 = c12 ∩ c35 ∩ c46 = P30 = P54 = P(1, 0, ω, 1) = P(1, 0, 2, 1), T = 36
31 : E35 = a3 ∩ b5 ∩ c35 = P31 = P59 = P(ω, 1, ω, 1) = P(2, 1, 2, 1), T = 49
32 : E63 = a6 ∩ b3 ∩ c36 = P32 = P60 = P(ω 2 , 1, ω, 1) = P(3, 1, 2, 1), T = 76
33 : E14,26,35 = c14 ∩ c26 ∩ c35 = P33 = P61 = P(0, ω, ω, 1) = P(0, 2, 2, 1), T = 51
34 : E23 = a2 ∩ b3 ∩ c23 = P34 = P62 = P(1, ω, ω, 1) = P(1, 2, 2, 1), T = 39
35 : E13 = a1 ∩ b3 ∩ c13 = P35 = P67 = P(ω, ω 2 , ω, 1) = P(2, 3, 2, 1), T = 50
36 : E16,24,35 = c16 ∩ c24 ∩ c35 = P36 = P68 = P(ω 2 , ω 2 , ω, 1) = P(3, 3, 2, 1), T = 74
37 : E45 = a4 ∩ b5 ∩ c45 = P37 = P69 = P(0, 0, ω 2 , 1) = P(0, 0, 3, 1), T = 83
38 : E64 = a6 ∩ b4 ∩ c46 = P38 = P70 = P(1, 0, ω 2 , 1) = P(1, 0, 3, 1), T = 31
39 : E12,36,45 = c12 ∩ c36 ∩ c45 = P39 = P75 = P(ω, 1, ω 2 , 1) = P(2, 1, 3, 1), T = 59
40 : E34 = a3 ∩ b4 ∩ c34 = P40 = P76 = P(ω 2 , 1, ω 2 , 1) = P(3, 1, 3, 1), T = 71

159


41 : \( E_{24} = a_2 \cap b_4 \cap c_{24} = P_{41} = P_{79} = P(\omega, \omega, \omega, 1) = P(2, 2, 3, 1), \quad T = 58 \)
42 : \( E_{13,26,45} = c_{13} \cap c_{26} \cap c_{45} = P_{42} = P_{80} = P(\omega^2, \omega, \omega^2, 1) = P(3, 2, 3, 1), \quad T = 70 \)
43 : \( E_{14} = a_1 \cap b_4 \cap c_4 = P_{43} = P_{81} = P(0, \omega^2, \omega^2, 1) = P(0, 3, 3, 1), \quad T = 72 \)
44 : \( E_{16,23,45} = c_{16} \cap c_{23} \cap c_{45} = P_{44} = P_{82} = P(1, \omega^2, \omega^2, 1) = P(1, 3, 3, 1). \quad T = 33 \)

Set of tangent planes: ( 0, 4, 20, 84, 27, 1, 2, 3, 5, 10, 15, 9, 6, 12, 18, 21, 25, 22, 46, 24, 67, 23, 41, 26, 30, 53, 80, 55, 79, 62, 36, 49, 76, 51, 39, 50, 74, 83, 31, 59, 71, 58, 70, 72, 33 )

Line type of Eckardt points: 5^{27}, 3^{240}, 1^{90}
Plane type of Eckardt points: 13^{45}, 9^{40}

**Hesse planes**

Number of Hesse planes: 40
Set of Hesse planes: ( 7, 8, 11, 13, 14, 16, 17, 19, 28, 29, 32, 34, 35, 37, 38, 40, 42, 43, 44, 45, 47, 48, 52, 54, 56, 57, 60, 61, 63, 64, 65, 66, 68, 69, 73, 75, 77, 78, 81, 82 )

subspace 0 / 40 is 7:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & \omega
\end{bmatrix}
\]

\,

subspace 39 / 40 is 82:

\[
\begin{bmatrix}
1 & 0 & \omega^2 & 0 \\
0 & 1 & \omega^2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

0 : 7 : \( E_{56}, E_{31}, E_{15,24,36}, E_{16,25,34}, E_{12}, E_{14,26,35}, E_{23}, E_{45}, E_{64} \)

39 : 82 : \( E_{41}, E_{52}, E_{16}, E_{12,34,56}, E_{15,24,36}, E_{35}, E_{23}, E_{64}, E_{13,26,45} \)

**Axes**

Number of axes: 240
Axes:

0 : 0 = 0,0 = E_{23}, E_{31}, E_{12} 

\,

239 : 239 = 119,1 = E_{12,36,45}, E_{14,26,35}, E_{13,25,46}
Tritangent planes

The 45 tritangent planes are:

\[
\pi_{12} = \pi_0 = 79 = \begin{bmatrix}
1 & 0 & 0 & \omega^2 \\
0 & 1 & 0 & \omega^2 \\
0 & 0 & 1 & 1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 3 \\
0 & 1 & 0 & 3 \\
0 & 0 & 1 & 1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 3 \\
0 & 1 & 0 & 3 \\
0 & 0 & 1 & 1
\end{bmatrix} = V(\omega^2 X_0 + \omega^2 X_1 + X_2 + X_3) = V(3X_0 + 3X_1 + X_2 + X_3)
\]

dual pt rank = 52 = (3, 3, 1, 1).

\[
\pi_{16,25,34} = \pi_{44} = 55 = \begin{bmatrix}
1 & 0 & 0 & \omega \\
0 & 1 & 0 & \omega \\
0 & 0 & 1 & \omega^2
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 2 \\
0 & 1 & 0 & 2 \\
0 & 0 & 1 & 3
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 & 2 \\
0 & 1 & 0 & 2 \\
0 & 0 & 1 & 3
\end{bmatrix} = V(\omega X_0 + \omega X_1 + \omega^2 X_2 + X_3) = V(2X_0 + 2X_1 + 3X_2 + X_3)
\]

dual pt rank = 79 = (2, 2, 3, 1).

Karaoglu [28] describes a different algorithm, based on non-conical six-arcs and trihedral pairs. The command

\[
\text{orbiter.out -v 4 -define G} \ \\text{linear_group} \ \text{-PGGL 4 4 -end -end} \ \\text{-with G} \ \text{-do} \ \\text{-group_theoretic_activities} \ \\text{-trihedral_control} \ \text{-problem_label tri1_q4 -end} \ \\text{-trihedra2_control} \ \text{-problem_label tri2_q4 -end} \ \\text{-control_six_arcs} \ \text{-problem_label sixarcs_q4 -end} \ \\text{-classify_surfaces_through_arcs_and_trihedral_pairs} \ \text{-end}
\]

classifies all cubic surfaces with 27 lines over the field \(\mathbb{F}_4\) using the algorithm of Karaoglu. Of course, the result agrees with the previous algorithm. The only surface in PG(3, 4) is the Hirschfeld surface.

Besides classification, Orbiter provides recognition, isomorphism testing and study of cubic surfaces. Table 44 lists the relevant Orbiter commands. These commands are group theoretic activities as described in Section 25.
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-surface_identify_HCV</td>
<td></td>
<td>Identifies the isomorphism type of the Hilbert Cohn-Vossen surface with parameter $a$. All values of $a$ are considered.</td>
</tr>
<tr>
<td>-surface_identify_F13</td>
<td></td>
<td>Identifies the isomorphism type of the $F_{13}$ surface with parameter $a$. All values of $a$ are considered.</td>
</tr>
<tr>
<td>-surface_identify_Bes</td>
<td></td>
<td>Identifies the isomorphism type of the Bes surface with parameters $a$ and $c$. All values of $a, c$ are considered.</td>
</tr>
<tr>
<td>-surface_identify_general_abcd</td>
<td></td>
<td>Identifies the isomorphism type of the general surface with parameters $a, b, c, d$. All values of $a, b, c, d$ are considered.</td>
</tr>
<tr>
<td>-surface_isomorphism_testing</td>
<td>surface-descr-1, surface-descr-2</td>
<td>Computes an isomorphism between two given surfaces or concludes that none exists.</td>
</tr>
<tr>
<td>-surface_recognize</td>
<td>surface-descr</td>
<td>Identifies the isomorphism type of the given surface.</td>
</tr>
<tr>
<td>-create_surface</td>
<td>surface-descr</td>
<td>Creates a surface from a description. See Section 36.</td>
</tr>
</tbody>
</table>

Table 44: Group theoretic activities related to the recognition of cubic surfaces
The -surface_recognize option can be used to identify a given surface in the list produced by the classification. The command computes an isomorphism between the given surface and the surface in the catalogue. For instance,

```
orbiter.out -v 3 -define G \
    -linear_group -PGGL 4 8 -wedge -end -end \n    -with G -do \n    -group_theoretic_activities -surface_recognize -q 8 \n    -by_coefficients "1,6,1,8,1,11,1,13,1,19" -end -end
```

identifies the surface (cf. Table 19)

\[X_0^2X_3 + X_1^2X_2 + X_1X_2^2 + X_0X_3^2 + X_1X_2X_3 = 0\] (4)

in the classification of surfaces over the field \(\mathbb{F}_8\). This means that an isomorphism from the given surface to the surface in the list is computed. Also, the generators of the automorphism group of the given surface are computed, using the known generators for the automorphism group of the surface in the classification. For instance, executing the command above produces the isomorphism

\[
\begin{bmatrix}
1 & 4 & 4 & 0 \\
6 & 0 & 0 & 0 \\
6 & 2 & 0 & 1 \\
7 & 0 & 4 & 0 \\
\end{bmatrix}
\]

(5)

Orbiter can compute isomorphism between two given surfaces. Both surfaces must have 27 lines. For instance, the command

```
orbiter.out -v 3 -linear_group -define G \
    -PGGL 4 8 -wedge -end -end \n    -with G -do \n    -group_theoretic_activities -surface_isomorphism_testing \n    -q 8 -by_coefficients \n    "5,5,5,8,5,9,5,10,5,11,5,12,4,14,4,15,1,18,1,19" -end \n    -q 8 -by_coefficients "1,6,1,8,1,11,1,13,1,19" -end
```

computes an isomorphism between the two \(\mathbb{F}_8\)-surfaces

\[
0 = \alpha^3X_0^2X_2 + \alpha^3X_1^2X_2 + \alpha^3X_1^2X_3 + \alpha^3X_0X_1X_2^2 + \alpha^3X_1X_2^2 + \alpha^3X_2X_3 + \alpha^2X_1X_2^2 + \alpha^2X_2X_3 + X_0X_2X_3 + X_1X_2X_3,
\]

\[
0 = X_0^2X_3 + X_1^2X_2 + X_1X_2^2 + X_0X_3^2 + X_1X_2X_3.
\]

The isomorphism is given as a collineation:
In here, the numerical representation of elements of $\mathbb{F}_8$ as integers in the interval $[0, 7]$ is used. The exponent of the Frobenius automorphism is listed as a subscript.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-jacobi</td>
<td>$a$ $p$</td>
<td>Computes the Jacobi symbol $\left( \frac{a}{p} \right)$</td>
</tr>
<tr>
<td>-sift_smooth</td>
<td>$a$ $n$ primes</td>
<td>Computes all smooth numbers in the interval $[a, a + n - 1]$. Smooth means that they factor completely over the list of primes given.</td>
</tr>
<tr>
<td>-random</td>
<td>$n$ $fname$</td>
<td>Creates $n$ random numbers and writes them to the csv file $fname$</td>
</tr>
<tr>
<td>-random_last</td>
<td>$n$</td>
<td>Creates $n$ random numbers prints the last one</td>
</tr>
<tr>
<td>-affine_sequence</td>
<td>$a$ $b$ $p$</td>
<td>Splits the interval $[0, p - 1]$ into affine sequences of the form $x_{n+1} = ax_n + b \mod p$</td>
</tr>
</tbody>
</table>

Table 45: Number Theoretic Commands

38 Number Theory

In Table 45, some number theoretic commands are shown. For instance,

```
orbiter.out -v 2 -inverse_mod 18059241 58014043
```

computes the inverse of 18059241 modulo 58014043.

The Legendre symbol tells us if a number $a$ is a square modulo an odd prime $p$. By definition,

$$\left( \frac{a}{p} \right) = \begin{cases} 
1 & \text{if there exists } r \text{ s.t. } r^2 \equiv a \mod p \\
-1 & \text{if there does not exist } r \text{ s.t. } r^2 \equiv a \mod p \\
0 & \text{if } p \text{ divides } a.
\end{cases}$$

The Jacobi symbol generalizes the Legendre symbol to allow non-prime bottom arguments. By definition,

$$\left( \frac{a}{b} \right) = \prod_{i=1}^{k} \left( \frac{a}{r_i} \right)^{e_i},$$

where

$$b = \prod_{i=1}^{k} r_i^{e_i}$$

is the prime factorization of $b$ with pairwise distinct primes $r_i$. The Jacobi symbol agrees with the Legendre symbol whenever the bottom argument $b$ is an odd prime. Because there is no ambiguity, the same notation is used for the Jacobi symbol as for the Legendre symbol. Orbiter can compute Jacobi symbols. For instance, the command

```
orbiter.out -v 5 -jacobi 2221 7817
```
computes the Jacobi symbol \[ \left( \frac{2221}{7817} \right). \]

In the Jacobi symbol, the denominator \( p \) has to be a positive odd integer. This command creates the file `jacobi_2221_7817.tex` which contains a detailed step-by-step description of the computation. The steps correspond to the basic rules for computing the Jacobi symbol and can be found in many textbooks. After reformatting, the description looks like this:

\[
\begin{align*}
\left( \frac{2221}{7817} \right) &= \left( \frac{7817}{2221} \right) \cdot (-1)^{\frac{2221-1}{2} \cdot \frac{7817-1}{2}} = \left( \frac{7817}{2221} \right) = \left( \frac{1154}{2221} \right) = \left( \frac{-2}{2221} \right) \cdot \left( \frac{577}{2221} \right) \\
&= (-1)^{\frac{2221^2-1}{8}} \cdot \left( \frac{577}{2221} \right) = (-1) \cdot \left( \frac{577}{2221} \right) = (-1) \cdot \left( \frac{2221}{577} \right) \cdot (-1)^{\frac{577^2-1}{8}} \cdot \left( \frac{2221^{2-1}}{577^{2-1}} \right) \\
&= (-1) \cdot \left( \frac{2221}{577} \right) = (-1) \cdot \left( \frac{490}{577} \right) = (-1) \cdot \left( \frac{2}{577} \right) \cdot \left( \frac{245}{577} \right) = (-1) \cdot (-1)^{\frac{577^2-1}{8}} \cdot \left( \frac{245}{577} \right) \\
&= (-1) \cdot \left( \frac{245}{577} \right) \cdot (-1)^{\frac{245-1}{2} \cdot \frac{577-1}{2}} = (-1) \cdot \left( \frac{245}{577} \right) = (-1) \cdot \left( \frac{71}{87} \right) = (-1) \cdot \left( \frac{87}{71} \right) \cdot (-1)^{\frac{71^2-1}{2} \cdot \frac{87^2-1}{2}} \\
&= \left( \frac{87}{71} \right) = \left( \frac{16}{71} \right) = \left( \frac{2}{71} \right)^4 \cdot \left( \frac{1}{71} \right) = \left( (-1)^{\frac{71^2-1}{8}} \right)^4 \cdot \left( \frac{1}{71} \right) = \left( \frac{1}{71} \right) = 1.
\end{align*}
\]

The answer 1 tells us that 2221 is a square modulo 7817. Because 7817 is prime, the Jacobi symbol and the Legendre symbol agree on this input pair. We can use the `square_root_mod` command from Section 7 to compute a square root of 2221 modulo 7817 and verify this fact. The command

\[
\text{orbiter.out -v 2 -square_root_mod 2221 7817}
\]

yields that 7634 is a square root. Indeed,

\[ 7634^2 \equiv 2221 \mod 7817. \]
### Table 46: Cryptographic Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-solovay_strassen</td>
<td>a n</td>
<td>Performs $n$ Solovay / Strassen tests on the number $a$</td>
</tr>
<tr>
<td>-miller_rabin</td>
<td>a n</td>
<td>Performs $n$ Miller / Rabin tests on the number $a$</td>
</tr>
<tr>
<td>-fermat</td>
<td>a n</td>
<td>Performs $n$ Fermat tests on the number $a$</td>
</tr>
<tr>
<td>-find_pseudoprime</td>
<td>$a_{n_1} n_2 n_3$</td>
<td>Computes a pseudoprime which survives $n_1$ Fermat tests, $n_2$ Miller Rabin tests, $n_3$ Solovay Strassen tests</td>
</tr>
<tr>
<td>-find_strong_</td>
<td>$a n_1 n_2$</td>
<td>Computes a pseudoprime which survives $n_1$ Fermat tests and $n_2$ Miller Rabin tests</td>
</tr>
<tr>
<td>-find_strong_</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-RSA_encrypt_text</td>
<td>$d n \text{ text}$</td>
<td>Using blocks of $b$ letters at a time, encrypt “text” using RSA with exponent $d$ modulo $n$</td>
</tr>
<tr>
<td>-RSA</td>
<td>$d n \text{ list-of-integers}$</td>
<td>encrypt the given sequence of integers using RSA with exponent $d$ modulo $n$</td>
</tr>
</tbody>
</table>

### 39 Cryptography

In Table 46, some cryptographic commands are shown. In Table 46, some cryptographic commands depending on a finite field are shown. We assume that the field $\mathbb{F}_q$ has been defined. For instance,

```bash
orbiter.out -v 2 -finite_field_activity -q 11 -EC_add 1 3 "1,4" "1,4" -end
```

adds the point $(1,4)$ on the curve $y^2 = x^3 + x + 3 \mod 11$ to itself. The command

```bash
orbiter.out -v 2 -finite_field_activity -q 11 -EC_cyclic_subgroup 1 3 "1,4" -end
```

computes the cyclic subgroup generated by the point $(1,4)$ on the curve $y^2 = x^3 + x + 3 \mod 11$. The command

```bash
orbiter.out -v 2 -finite_field_activity -q 199 -EC_points 5 7 -end
```

computes all points on the curve $y^2 = x^3 + 5x + 7 \mod 199$. The command

```bash
orbiter.out -v 6 -seed 17 -finite_field_activity -q 199 \ -EC_Koblitz_encoding 5 7 67 "147,164" "DEADBEEF" \ -end
```

167
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-EC_add</td>
<td>a b i₁ i₂</td>
<td>On the elliptic curve $y^2 \equiv x^3 + ax + b$ in $\mathbb{F}_q$, add the points with indices $i₁$ and $i₂$, each given as a pair $x, y$.</td>
</tr>
<tr>
<td>-EC_points</td>
<td>a b</td>
<td>Computes all points of the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-EC_multiple_of</td>
<td>a b pt n</td>
<td>Computes the $n$ fold multiple of the given point $pt$ on the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-EC_cyclic_subgroup</td>
<td>a b pt</td>
<td>Computes the cyclic subgroup generated by the given point $pt$ on the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-EC_Koblitz_encoding</td>
<td>a b s pt plain</td>
<td>Computes the Koblitz encoding of “plain” (all caps) on the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$ using the base point $pt$ and the secret exponent $s$.</td>
</tr>
<tr>
<td>-EC_bsgs</td>
<td>a b pt n cipher</td>
<td>Prepare the baby-step giant-step tables for the ciphertext “cipher” on the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$ using the base point $pt$ of order $n$.</td>
</tr>
<tr>
<td>-EC_bsgs_decode</td>
<td>a b pt n cipher round-keys</td>
<td>Decodes the ciphertext “cipher” on the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$ using the base point $pt$ of order $n$ and the round keys “keys”.</td>
</tr>
<tr>
<td>-EC_discrete_log</td>
<td>a b pt base-pt</td>
<td>Computes the elliptic curve discrete log analogue of $pt$ with respect to $base-pt$ on the elliptic curve $y^2 \equiv x^3 + ax + b$ over $\mathbb{F}_q$.</td>
</tr>
<tr>
<td>-NTRU_encrypt</td>
<td>N p H R M</td>
<td>NTRU encryption for the message $M(X)$ using the public key $H(X)$ and one-time-key $R(X)$.</td>
</tr>
<tr>
<td>-polynomial_center_lift</td>
<td>A(X)</td>
<td>Compute the center lift mod $q$ for the coefficients of $A$.</td>
</tr>
<tr>
<td>-polynomial_reduce_mod_p</td>
<td>p A(X)</td>
<td>Reduce the coefficients of the polynomial $A$ modulo $p$.</td>
</tr>
</tbody>
</table>

Table 47: Finite Field Activities related to Cryptography
encode the message “DEADBEEF” on the curve $y^2 = x^3 + 5x + 7 \mod 199$ using the base point $(147, 164)$ and the secret key 67. The $i$th input character is encoded as two points $(R_i, T_i)$ on the curve using the Elgamal scheme. A random round key is generated for each plaintext symbol. As seen in this example, the `-seed` command can be used to seed the random number generator with an arbitrary integer (here 17). The command

```
orbiter.out -v 2 -finite_field_activity -q 199 -EC_bsgs 5 7 "147,164" 212 \ 
"172,158,45,195,50,22,10,103,55,33,50,22,145,105,31,74,73,155,67,60,25,6" \ 
-end
```

performs a baby-step-giant-step brute force attack on the ciphertext sequence

$$ R_i = (172, 158), (45, 195), (50, 22), (10, 103), (55, 33), $$

$$ (50, 22), (145, 105), (31, 74), (73, 155), (67, 60), (25, 6), $$

using the base point $(147, 164)$ on the curve $y^2 = x^3 + 5x + 7 \mod 199$, assuming a group order of 212. The command

```
orbiter.out -v 2 -finite_field_activity -q 199 -EC_bsgs_decode 5 7 "129,176" 212 \ 
"127,188,51,141,85,29,106,90,41,105,179,71,171,2,16,197,183,72,27,129,37,10" \ 
"50,179,169,13,153,169,115,116,188,110,176" \ 
-end
```

performs a decoding of the ciphertext sequence

$$ T_i = (127, 188), (51, 141), (85, 29), (106, 90), (41, 105), (179, 71), $$

$$ (171, 2), (16, 197), (183, 72), (27, 129), (37, 10), $$

assuming round keys

$$ k_i = 50, 179, 169, 13, 153, 169, 115, 116, 188, 110, 176, $$

using the base point $(147, 164)$ on the curve $y^2 = x^3 + 5x + 7 \mod 199$, and assuming a group order of 212.

The next sequence of examples discusses the NTRU cryptosystem (cf. Example 7.53 in [25]). We are using some advanced makefile features such as variables. In the example, we choose the parameters of the cryptosystem to be $(N, p, q, d) = (7, 41, 3, 2)$. Orbiter uses the following convention for polynomials over a finite field $\mathbb{F}_q$: The coefficients of $A(X) = a_0 + a_1X + \cdots + a_dX^d$ are listed as a sequence, starting with the constant term and ending with the leading coefficient. The cryptosystem requires coefficients $a_i$ in the range $-\frac{p}{2} \leq a_i \leq \frac{p}{2}$. So, in an extension to the conventions for field elements in $\mathbb{F}_q$, Orbiter allows negative coefficients as well. The assumption is that $q$ is prime and negative coefficients are considered modulo $q$. In the example, Alice picks the private polynomials $f(x) = x^6 - x^4 + x^3 + x^2 - 1$ (with $d + 1$ coefficients equal to plus one and $d$ coefficients equal to minus one) and $g(x) = x^6 + x^4 - x^2 - x$ with $d$ coefficients plus one and $d$ coefficients minus one. We also need the polynomial $x^N - 1$. The makefile commands
NTRU_N=7
NTRU_P=3
NTRU_Q=41
NTRU_D=2
NTRUE_XN1="-1,0,0,0,0,0,0,1,"
# D + 1 plus ones and D minus ones
ALICE_PRIVATE_F="-1,0,1,1,-1,0,1"
# D plus ones and D minus ones
ALICE_PRIVATE_G="0,-1,-1,0,1,0,1,1"

are used to set up the appropriate variables according to these choices. The purpose of this is so that we never have to write out these polynomials or parameter value more than once. The use of makefile variables makes the code immune to typographical errors, since it reduces duplication of code. Also, it makes it easy for us to change the values in the example to different parameters and polynomials. The makefile variable substitution process will allow us to access these values easily through the variable name, using the dollar-parenthesis syntax. So, for instance, $(NTRUE_XN1)$ represents the string "-1,0,0,0,0,0,0,1," which in turn represents the polynomial \(x^7 - 1\). Keep in mind that our makefile also has a variable $(ORBITER_PATH)$ which contains the path to the orbiter binary. As discussed in Section 3, the orbiter path is put right in front of the orbiter binary, so that our makefile and our shell can find the orbiter binary. Of course, there is no need for this if orbiter is installed in a system directory, where it can be found by the shell.

Regarding the NTRU set-up, Alice needs to compute her private keys \(F_p(x)\) and \(F_q(x)\). These two polynomials are defined as follows:

1. \(F_p(x)\) is the inverse of \(f(x)\) in \(\mathbb{F}_p[x]/(x^n - 1)\),
2. \(F_q(x)\) the inverse of \(f(x)\) in \(\mathbb{F}_q[x]/(x^n - 1)\).

To this end, we can use the extended_gcd_for_polynomials command from Table 45. The following two makefile commands do the job:

NTRU_Alice1:
orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \ 
-extended_gcd_for_polynomials $(NTRUE_XN1) $(ALICE_PRIVATE_F) -end

#F_q(x) = 8X^6 + 26X^5 + 31X^4 + 21X^3 + 40X^2 + 2X + 37
ALICE_PRIVATE_FQ="37,2,40,21,31,26,8"

NTRU_Alice2:
orbiter.out -v 2 -finite_field_activity -q $(NTRU_P) \ 
-extended_gcd_for_polynomials $(NTRUE_XN1) $(ALICE_PRIVATE_F) -end

#F_p(x) = X^6 + 2X^5 + X^3 + X^2 + X + 1
ALICE_PRIVATE_FP="1,1,1,0,2,1"
Note how the resulting polynomials (indicated as comments by means of the # symbol) are again encoded as makefile variables. This simplifies the remaining makefile commands. Note also that there is a chance that the polynomial $f(x)$ does not have an inverse in either $\mathbb{F}_p[x]$ or in $\mathbb{F}_q[x]$. In that case, Alice simply chooses a different polynomial $f(x)$ and tries again. Alice can now compute her public key:

```
NTRU_Alice_public_key:
  orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \$
  -polynomial_mult_mod $(ALICE_PRIVATE_F) $(ALICE_PRIVATE_G) $(NTRUE_XN1) \$
  -end

# C(X)=20X^{6} + 40X^{5} + 2X^{4} + 38X^{3} + 8X^{2} + 26X + 30
ALICE_PUBLIC_KEY="30,26,8,38,2,40,20"
```

The public key is assigned to the makefile variable ALICE_PUBLIC_KEY. Now, Bob chooses his message to Alice and his one-time-key. The message must be the center lift of a polynomial in $\mathbb{F}_p[x]$. The round-key must have exactly $d$ coefficients one and $d$ coefficients $-1$ (rest zeroes).

```
BOB_MESSAGE="1,-1,1,1,0,-1"
BOB_ONE_TIME_KEY="-1,1,0,0,0,-1,1"
```

The encryption proceeds using the NTRU_encrypt command, and the result is stored in the makefile variable BOB_ENCRYPT:

```
NTRU_encrypt:
  orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \$
  -NTRU_encrypt $(NTRU_N) $(NTRU_P) $(ALICE_PUBLIC_KEY) \$
  $(BOB_ONE_TIME_KEY) $(BOB_MESSAGE) -end

# E(X) = 31X^{6} + 19X^{5} + 4X^{4} + 2X^{3} + 40X^{2} + 3X + 25
BOB_ENCRYPT = "25,3,40,2,4,19,31"
```

Decryption is done in five steps and ultimately yield back Bob's message to Alice:

```
NTRU_decrypt1:
  orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \$
  -polynomial_mult_mod $(ALICE_PRIVATE_F) $(BOB_ENCRYPT) $(NTRUE_XN1) \$
  -end

# C(X)=X^{6} + 10X^{5} + 33X^{4} + 40X^{3} + 40X^{2} + X + 40
ALICE_C1="40,1,40,40,33,10,1"
```

```
NTRU_decrypt2:
  orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \$
  -polynomial_center_lift $(ALICE_C1) -end
```

NTRU_decrypt2:
  orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \$
  -polynomial_center_lift $(ALICE_C1) -end
#A(X) = X^6 + 10X^5 - 8X^4 - X^3 - X^2 + X - 1
ALICE_C2 = "-1,1,-1,-1,-8,10,1"

NTRU_decrypt3:
    orbiter.out -v 2 -finite_field_activity -q $(NTRU_P) \
    polynomial_reduce_mod_p $(ALICE_C2) -end

#A(X) = X^6 + X^5 + X^4 + 2X^3 + 2X^2 + X + 2
ALICE_C3 = "2,1,2,2,1,1,1"

NTRU_decrypt4:
    orbiter.out -v 2 -finite_field_activity -q $(NTRU_Q) \
    -polynomial_mult_mod $(ALICE_PRIVATE_FP) $(ALICE_C3) $(NTRUE_XN1) \
    -end

#C(X) = 2X^5 + X^3 + X^2 + 2X + 1
ALICE_C4 = "1,2,1,1,0,2"

NTRU_decrypt5:
    orbiter.out -v 2 -finite_field_activity -q $(NTRU_P) \
    -polynomial_center_lift $(ALICE_C4) -end

#A(X) = - X^5 + X^3 + X^2 - X + 1
#plaintext as in $(BOB_MESSAGE)
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-BCH</td>
<td>(n q t)</td>
<td>Creates the BCH-code of length (n) over the field (\mathbb{F}_q) with designed distance (t)</td>
</tr>
<tr>
<td>-Hamming_graph</td>
<td>(n q)</td>
<td>Creates the distance matrix of the Hamming graph (H(n,q)). The vertices are the elements of (\mathbb{F}_q^n), and the (i,j)-entry is the distance between the vectors whose affine ranks are (i) and (j), respectively. The matrix is written as csv-file.</td>
</tr>
<tr>
<td>-general_code_binary</td>
<td>(n R)</td>
<td>Creates the binary code of length (n) containing the elements corresponding to the integers in the list (R) under the binary representation.</td>
</tr>
<tr>
<td>-linear_code_through_basis</td>
<td>(n R)</td>
<td>Creates the binary linear code of length (n) generated by the elements corresponding to the integers in the list (R) under the binary representation.</td>
</tr>
<tr>
<td>-long_code</td>
<td>(n k r_1 ... r_k)</td>
<td>Creates the binary code of length (n) and dimension (k) whose generators are given as (r_1, \ldots, r_k).</td>
</tr>
</tbody>
</table>

Table 48: Coding Theoretic Commands

40 Coding Theory

In Table 48, some coding theoretic commands of Orbiter are shown. For instance, the command

```
orbiter.out -BCH 15 2 3
```
Figure 21: The color-coded distance matrix of the Hamming graph $H(4, 2)$ creates a binary BCH-code of length 15 with minimum distance at least 3. The generator matrix produced by this command is

$$
\begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}
$$

The commands

`orbiter.out -Hamming_graph 4 2`
`orbiter.out -draw_matrix Hamming_n4_q2.csv 20`

create the csv-file `Hamming_n4_q2.csv` and produce the bitmap file `Hamming_n4_q2_draw.bmp` shown in Figure 21. Table 49 lists Orbiter activities for finite fields related to coding theory.

The classification problem of optimal codes in coding theory is the problem of determining
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-weight_enumerator</td>
<td>m n L</td>
<td>Compute the complete weight enumerator of the linear code generated by the $m \times n$ matrix $L$.</td>
</tr>
</tbody>
</table>

Table 49: Finite Field Activities related to Coding Theory

the equivalence classes of codes for a given set of values of $n$ and $k$ and $q$ with a lower bound on $d$. Orbiter can be used for solving this problem for small instances.

Orbiter can be used to classify linear codes with given redundancy and bounded minimum distance. The redundancy of a linear $[n, k]$ code is the parameter $r = n - k$. Codes with redundancy $r$ can be identified with subsets of $\text{PG}(r - 1, q)$. Under this correspondence, a code with minimum distance at least $d$ corresponds to a subset such that any $d - 1$ elements are independent. We use the notation $\Lambda_{r-1,s}(q)$ to denote the poset of subsets of $\text{PG}(r - 1, q)$ for which any $d - 1$-subset (if any) is independent. Under the correspondence, the action of $\text{PGL}(r, q)$ on $\Lambda_{r-1,s}(q)$ corresponds to the orbits of equivalent linear codes. For this reason, we are interested in determining the orbits of $\text{PGL}(r, q)$ on $\Lambda_{r-1,s}(q)$. An orbit of size $n$ represents an isometry class of $[n, n - r, d; q]$ codes with $d \geq s + 1$. The projective stabilizer of the subset is the automorphism group of the code. The Orbiter command

```
orbiter.out -v 6 \
  -orbiter_path $(ORBITER_PATH) \
  -define G \
  -linear_group -PGL 4 2 -end -end \
  -with G -do \
  -group_theoretic_activities \ 
  -poset_classification_control -problem_label codes_8_4_4 \ 
  -draw_poset \ 
  -draw_options -embedded -rad 250 -line_width 1.0 -spanning_tree -end \ 
  -report \ 
  -end \ 
  -linear_codes 3 8
```

```
pdflatex codes_8_4_4_poset.tex
open codes_8_4_4_poset.pdf
```

can be used to classify linear codes with redundancy 4 and minimum distance at least 4. The extremal code that satisfies these conditions is the $[8, 4, 4]$ codes over $\mathbb{F}_2$. The Orbiter program confirms that there is exactly one such code. Orbiter computes the code together with the projective stabilizer. To do so, we first create the group $\text{PGL}(4, 2)$ acting on the poset $\Lambda_{3,3}(2)$. Orbiter then produces the poset of orbits shown in Figure 22. In this diagram, the numbers stand for Orbiter ranks of points in $\text{PG}(3, 2)$. All nodes except for the root node have a number attached to it. The nodes represent subsets. In order to determine the set associated to a node, follow the path from the root node to the node and collect the points
Figure 22: Orbits of PGL(4, 2) on the poset $\Lambda_{3,3}(2)$
according to their labels. The root node represents the empty set. The \([8, 4, 4; 2]\)-code is represented by the set \(\{0, 1, 2, 3, 8, 11, 13, 14\}\). The fact that there is only one node at level 8 in the poset of orbits tells us that the code is unique up to equivalence. Let us look at the code. The elements of the set \(\{0, 1, 2, 3, 8, 11, 13, 14\}\) are points in \(\text{PG}(3, 2)\). We write the coordinate vectors in the columns of a matrix \(H\):

\[
H = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1
\end{bmatrix}.
\]

This matrix is the parity check matrix \(H\) of the code \(C\). This means that the words of the code are the vectors \(c\) such that \(c \cdot H^\top = 0\). Observe that the vectors that we put in the columns of \(H\) all have odd weight. They are in fact the points of the hyperplane \(x + y + z + w = 0\). This shows that the stabilizer of the code which is the stabilizer of the set is equal to \(\text{AGL}(3, 2)\), a group of order 1344.

The Hamming code is generated by the matrix

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}
\]

Using the binary representation of decimal numbers, the rows in the generator matrix correspond to 67, 37, 22, and 15. These decimals can be used to specify the code using the `linear_code_through_basis` command. The sequence

```
orbiter.out -v 2 -linear_code_through_basis 7 "67,37,22,15"
orbiter.out -draw_matrix code_matrix_16_8.csv 10 8 \
    -draw_matrix_partition 2 16 8
```

creates the Hamming code and produces a drawing of the codewords in Hamming space shown in Figure 23.

Some codes are too long for the binary representation to be useful. In this case, it is possible to specify the entries of the generator matrix individually, using the `long_code` command. For instance, the sequence

```
orbiter.out -v 2 -long_code 64 7 \
    -set_builder -loop 0 64 1 -end \ 
    "1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63" \ 
    "2,3,6,7,18,19,22,23,10,11,14,15,26,27,30,31,34,35,38,39,42,43,46,47,50,51,54,55,58,59,62,63" \ 
    "4,6,12,14,36,38,52,54,57,7,13,15,37,39,53,55,20,22,28,30,44,46,47,50,51,54,55,58,59,62,63" \ 
    "8,9,12,13,24,25,28,29,10,11,14,15,26,27,30,31,40,41,44,45,46,47,50,51,42,43,46,47,58,59,62,63" \ 
    "16,18,24,26,48,50,56,58,17,19,25,27,49,51,57,59,20,22,28,30,52,54,60,62,21,23,29,31,53,55,61,63" \ 
    "32,34,48,50,33,35,49,51,36,38,52,54,37,39,53,55,40,42,56,58,41,43,57,59,44,46,60,62,45,47,61,63"
orbiter.out -draw_matrix long_code_gemma_n64_k7.csv 25 8 -draw_matrix_partition 3 7 64
```

177
creates the Reed-Muller code of order 6 and produces a diagram of the generator matrix as shown in Figure 24. For the set builder command, see Section 5.
41 Diophantine Systems

Diophantine systems of equations and inequalities arise frequently in Combinatorics. In Table 50, Orbiter commands for creating diophantine systems are shown. In Table 51, Orbiter activities for diophantine systems are shown.

Suppose we want all partitions of an integer \( n \) as

\[
n = a_1 + a_2 + \ldots + a_k, \quad a_1 \geq a_2 \geq \cdots \geq a_k \geq 1,
\]

with \( a_i \in \mathbb{Z}_{>0} \). For \( 1 \leq j \leq n \), let

\[
c_j = \# \{ i \mid a_i = j \}.
\]

The following diophantine equation holds for any partition

\[
\sum_{j=1}^{n} j c_j = n
\]  \( (6) \)

Conversely, any partition is uniquely determined by a solution of this equation. Therefore, counting partitions of \( n \) is the same as counting nonnegative integer solutions of (6). Let \( p_n \) be the number of partitions of \( n \). Suppose we wish to compute \( p_{10} \). In this case, the extended coefficient matrix of the system is

\[
\begin{bmatrix}
10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \ 
\end{bmatrix}
\]

Orbiter creates this diophantine system using the command

```
orbiter.out -v 4 -diophant -label part10 \
   -coefficient_matrix 1 10 "10,9,8,7,6,5,4,3,2,1" \ 
   -RHS "10,10,1" -x_min_global 0 -x_max_global 10
```

It also creates three csv-files: one for the coefficient matrix, one for the RHS information, and one for the bounds on the variables. With these files, it is possible to recreate the diophantine system using the command

```
orbiter.out -v 4 -diophant -label part10 \ 
   -coefficient_matrix_csv part10_coeff_matrix.csv \ 
   -RHS_csv part10_RHS.csv \ 
   -x_bounds_csv part10_x_bounds.csv
```

The command

```
orbiter.out -v 4 -diophant_activity -input_file part10.diophant -solve_mckay
```

solves the system and finds that \( p_{10} = 42 \). The sequence \( p_n \) is recorded under the key A000041 in Sloane’s Handbook of integer sequences [51].
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-label</td>
<td>label</td>
<td>Use the given name as file name.</td>
</tr>
<tr>
<td>-coefficient_matrix</td>
<td>m n list-of-integers</td>
<td>Set the $m \times n$ coefficient matrix.</td>
</tr>
<tr>
<td>-coefficient_matrix_csv</td>
<td>fname</td>
<td>Read the coefficient matrix from the given csv-file.</td>
</tr>
<tr>
<td>-RHS</td>
<td>list-of-integers</td>
<td>$3n$ values: (RHS-low, RHS-high, RHS-type) for each row of the system.</td>
</tr>
<tr>
<td>-RHS_csv</td>
<td>fname</td>
<td>Read the RHS information from the given csv file.</td>
</tr>
<tr>
<td>-RHS_constant</td>
<td>low,high,type</td>
<td>Set the RHS according to low,high,type.</td>
</tr>
<tr>
<td>-x_max_global</td>
<td>a</td>
<td>Set the upper bound for all variables to $a$.</td>
</tr>
<tr>
<td>-x_min_global</td>
<td>a</td>
<td>Set the lower bound for all variables to $a$.</td>
</tr>
<tr>
<td>-x_bounds</td>
<td>list-of-values</td>
<td>Set the lower and upper bounds for all variables.</td>
</tr>
<tr>
<td>-x_bounds_csv</td>
<td>fname</td>
<td>Read the lower and upper bounds for all variables from the given file.</td>
</tr>
<tr>
<td>-has_sum</td>
<td>s</td>
<td>For the sum of the variables to be $s$.</td>
</tr>
<tr>
<td>-maximal_arc</td>
<td>s d secants subset</td>
<td>Create system for a maximal arc of size $s$ and degree $d$ in $\text{PG}(2,q)$. Use the given set of two pencil lines. The subset picks the lines from the given pencils which are external.</td>
</tr>
<tr>
<td>-q</td>
<td>q</td>
<td>Use $\text{PG}(2,q)$ for maximal arcs.</td>
</tr>
<tr>
<td>-override_polynomial</td>
<td>a</td>
<td>Use polynomial numerically coded as $a$ for creating $\mathbb{F}_q$.</td>
</tr>
</tbody>
</table>

Table 50: Orbiter Commands to create Diophantine systems
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-input_file</td>
<td>file</td>
<td>Specify the input file</td>
</tr>
<tr>
<td>-print</td>
<td></td>
<td>Print the system</td>
</tr>
<tr>
<td>-solve_mckay</td>
<td></td>
<td>Solve the system using McKay’s pos-solve</td>
</tr>
<tr>
<td>-solve_standard</td>
<td></td>
<td>Solve the system using the standard solver</td>
</tr>
<tr>
<td>-draw</td>
<td></td>
<td>Produce a drawing of the coefficient matrix of the system</td>
</tr>
<tr>
<td>-draw_as_bitmap</td>
<td>w b</td>
<td>Produce a bitmap drawing of the coefficient matrix of the system, using boxes of w pixels with. Set the color bit-depth to b (b = 8 or b = 24). The output is a bmp-file.</td>
</tr>
<tr>
<td>-perform_column_reductions</td>
<td></td>
<td>Eliminate variables which must be zero and write a reduced system</td>
</tr>
<tr>
<td>-test_single_equation</td>
<td></td>
<td>For each row of the system, compute the number of solutions of the system restricted to the nonzero coefficients.</td>
</tr>
<tr>
<td>-project_to_single_equation_and_solve</td>
<td>i j</td>
<td>Solve the system assuming the jth solution to the restricted system consisting of the ith row.</td>
</tr>
<tr>
<td>-project_to_two_equations_and_solve</td>
<td>i j r m</td>
<td>Solve the system assuming any solution to the restricted system consisting of the ith and the j-th row whose number is congruent to r mod m.</td>
</tr>
</tbody>
</table>

Table 51: Orbiter activities for Diophantine systems
42 Combinatorial Linear Spaces

A linear space is a pair \((S, \mathcal{L})\) where \(S\) is a set and \(\mathcal{L}\) is a set of subsets of \(S\) such that each set \(L \in \mathcal{L}\) satisfies \(|L| \geq 2\) and moreover for any two \(a, b \in S\) there is exactly one element \(L \in \mathcal{L}\) such that both \(a\) and \(b\) belong to \(L\). The usual notions of isomorphism and automorphism apply. For finite linear spaces, a first combinatorial property is the number \(a_i\) which counts the number of sets \(L \in \mathcal{L}\) of size \(i\). The vector \((a_2, \ldots, a_n)\) is the line type of \((S, \mathcal{L})\). The equation

\[
\binom{n}{2} = \sum_{j=2}^{n} a_j \binom{j}{2}
\]  

is satisfied. The equation can be used to generate all possible line types of a putative linear space. Here is an example. For \(|S| = 6\), \((7)\) becomes

\[
x_0 \binom{6}{2} + x_1 \binom{5}{2} + x_2 \binom{4}{2} + x_3 \binom{3}{2} + x_4 \binom{2}{2} = \binom{6}{2}.
\]

Here, \((x_0, x_1, \ldots, x_4)\) is the line type of a putative linear space on 6 points. That is, \(x_i = a_{6-i}\) is the number of lines of size \(6-i\). The extended coefficient matrix of the system is

\[
\begin{bmatrix}
15 & 10 & 6 & 3 & 1 \\
15
\end{bmatrix}
\]

The Orbiter command

```
orbiter.out -v 4 -diophant -label linsp6 \ 
-coefficient_matrix 1 5 "15,10,6,3,1" -RHS "15,15,1" \ 
-x_min_global 0 -x_max_global 15
```

creates this system and stores it in the file `linsp6.diophant` using the specified name. The command

```
orbiter.out -v 3 -diophant_activity -input_file linsp6.diophant -solve_mckay
```

solves the system using McKay's program possolve [41]. The program finds 15 solutions, written to the file `linsp6.sol`.

Let us consider a problem from [8]. Suppose we are interested in linear spaces on 30 points with line type \((7, 5^{27}, 4^{24})\). This notation means that we assume one 7-lines, 27 5-lines and 24 4-lines. The type of a point \(P\) is the set of integers

\[p_j = \#j\text{-lines though } P.\]

We are trying to precompute the matrix of point types

\[
(p_{ij})
\]

where \(j = 7, 5, 4\) and \(i\) belongs to an index set of all possible point types. Fixing a point \(P\), counting points \(Q \neq P\) collinear with \(P\) yields

\[
6p_7 + 4p_5 + 3p_4 = 29, \quad p_7 \leq 1, \quad p_5 \leq 27, \quad p_4 \leq 24.
\]

Using the Orbiter commands
**orbiter.out** -v 4 -diophant -label linsp30_pt_types \
-coefficient_matrix 1 3 "6,4,3" -RHS "29,29,1" -x_bounds "0,1,0,27,0,24" 
**orbiter.out** -v 4 -diophant_activity -input_file \
 linsp30_pt_types.diophant -solve_mckay

we determine the possibilities

\[
(p_7, p_5, p_4) = \begin{cases} 
1 & 5 & 1 \\
1 & 2 & 5 \\
0 & 5 & 3 \\
0 & 2 & 7 
\end{cases}
\]

The rows in this matrix are called the point types \((i = 0, 1, 2, 3)\). Let \(b_i\) be the number of points of type \(i\). By counting points, incident (point,line) pairs by \(j\)-lines and pairs of intersecting \(j\)-lines, we arrive at the following system:

\[
\begin{align*}
b_0 + b_1 + b_2 + b_3 &= 30 \\
b_0 + b_1 &= 7 \\
5b_0 + 2b_1 + 5b_2 + 2b_3 &= 135 = 27 \cdot 5 \\
b_0 + 5b_1 + 3b_2 + 7b_3 &= 96 = 24 \cdot 4 \\
10b_0 + b_1 + 10b_2 + b_3 &\leq 351 = \binom{27}{2} \\
10b_1 + 3b_2 + 21b_3 &\leq 276 = \binom{24}{2}
\end{align*}
\]

Using the Orbiter commands

**orbiter.out** -v 4 -diophant -label linsp30_pt_distribution \
-coefficient_matrix 6 4 "1,1,1,1,1,1,0,0,0,5,2,5,2,1,5,3,7,10,1,10,1,0,10,3,21" \
-RHS "30,30,1,7,7,1,135,135,1,96,96,1,0,351,2,0,276,2" \
-x_min_global 0 -x_max_global 30 
**orbiter.out** -v 4 -diophant_activity -input_file \
 linsp30_pt_distribution.diophant -solve_mckay

we determine the possibilities

\[
(b_0, b_1, b_2, b_3) = \begin{cases} 
2 & 5 & 23 & 0 \\
3 & 4 & 22 & 1 \\
4 & 3 & 21 & 2 \\
5 & 2 & 20 & 3 \\
6 & 1 & 19 & 4 \\
7 & 0 & 18 & 5 
\end{cases}
\]
43 Design Theory

We use the convention of design theory that the incidence matrix of a design has rows indexed by points and columns indexed by blocks (also called lines). A decomposition is a partition of the points and blocks of the geometry such that each class consists either exclusively of points or exclusively of blocks.

A decomposition is point-tactical if for all points, the number of incident lines in the \( j \)th block class depends only on the class of the point. If the point belongs to class \( i \), this number is denoted as \( a_{ij} \). A decomposition is block-tactical if for all blocks, the number of incident points in the \( i \)th point class depends only on the class of the block. If the block belongs to class \( j \), this number is denoted as \( b_{ij} \).

A projective plane of order \( n \) is a design with \( n^2 + n + 1 \) points and equally many blocks (also called lines), each of size \( n + 1 \) such that any two points lie in exactly one block and any two blocks have exactly one point in common. Projective planes are known to exist for all \( n = q \) which are a power of a prime. This follows from a construction which utilizes the projective geometry \( \text{PG}(2,q) \). Points are the one-dimensional subspaces of \( \mathbb{F}_q^3 \), blocks are the two-dimensional subspaces of \( \mathbb{F}_q^3 \), and incidence is natural (inclusion of subspaces). The automorphism group of this design is the collineation group of the projective space. Projective planes other than these exist, though none are known when \( n \) is not a prime power. The number of lines through a point equals the number of points on a line. The fact that these numbers exist imply that there is a tactical decomposition. Namely, the trivial decomposition with two classes, one containing all points and one containing all lines. The structure constants of the decomposition are the numbers just described.

The command

\texttt{orbiter.out -v 8 -create_design -q 3 -family PG\_2\_q -end}

creates the design \( \text{PG}(2,3) \) and its automorphism group:

\begin{verbatim}
We have created the following design:

\{19, 79, 126, 219, 256, 284, 371, 392, 465, 541, 619, 627, 653\}

The stabilizer is generated by:
Strong generators for a group of order 5616:

\begin{bmatrix}
2 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 1
\end{bmatrix},
\end{verbatim}
The blocks of the design are encoded in the lexicographic ordering of $k$-subsets (here $k = 4$). The program also displays the tactical decomposition schemes of the design, which are

\[
\begin{array}{c|c|c}
\rightarrow & 13_1 & \downarrow \\
13_0 & 4 & 13_0 \\
\end{array}
\]

One way to construct new designs is by assuming a suitable group of symmetries. Let us consider an example. It is possible to construct $t-(v,k,\lambda)$ designs invariant under a permutation group $G$ acting on a set $V$ with $|V| = v$ as follows: Classify the orbits of $G$ on subsets of size $k$ and less. Construct a matrix which describes the relationship between the orbits on $t$-sets and the orbits on $k$-sets. This matrix is often referred to as the Kramer-Mesner matrix (cf. [30]). For each pair of $t$-orbit and $k$-orbit, for instance with representatives $T$ and $K$, say, we count the number of elements in the orbit of $K$ which contain $T$. The rows of the matrix are in correspondence to the $t$-orbits, while the columns are in correspondence to the $k$-orbits. The matrix entry $a_{ij}$ is the number just defined where $T$ is the representative of the $i$-th orbit on $t$-sets, and where $K$ is the representative of the $j$-th orbit on $k$-sets. Let $M_{t,k}(G)$ be the Kramer-Mesner matrix for the group $G \leq \text{Sym}(V)$ defined in this way. The $t-(v,k,\lambda)$ designs invariant under $G$ are in one-to-one correspondence to the solutions of

\[
M_{t,k}(G) \cdot \mathbf{x} = \lambda \mathbf{1},
\]

where $\mathbf{x}$ is a column vector of zeros and ones and $\mathbf{1}$ is the column vector of all ones. The length of $\mathbf{x}$ is the number of $k$-orbits of $G$ on $V$, while the length of $\mathbf{1}$ is the number of $t$-orbits of $G$ on $V$. Any vector $\mathbf{x}$ satisfying the matrix equation corresponds to a design invariant under $G$. Simply take the blocks of the design to be the union of those orbits of $G$ on $k$-subsets whose associated entry in $\mathbf{x}$ is one. We assume the group $\text{PGL}(2,32)$ in the action on points of the projective line $\text{PG}(1,32)$ over the field $\mathbb{F}_{32}$. The parameters of the design are $7-(33,8,10)$, that is, each 7-subset of $\text{PG}(1,32)$ is covered exactly 10 times by the chosen 8-subsets comprising the design. The first orbiter command creates the group $\text{PGL}(2,32)$ and computes the Kramer-Mesner matrix

\[
M_{7,8}(\text{PGL}(2,32)).
\]
The number of 7-orbits is 32. The number of 8-orbits is 97. Correspondingly, the Kramer-Menser matrix has 32 rows and 97 columns. The matrix is stored in the csv-file

```
KM_PGGL_2_32_KM_7_8.csv.
```

The second command produces the graphical representation of the matrix shown in Figure 25 (different colors represent different values of entries in the matrix). The third Orbiter command creates the diophantine system associated with the Kramer mesner matrix just computed. The fourth Orbiter command solves the system and produces the solution vector \( x \) which correspond to the designs. The command performs a complete enumeration of all solutions. Here is the full command sequence:

```
orbiter.out -v 3 -define G \\
  -linear_group -PGGL 2 32 -end -end \\
  -with G -do \\
  -group_theoretic_activities \\
  -poset_classification_control -problem_label KM_PGGL_2_32 -W -depth 8 \\
  -Kramer_Mesner_matrix 7 8 \\
  -draw_poset \\
  -draw_options -embedded -sideways -rad 50 -scale 0.5 \\
  -line_width 0.3 -end \\
  -end \\
  -orbits_on_subsets 8
```

```
orbiter.out -v 2 -draw_matrix KM_PGGL_2_32_KM_7_8.csv 10 8 \\
  -draw_matrix_partition 3 32 97
```

```
pdflatex KM_PGGL_2_32_poset_lvl_8.tex
open KM_PGGL_2_32_poset_lvl_8.pdf
open KM_PGGL_2_32_KM_7_8_draw.bmp
```

```
orbiter.out -v 4 -diophant \\
  -label "KM_PGGL_2_32_KM_7_8_system" \\
  -coefficient_matrix_csv KM_PGGL_2_32_KM_7_8.csv
```

186

Figure 25: Kramer-Mesner matrix \( M_{7,8}(PGL(2,32)) \)
-RHS_constant "10,10,1" \
-x_min_global 0 -x_max_global 1
orbiter.out -v 4 -diophant_activity -input_file \ 
KM_PGGL_2_32_KM_7_8_system.diophant -solve_mckay
44 Tactical Decompositions

Suppose we want to study the projective plane of order 16. So, the geometry is is a linear space with \(16^2 + 16 + 1 = 273\) points and equally many lines. Each point lies on 17 lines and each line contains 17 points. Any two points lie on exactly one line and any two lines intersect in exactly one point. Of course, the linear space could be the desarguesian plane \(PG(2, 16)\), but it could also be any of the other projective planes of order 16. At this point, we are only working with the parameters of the geometry as a linear space, and the isomorphism type of the plane is as yet undecided.

We decide to study maximal arcs of degree 4 in this plane (the degree has to divide the order of the plane). A maximal arc of degree \(d\) is a set of points so that each line intersects in either \(d\) or zero points. A line which intersects in \(d\) points is called a secant. A line which intersects in no point is called an external line. The command

```
orbiter.out -v 4 -maximal_arc_parameters 16 4
```

creates a decomposition stack for the parameters of the arc and writes the file `max_arc_q16_r4.stack`

```
<HTDO type=pt ptanz=2 btanz=2 fuse=simple>
  221 52
  52 17 0
  221 13 4
  1 1
</HTDO>
```

This is a point-tactical decomposition with 2 point-classes and 2 block-classes. The point classes are associated with the rows. The block-classes are associated with the columns. The first row and column indicates the size of the classes. The entries \(a_{ij}\) count the number of blocks in the column class \(j\) that are incident with a given point in the \(i\)th row class. The fuse information at the bottom (1 1) is a partition of the row classes which indicates the ancestor decomposition which was column tactical. The next step is to convert the stack file to a tdo file. The command

```
orbiter.out -v 4 -convert_stack_to_tdo max_arc_q16_r4.stack
```

does that. It creates the file `max_arc_q16_r4.tdo`. It also prints the decomposition stack:

```
lambda_scheme at level 2 :
is 1 x 1
  | 273_{ 1}
  ===============
273_{ 0} |
```

```
row_scheme at level 4 :
is 2 x 2
```

188
Next, we can compute all coarsest column-tactical refinements of the decomposition. To this end, the command

```
orbiter.out -v 4 -tdo_refinement -input_file max_arc_q16_r4.tdo -dual_is_linear_space -end
```

is used. Because the incidence structure is a projective plane, the dual is a linear space also. Hence the option `-dual_is_linear_space` can be used, which is helpful to reduce possibilities. As it turns out, there is exactly one refinement, and it is tactical. The file `max_arc_q16_r4r.tdo` is produced. Note the added letter `r` at the end of the file name (r for refinement). We can use the following command to display the decomposition stack in the file:

```
orbiter.out -v 4 -tdo_print max_arc_q16_r4r.tdo
```

This produces the following output:

```
decomposition 0.1:
lambda_scheme at level 2 :
is 1 x 1
  | 273_{ 1}

273_{ 0} |

row_scheme at level 4 :
is 2 x 2
  | 221_{ 1} 52_{ 2}

52_{ 0} | 17 0
221_{ 3} | 13 4

52_{ 0} | 4 0
297_{ 60} 189

col_scheme at level 4 :
is 2 x 2
  | 221_{ 1} 52_{ 2}

52_{ 0} | 4 0
221_{ 3} | 13 4

```
At the moment, the classes of the partitions are too large to be useful for generation of these objects. It would be helpful to investigate the block derived maximal arc. This would lead to finer partitions with more classes of smaller size. We start with the block tactical decomposition in the following stack file:

```
<HTDO type=bt ptanz=2 btanz=2 fuse=simple>
  221  52
  52   4   0
  221  13  17
1  1
</HTDO>
```

This decomposition is block-tactical (note the `type=bt` entry). The matrix entries are $b_{ij}$ which is the number of points in point class $i$ which lie on a line in block class $j$. The fuse partition is a partition of the block classes. And now for the block derived decomposition.

We use the stack file `max_arc_q16_r4bd.stack`:

```
<HTDO type=bt ptanz=2 btanz=3 fuse=simple>
  221  51  1
  52   4   0   0
  221  13  17  17
3
</HTDO>
```

This time, we split off one line from the class of 52 external lines, to yield two classes (of size 51 and 1, respectively). The block partition is three, indicating that all three block classes need to be fused in order to arrive at the row-tactical ancestor decomposition in the stack. We are using the command

```
orbiter.out -v 4 -convert_stack_to_tdo max_arc_q16_r4bd.stack
```

to convert the stack file into a tdo file. This produces the following output:

```
lambda_scheme at level 2 :
is 1 x 1
  | 273_{ 1}
===
273_{ 0} | 190
```
The command

```
orbiter.out -v 4 -tdo_refinement -input_file max_arc_q16_r4bd.tdo \\
-dual_is_linear_space -end
```

is used to compute the coarsest row-tactical refinements. It turns out that there is exactly one, obtained by splitting the point-class of size 221 into two, one of size 204 and one of size 17. The command

```
orbiter.out -v 4 -tdo_print max_arc_q16_r4bdr.tdo
```

can be used to print this decomposition as

\[
\begin{array}{c|ccc}
0.1 & 221 & 52 & 1 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
52_0 & 17 & 0 & 0 \\
204_3 & 13 & 4 & 0 \\
17_5 & 13 & 3 & 1 \\
\end{array}
\quad \begin{array}{c|ccc}
0.1 & 221 & 52 & 1 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
52_0 & 4 & 0 & 0 \\
221_3 & 13 & 17 & 17 \\
\end{array}
\]

Using the command

```
orbiter.out -v 4 -tdo_refinement -input_file max_arc_q16_r4bdr.tdo \\
-dual_is_linear_space -end
```

we compute the coarsest column tactical refinements yet again. As it turns out, the decomposition does not split because it is already tactical. Another print command

```
orbiter.out -v 4 -tdo_print max_arc_q16_r4bdrr.tdo
```

yields the tactical decomposition

\[
\begin{array}{c|ccc}
0.1.1 & 221 & 52 & 1 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
52_0 & 17 & 0 & 0 \\
204_3 & 13 & 4 & 0 \\
17_5 & 13 & 3 & 1 \\
\end{array}
\quad \begin{array}{c|ccc}
0.1.1 & 221 & 52 & 1 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
52_0 & 4 & 0 & 0 \\
204_3 & 12 & 16 & 0 \\
17_5 & 1 & 1 & 17 \\
\end{array}
\]
How do we read this decomposition? We see that there is one external line of 17 points. We may call this the line at infinity. Through each of these 17 points at infinity, there are 16 further lines, coming in two types: Always 3 lines have the property that they are external, and the remaining 13 lines are secants. Taken together, all 51 external lines (distinct from the very first line) arises in this way and cover 204 points. Likewise, all 221 secants arise from the second type of lines. Each of the secants has the following type with respect to points: it intersects 4 points of the maximal arc, 12 points from the set of 204, and exactly one point at infinity.

Table 52 lists the Orbiter commands for TDO-refinement.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lambda3</td>
<td>$\lambda_3, s$</td>
<td>Refine as 3-design with $\lambda_3$ and with block-size $s$.</td>
</tr>
<tr>
<td>-solution</td>
<td>$i, \text{fname}$</td>
<td>Use solutions to system $i$ from file \text{fname}.</td>
</tr>
<tr>
<td>-range</td>
<td>$f, l$</td>
<td>Refine cases $i$ with $f \leq i &lt; f + l$ only.</td>
</tr>
<tr>
<td>-select</td>
<td>label</td>
<td>Select the case for refinement by label.</td>
</tr>
<tr>
<td>-o1</td>
<td>$s$</td>
<td>Omit $s$ variables from the first refinement system.</td>
</tr>
<tr>
<td>-o2</td>
<td>$s$</td>
<td>Omit $s$ variables from the second refinement system.</td>
</tr>
<tr>
<td>-D1_upper_bound_x0</td>
<td>$b$</td>
<td>Add the bound $x_0 \leq b$ in the first refinement.</td>
</tr>
<tr>
<td>-reverse</td>
<td></td>
<td>Sort the distributions in reverse order.</td>
</tr>
<tr>
<td>-reverse_inverse</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-nopacking</td>
<td></td>
<td>Do not use packing inequalities.</td>
</tr>
<tr>
<td>-dual_is_linear_space</td>
<td></td>
<td>Assume that the dual incidence structure is a linear space also. This is valid for projective planes, for instance.</td>
</tr>
<tr>
<td>-geometric_test</td>
<td></td>
<td>Subject the distributions to the geometric test.</td>
</tr>
<tr>
<td>-once</td>
<td></td>
<td>Find at most one refinement in each case. This can be used to test which cases can be refined.</td>
</tr>
<tr>
<td>-mckay</td>
<td></td>
<td>Use McKay’s solver instead (by default, a lexicographic solver is used).</td>
</tr>
<tr>
<td>-input_file</td>
<td>\text{fname}</td>
<td>Specify the input TDO-file for refinement.</td>
</tr>
</tbody>
</table>

Table 52: TDO refinement options
45 Spreads

A $t$-spread of $\text{PG}(n, q)$ is a set of disjoint $\text{PG}(t, q)$ that cover all of $\text{PG}(n, q)$ pointwise. $t$-spreads in $\text{PG}(n, q)$ exist if $t + 1$ divides $n + 1$. The reason is the existence of the Desarguesian spread (also called the regular spread). The Desarguesian spread is created from $\text{PG}(m - 1, Q)$ where $Q = q^s$ for some integer $s$. The spread elements are the subspaces which arise by considering the elements of $\text{PG}(m - 1, Q)$ as vector spaces over $\mathbb{F}_q$. As such, they are rank $s$ subspaces in $\text{PG}(n - 1, q)$. So, with $t = s - 1$, we have a $t$-spread in $\text{PG}(n - 1, q)$. The following command creates the Desarguesian line-spread in $\text{PG}(3, 2)$ (so $s = 2$, $t = s - 1 = 1$, $m = 2$, $q = 2$, and $Q = 4$):

```
orbiter.out -v 3 \n-define FQ -finite_field -q 4 -end -end \n-define Fq -finite_field -q 2 -end -end \n-with FQ -and Fq -do -finite_field_activity \n- cheat_sheet_desarguesian_spread 2 -end
```

The cheat sheet contains the following spread:

<table>
<thead>
<tr>
<th>Spread element</th>
<th>Element</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 0)</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 0 \ 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>1</td>
<td>(0, 1)</td>
<td>$\begin{bmatrix} 1 &amp; 1 &amp; 0 &amp; 1 \ 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>2</td>
<td>(1, 1)</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 1 \ 1 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>3</td>
<td>(2, 1)</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 1 \ 0 &amp; 1 &amp; 1 &amp; 0 \ 1 &amp; 1 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>4</td>
<td>(3, 1)</td>
<td>$\begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 0 \ 0 &amp; 1 &amp; 1 &amp; 0 \ 1 &amp; 1 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
</tbody>
</table>

Spread elements by rank: (0, 34, 9, 17, 22).

The following command creates the Desarguesian plane-spread in $\text{PG}(5, 2)$:

```
orbiter.out -v 3 \n-define FQ -finite_field -q 8 -end -end \n-define Fq -finite_field -q 2 -end -end \n-with FQ -and Fq -do -finite_field_activity \n- cheat_sheet_desarguesian_spread 2 -end
```
Spread element 0 is \((1, 0)\) =
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Spread element 1 is \((0, 1)\) =
\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
\]

Spread element 2 is \((1, 1)\) =
\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Spread element 3 is \((2, 1)\) =
\[
\begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}
\]

Spread element 4 is \((3, 1)\) =
\[
\begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0
\end{bmatrix}
\]

Spread element 5 is \((4, 1)\) =
\[
\begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0
\end{bmatrix}
\]

Spread element 6 is \((5, 1)\) =
\[
\begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

Spread element 7 is \((6, 1)\) =
\[
\begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

Spread element 8 is \((7, 1)\) =
\[
\begin{bmatrix}
1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

Spread elements by rank: ( 0, 1394, 189, 671, 562, 1040, 792, 1161, 373 )

Apart from the first spread element, the left halves of the generator matrices of the subspaces in the Desarguesian spread are the elements of \(F_8\) in a matrix representation over \(F_2\).

Two \(t\)-spreads are isomorphic if there is a collineation which maps one to the other. The classification problem for \(t\)-spreads is the problem of determining a complete set of pairwise non-isomorphic \(t\)-spreads. Orbiter can be used to classify spreads for small parameters. For instance, the command

195
classifies the line-spreads of PG(3, 4) under the action of PΓL(4, 4). Under the André, Bruck-Bose construction [3, 13], these spreads correspond to translation planes of order 16 with kernel $\mathbb{F}_4$. Up to isomorphism, there are exactly three line-spreads in PG(3, 4). They are the dearguesian spread, the Hall spread, and the semifield spread. Here is the relevant output taken from the latex report:

There are 3 orbits at level 17.

**Orbit 0 / 3 at Level 17**

Node number: 1126

$$\{0, 25, 50, 75, 90, 107, 122, 140, 144, 157, 179, 204, 213, 238, 268, 334, 345\}_{1200}$$

Strong generators for a group of order 1200:

$$\begin{bmatrix}
\omega^2 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
1 & \omega & 0 & 1 \\
\omega^2 & \omega & \omega & 1 \\
\end{bmatrix}_0,$$

$$\begin{bmatrix}
\omega^2 & 0 & 0 & 0 \\
\omega & \omega^2 & 0 & 0 \\
\omega & \omega & 1 & \omega^2 \\
0 & 1 & 1 & 0 \\
\end{bmatrix}_1,$$

$$\begin{bmatrix}
\omega & 1 & \omega & \omega \\
\omega^2 & \omega^2 & \omega & 0 \\
\omega & 0 & 0 & 1 \\
0 & \omega & \omega & 1 \\
\end{bmatrix}_0$$

1,0,0,0,0,1,0,0,2,3,0,2,1,1,3,2,0,
1,0,0,0,1,0,0,2,3,0,2,1,0,2,2,0,1,
1,3,1,1,2,2,0,1,0,0,3,0,1,1,3,0,
There are 0 extensions
Number of generators 3

**Orbit 1 / 3 at Level 17**

Node number: 1127

\{0, 25, 50, 75, 90, 107, 140, 157, 179, 204, 213, 238, 265, 282, 299, 316, 356\}_{81600}

Strong generators for a group of order 81600:

\[
\begin{bmatrix}
\omega & 0 & 0 & 0 \\
0 & \omega & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1
\end{bmatrix}, \begin{bmatrix}
\omega^2 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
1 & \omega & \omega^2 \\
1 & \omega & 1
\end{bmatrix}, \begin{bmatrix}
\omega^2 & 0 & 0 & 0 \\
\omega & \omega^2 & 1 & \omega^2 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1
\end{bmatrix}, \begin{bmatrix}
0 & 0 & 0 & 1 \\
0 & 0 & \omega & 1 \\
0 & 1 & \omega & \omega^2 \\
\omega & 1 & 1 & 1
\end{bmatrix}
\]

There are 0 extensions

Number of generators 7

**Orbit 2 / 3 at Level 17**

Node number: 1128

\{0, 25, 50, 75, 90, 108, 122, 140, 158, 183, 199, 217, 233, 250, 268, 312, 345\}_{576}
Strong generators for a group of order 576:

\[
\begin{bmatrix}
\omega & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
0 & 0 & \omega^2 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
\omega^2 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
\omega & 0 & \omega & 1 \\
\omega^2 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
\omega & \omega & \omega & \omega \\
\omega^2 & 0 & \omega^2 & 0 \\
\omega^2 & 0 & \omega^2 & \omega \\
0 & \omega^2 & \omega^2 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & \omega^2 & 1 \\
1 & \omega^2 & 1 & 0 \\
1 & 0 & \omega & \omega \\
0 & 0 & 0 & 1 \\
\end{bmatrix},
\begin{bmatrix}
\omega^2 & 0 & 0 & 0 \\
0 & \omega^2 & 0 & 0 \\
\omega & 0 & 1 & 1 \\
\omega^2 & 0 & \omega^2 & 1 \\
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix},
\begin{bmatrix}
0 & \omega^2 & \omega & \omega \\
1 & 0 & \omega & \omega \\
1 & 0 & 1 & \omega^2 \\
1 & \omega & \omega & 1 \\
\end{bmatrix}
\]

1,0,0,0,2,0,0,0,0,2,0,0,0,0,3,1,  
1,0,0,0,1,0,0,3,0,3,2,1,0,0,2,0,  
1,0,0,0,3,1,0,0,3,0,2,2,1,0,1,2,0,  
1,1,1,2,0,2,0,2,0,2,1,0,2,2,3,0,  
1,0,3,1,3,1,0,1,0,2,2,0,0,0,1,1,  
0,1,1,0,0,0,0,1,2,0,2,1,3,2,3,2,0,  
There are 0 extensions
Number of generators 6

Via the André, Bruck, Bose construction, spreads give rise to translation planes. For instance, the command

```
orbiter.out -v 3 \\
-define F -finite_field -q 4 -end -end \\
-define PGGL4 -linear_group -PGGL 4 F -end -end \\
-define PGGL5 -linear_group -PGGL 5 F -end -end \\
-with PGGL4 -and PGGL5 -do \\
-group_theoretic_activities -Andre_Bruck_Bose_construction 0 \\
"TP16-4-HALL" -end
```

creates the Hall plane of order 16 using the first of the three spreads from the previous example. The report lists the spread first, then the automorphism group of the plane and then the tactical decomposition:

The spread:
subspace 0 / 17 is 0:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

subspace 1 / 17 is 356:
<table>
<thead>
<tr>
<th>Subspace</th>
<th>Base Value</th>
<th>Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 / 17</td>
<td>25</td>
<td>$\begin{bmatrix} 0 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 0 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>3 / 17</td>
<td>50</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 1 &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>4 / 17</td>
<td>75</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; \omega &amp; 0 \ 0 &amp; 1 &amp; 0 &amp; \omega \end{bmatrix}$</td>
</tr>
<tr>
<td>5 / 17</td>
<td>97</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; 1 \ 0 &amp; 1 &amp; 1 &amp; \omega^2 \end{bmatrix}$</td>
</tr>
<tr>
<td>6 / 17</td>
<td>114</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 1 &amp; 1 \ 0 &amp; 1 &amp; 1 &amp; \omega \end{bmatrix}$</td>
</tr>
<tr>
<td>7 / 17</td>
<td>127</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; \omega &amp; 1 \ 0 &amp; 1 &amp; 1 &amp; 0 \end{bmatrix}$</td>
</tr>
<tr>
<td>8 / 17</td>
<td>153</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; \omega^2 &amp; 1 \ 0 &amp; 1 &amp; \omega &amp; 1 \end{bmatrix}$</td>
</tr>
<tr>
<td>9 / 17</td>
<td>179</td>
<td>$\begin{bmatrix} 1 &amp; 0 &amp; 0 &amp; \omega \ 0 &amp; 1 &amp; \omega^2 &amp; \omega \end{bmatrix}$</td>
</tr>
</tbody>
</table>
subspace 10 / 17 is 191:

\[
\begin{bmatrix}
1 & 0 & 1 & \omega \\
0 & 1 & \omega & 0
\end{bmatrix}
\]

subspace 11 / 17 is 224:

\[
\begin{bmatrix}
1 & 0 & \omega & \omega \\
0 & 1 & \omega & \omega^2
\end{bmatrix}
\]

subspace 12 / 17 is 236:

\[
\begin{bmatrix}
1 & 0 & \omega^2 & \omega \\
0 & 1 & 1 & 1
\end{bmatrix}
\]

subspace 13 / 17 is 262:

\[
\begin{bmatrix}
1 & 0 & 0 & \omega^2 \\
0 & 1 & \omega & \omega
\end{bmatrix}
\]

subspace 14 / 17 is 288:

\[
\begin{bmatrix}
1 & 0 & 1 & \omega^2 \\
0 & 1 & \omega^2 & \omega^2
\end{bmatrix}
\]

subspace 15 / 17 is 297:

\[
\begin{bmatrix}
1 & 0 & \omega & \omega^2 \\
0 & 1 & \omega^2 & 0
\end{bmatrix}
\]

subspace 16 / 17 is 322:

\[
\begin{bmatrix}
1 & 0 & \omega^2 & \omega^2 \\
0 & 1 & \omega^2 & 1
\end{bmatrix}
\]

Automorphism group:

Strong generators for a group of order 921600:

\[
\begin{align*}
\omega & 0 0 0 0 \\
0 & \omega 0 0 0 \\
0 & 0 \omega 0 0 \\
0 & 0 0 \omega 0 \\
0 & 0 0 0 1
\end{align*}
\]

\[
\begin{align*}
1 & 0 0 0 0 \\
0 & 1 0 0 0 \\
0 & 0 1 0 0 \\
0 & 0 0 1 0 \\
0 & 0 0 0 1
\end{align*}
\]

\[
\begin{align*}
1 & 0 0 0 0 \\
0 & 1 0 0 0 \\
0 & 0 1 0 0 \\
0 & 0 0 1 0 \\
0 & 0 0 0 1
\end{align*}
\]
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
\omega^2 & \omega & \omega & 1 & \omega
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,3,0,
1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,2,0,0,0,1,0,
1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,2,0,0,1,0,
1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,3,2,2,1,0,
1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,3,1,0,
1,0,0,0,0,1,0,0,0,0,2,1,1,3,0,1,1,3,3,0,0,0,0,1,0,
1,0,0,0,0,3,1,0,0,0,0,1,1,3,0,1,0,1,0,0,0,0,0,2,0,
1,0,0,0,0,2,2,0,0,0,1,0,3,0,0,2,2,1,1,0,0,0,0,0,1,1,
1,3,2,1,0,0,3,0,1,0,2,1,1,3,0,0,1,0,3,0,0,0,0,0,1,1,

Tactical decomposition schemes:

<table>
<thead>
<tr>
<th></th>
<th>80₁</th>
<th>192₅</th>
<th>1₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>256₀</td>
<td>5</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>5₃</td>
<td>16</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12₂</td>
<td>0</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>80₁</th>
<th>192₅</th>
<th>1₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>256₀</td>
<td>16</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>5₃</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>12₂</td>
<td>0</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>
46 Packings

A packing of $\text{PG}(3, q)$ is a set of pairwise line-disjoint line-spreads of $\text{PG}(3, q)$ of size $q^2+q+1$. Each spread contains $q^2 + 1$ lines. A simple counting argument shows that every line is contained in exactly one spread of the packing. The classification problem for packings is the problem of determining a complete set of pairwise non-isomorphic packings. Orbiter can be used to classify packings for small parameters. It is sometimes useful to make a symmetry assumption. This means that only those packings will be found that satisfy the symmetry assumption. The reason for making such an assumption is that the problem becomes easier and hence more tractable. Often, an assumption is made that the packings are invariant under a (nontrivial) group $H$. This section describes various ways in which Orbiter can help find and classify packings, with or without symmetry assumption.

Table 53 list Orbiter commands related to the construction of packings with assumed symmetry.
Table 54 list Orbiter commands related to the construction of packings with assumed symmetry by picking long orbits.

A packing is regular if it consists solely of regular spreads. The smallest regular packings exist in $\text{PG}(3, 5)$. They were first described by Prince [47]. Up to isomorphism, there are exactly two regular packings in $\text{PG}(3, 5)$. Let us construct these packings. The packings are invariant under a group of order 93. We may pick the subgroup of order 31 to help us construct the packing. Since the order of the 31-Sylow subgroup of $\text{PGL}(4, 5)$ is 31, by Sylow’s theorem there is just one conjugacy class of elements of order 31 in this group. Hence we are free to pick any element of order 31 we like. Since 31 divides $5^3 - 1 = 124$, the element of order 31 can be constructed from a Singer cycle in $\text{PGL}(3, 5)$. To this end, we first need a primitive polynomial of degree 3 over $\mathbb{F}_5$. The command

```
orbiter.out -v 6 -search_for_primitive_polynomial_in_range 5 5 3 3
```

produces such a polynomial. It is

$$X^3 + X^2 + 2.$$  

The companion matrix is

$$\begin{bmatrix} 
0 & 1 & 0 \\
0 & 0 & 1 \\
3 & 0 & 4 
\end{bmatrix}.$$  

The command

```
orbiter.out -v 6 -define G \ 
    -linear_group -GL 3 5 -end -end \ 
    -with G -do \ 
    -group_theoretic_activities \ 
    -raise_to_the_power "0,1,0, 0,0,1, 3,0,4" 31
```

pdflatex GL_3_5_power.tex

can be used to compute the 31-st power of this matrix in $\text{GL}(3, 5)$. Orbiter produces this output:
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-H</td>
<td>description</td>
<td>Specify the assumed group $H$ of symmetries. The orbits of $H$ on the set of spreads are considered. The packings will be constructed as union of orbits.</td>
</tr>
<tr>
<td>-N</td>
<td>description</td>
<td>Specify the normalizer of $H$.</td>
</tr>
<tr>
<td>-cliques_on_fixpoint_graph</td>
<td>$s$</td>
<td>Using poset classification, classify the orbits of $N$ on cliques of size $\leq s$ in the graph on fixed points.</td>
</tr>
<tr>
<td>-cliques_on_fixpoint_graph_control</td>
<td>descr</td>
<td>Specify poset classification options related to the classification of cliques on the fixed point graph as in Tables 31-32.</td>
</tr>
<tr>
<td>-fixp_clique_types_save_individually</td>
<td></td>
<td>Sort the cliques on fixed points by the type of their spreads and write one csv file for each possible type containing the index of the cliques of the given type.</td>
</tr>
<tr>
<td>-process_long_orbits</td>
<td>descr</td>
<td>Proceed on to long orbits using Table 54.</td>
</tr>
<tr>
<td>-problem_label</td>
<td>$L$</td>
<td>Use label $L$ within output file names.</td>
</tr>
<tr>
<td>-spread_tables_prefix</td>
<td>$P$</td>
<td>Use prefix $P$ to access spread tables.</td>
</tr>
<tr>
<td>-output_path</td>
<td>$P$</td>
<td>Use prefix $P$ for output files.</td>
</tr>
<tr>
<td>-report</td>
<td></td>
<td>Create a report of the classification process.</td>
</tr>
<tr>
<td>-regular_packing</td>
<td></td>
<td>Initialize Klein correspondence and identify (regular) spreads with external lines to the Klein quadric using the polarity of the Klein quadric.</td>
</tr>
</tbody>
</table>

Table 53: Orbiter commands related to the construction of packings with assumed symmetry
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-list_of_cases_from_file</td>
<td>fname</td>
<td>Define a subset of cases of fixed point cliques to be worked on. Only the cases listed the given file are considered.</td>
</tr>
<tr>
<td>-split</td>
<td>r m</td>
<td>Define a subset of cases of fixed point cliques to be worked on. Only those cases whose number is congruent to r modulo m are considered.</td>
</tr>
<tr>
<td>-orbit_length</td>
<td>l</td>
<td>Use orbits of length l.</td>
</tr>
<tr>
<td>-clique_size</td>
<td>s</td>
<td>Use exactly s orbits of length l.</td>
</tr>
<tr>
<td>-solution_path</td>
<td>P</td>
<td>Use P as a prefix for all solution files.</td>
</tr>
<tr>
<td>-create_graphs</td>
<td></td>
<td>For each case, create the graph that describes whether two orbits of length l are compatible.</td>
</tr>
<tr>
<td>-solve</td>
<td></td>
<td>Perform clique finding and write solutions to file.</td>
</tr>
<tr>
<td>-read_solutions</td>
<td></td>
<td>Read solutions from file.</td>
</tr>
</tbody>
</table>

Table 54: Orbiter commands related to the construction of packings with assumed symmetry related to picking long orbits
Projectively, this means that the associated group element in $\text{PGL}(3,5)$ has order 31. In order to find an element of order 31 in $\text{PGL}(4,5)$, we embed the $3 \times 3$ matrix into a $4 \times 4$ matrix. The trick is to pick the right element $\alpha$ on the diagonal in

$$
\begin{bmatrix}
\alpha & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 3 & 0 & 4 \\
\end{bmatrix}
$$

so that $\alpha^{31} \equiv 3 \mod 5$ as well. By Fermat’s little theorem,

$$
\alpha^{31} \equiv \alpha^3 \mod 5,
$$

and it is easy to verify that $\alpha = 2$ satisfies

$$
\alpha^3 \equiv 3 \mod 5.
$$

So, the projectivity associated with

$$
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 3 & 0 & 4 \\
\end{bmatrix}
$$

has order 31 in $\text{PGL}(4,5)$. In order to compute the normalizer, we use the command

```
orbiter.out -v 6 -define G \
 -linear_group -PGL 4 5 -end -end \ 
 -with G -do \ 
 -group_theoretic_activities \ 
 -normalizer_of_cyclic_subgroup "31" \ 
 "2,0,0,0, 0,0,1,0, 0,0,0,1, 0,3,0,4" 
 -end
```

We run the file `normalizer_of_31_in_PGL_4_5.magma` through magma to produce the file `normalizer_of_31_in_PGL_4_5.txt`. Repeating the Orbiter command yields the following information about the normalizer of the cyclic subgroup generated by the element of order 31:
The subgroup generated by
\[
\begin{bmatrix}
3 & 0 & 0 & 0 \\
0 & 0 & 4 & 0 \\
0 & 0 & 0 & 4 \\
0 & 2 & 0 & 1
\end{bmatrix}
\]
has order 31
The normalizer has order 372
Strong generators for a group of order 372:
\[
\begin{bmatrix}
4 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & 4 \\
0 & 2 & 3 & 1
\end{bmatrix},
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 2 & 0 \\
0 & 2 & 2 & 1
\end{bmatrix},
\]
1,0,0,0,0,4,0,0,0,0,4,0,0,0,0,4,
1,0,0,0,0,3,0,0,0,0,3,0,0,0,0,3,
1,0,0,0,0,4,0,0,0,0,2,1,0,3,2,4,
1,0,0,0,0,0,1,0,0,0,0,1,0,1,1,3,

Next, we use the following command sequence to create all regular spreads in PG(3, 5):

```
mkdir SPREAD_TABLES_5_REG
orbiter.out -v 6 \\
  -define G \\
  -linear_group -PGL 4 5 -end -end \\
  -with G -do \\
  -group_theoretic_activities \\
    -spread_table_init 2 "12" "SPREAD_TABLES_5_REG/" \\
  -end
```

The first command is a Unix command to create a directory `SPREAD_TABLES_5_REG`, which is where we want to store the files associated with the regular spreads. The second command creates a table of all spreads. There are 21 spreads in PG(3, 5). Orbiter’s built-in table of spreads is used to pull the desarguesian spread. The Orbiter isomorphism type number of the regular spread is 12, which is passed as second argument to the `-spread_table_init` command. The first argument, 2, specifies that we create linepreads in PG(3, 5). The third
argument specifies the prefix for the output files associated with the table of spreads. It is the name of the directory that we have just created. The command will create 155,000 regular spreads. Once the tables of spreads are created. In order to create the packings, we use a bit of makefile programming. The commands

```
PGL_4_5_SUBGROUP_31=-PGL 4 5 \   -subgroup_by_generators "31" 31 1 \   "2,0,0,0, 0,0,1,0, 0,0,0,1, 0,3,0,4"  
PGL_4_5_SUBGROUP_31_NORMALIZER=-PGL 4 5 \   -subgroup_by_generators "normalizer_31" "372" 4 \   "1,0,0,0,4,0,0,0,4,0,0,0,4," \   "1,0,0,0,3,0,0,0,3,0,0,0,3," \   "1,0,0,0,4,0,0,0,2,1,0,3,2,4," \   "1,0,0,0,0,1,0,0,0,1,0,1,1,3," 
```

are used to specify the subgroups $H$ and $N$ in Orbiter language (cf. Section 22). Here, $H$ is the group of order 31 generated by the embedded Singer cycle and $N$ is the normalizer of $H$ in $\text{PGL}(4,5)$. The next command

```
orbiter.out -v 5 \   -define G \   -linear_group -PGL 4 5 -end -end \   -with G -do \   -group_theoretic_activities \   -spread_table_init 2 "12" "SPREAD_TABLES_5_REG/" \   -packing_with_assumed_symmetry \   -problem_label _uniform_reg \   -H $(PGL_4_5_SUBGROUP_31) -end \   -N $(PGL_4_5_SUBGROUP_31_NORMALIZER) -end \   -end \   -end
```

creates the orbits of $H$ on the regular spreads, and filters out those orbits which are useful to build packings (namely, orbits whose spreads are pairwise line-disjoint). There are exactly 8 orbits or length 31 that survive. Each of these orbits is a packing. The orbits are written to the file

```
PGL_4_5_Subgroup_31_31_uniform_reg_reduced_orbits.csv
```

Each Orbit is given as one row in the csv file. The entries in the file are spread numbers and refer to the spread table that we created originally, specifically the file

```
SPREAD_TABLES_5_REG/spread_25_spreads.csv.
```

The command

```
orbiter.out -v 2 \
```
can be used to classify the eight packings. Orbiter produces the following output:

<table>
<thead>
<tr>
<th>Rep</th>
<th>#</th>
<th>Ago</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>93 0,2,4,5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>93 1,3,6,7</td>
</tr>
</tbody>
</table>

Ago distribution:

\[ 93^2 \]

Group order 93 appears for the following 2 classes: \( \{1, 0\} \)

This output shows that there are two isomorphism types of packings, each appearing four times. Orbiter also computes the full stabilizer of each packing, which turns out to be a group of order 93. Orbiter displays generators for the groups. Of course, since we assume \( H \) as group of symmetries, the automorphism groups contain \( H \) as a subgroup of index three.
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-catalogue</td>
<td>$i$</td>
<td>Create BLT-set number $i$ from the Orbiter catalogue ($i$ is zero-based).</td>
</tr>
<tr>
<td>-family</td>
<td>$F$</td>
<td>Create a BLT-set from family $F$. See Table 56 for possibilities for $F$.</td>
</tr>
</tbody>
</table>

Table 55: Commands for creating BLT-sets

### 47 BLT-Sets

A BLT-set of $Q(4,q)$ is a set of $q + 1$ point on the quadric such that no point on the quadric is collinear to more than two points of the set. BLT sets are related to spreads of $\text{PG}(3,q)$, to flocks of the quadratic cone in $\text{PG}(3,q)$, and to many other objects in combinatorics and finite geometry. They exist whenever $q$ is odd. BLT-sets have been defined in [4]. It is an interesting problem to classify BLT-sets of $Q(4,q)$ under the orthogonal group. Many authors have contributed to this problem, leading to ever more efficient algorithms. Some references are Law [32], Penttila-Royle [43], Penttila-Law [33, 34], Betten [7], AlAzemi-Betten-Chowdhury [1].

Orbiter can be used to create members of known families of BLT-sets and sets from a catalogue of BLT-sets over small fields. Besides that, Orbiter can be used to classify all BLT-sets for a given value of $q$. We will see how we create known examples of BLT-sets either from the catalogue or from known families. Afterwards, we will consider the problem of classification.

Table 55 shows options to create known BLT-sets. Table 56 shows options for known families or sporadic sets. For instance, the command

```plaintext
orbiter.out -v 2 \n   -draw_options -end \n   -define F -finite_field -q 11 -end -end \n   -define O -orthogonal_space 0 5 F -end -end \n   -with O -do -orthogonal_space_activity \n       -create_BLT_set -catalogue 0 -end \n   -end
```

creates the BLT-set #0 in $Q(4,11)$. The command

```plaintext
orbiter.out -v 2 \n   -define F -finite_field -q 11 -end -end \n   -define O -orthogonal_space 0 5 F -end -end \n   -with O -do -orthogonal_space_activity \n       -create_BLT_set -family "Mondello" -end \n   -end
```

creates the Mondello BLT-set in $Q(4,11)$. Orbiter creates the following report:
<table>
<thead>
<tr>
<th>$F$</th>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td></td>
<td>Linear BLT-set.</td>
</tr>
<tr>
<td>Fisher</td>
<td></td>
<td>Fisher BLT-set [19].</td>
</tr>
<tr>
<td>Mondello</td>
<td>$q \equiv \pm 1 \mod 10$</td>
<td>Mondello BLT-set due to Penttila [42].</td>
</tr>
<tr>
<td>FTWKB</td>
<td>$q \equiv \pm 2 \mod 3$</td>
<td>Fisher, Thas, Walker [56], Kantor, Betten [11] BLT-set.</td>
</tr>
<tr>
<td>Kantor1</td>
<td>$q = p^e$, $e &gt; 1$</td>
<td>Kantor’s first family.</td>
</tr>
<tr>
<td>Kantor2</td>
<td>$q \equiv \pm 2 \mod 5$</td>
<td>Kantor’s second family.</td>
</tr>
<tr>
<td>LP$_{37_72}$</td>
<td>$q = 37$</td>
<td>BLT-set for $q = 37$ with $\text{ago}=72$ due to Law and Penttila [34].</td>
</tr>
<tr>
<td>LP$_{37_41a}$</td>
<td>$q = 37$</td>
<td>First BLT-set for $q = 37$ with $\text{ago}=4$, due to Law and Penttila [34].</td>
</tr>
<tr>
<td>LP$_{37_41b}$</td>
<td>$q = 37$</td>
<td>Second BLT-set for $q = 37$ with $\text{ago}=4$, due to Law and Penttila [34].</td>
</tr>
<tr>
<td>LP$_{71}$</td>
<td>$q = 71$</td>
<td>BLT-set for $q = 71$ due to Law and Penttila [34].</td>
</tr>
</tbody>
</table>

Table 56: Families of BLT-sets

The quadratic form is:

$$X_0^2 + X_1X_2 + X_3X_4 = 0$$

The BLT-set is:

<table>
<thead>
<tr>
<th>$i$</th>
<th>Rank</th>
<th>Point</th>
<th>$(a, b, c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>846</td>
<td>(1, 6, 4, 10, 3)</td>
<td>(22, 11, 1)</td>
</tr>
<tr>
<td>1</td>
<td>851</td>
<td>(1, 5, 7, 10, 3)</td>
<td>(22, 110, 1)</td>
</tr>
<tr>
<td>2</td>
<td>1234</td>
<td>(1, 5, 1, 7, 7)</td>
<td>(37, 11, 1)</td>
</tr>
<tr>
<td>3</td>
<td>613</td>
<td>(1, 6, 10, 5, 1)</td>
<td>(73, 110, 1)</td>
</tr>
<tr>
<td>4</td>
<td>1307</td>
<td>(1, 1, 3, 8, 5)</td>
<td>(59, 36, 1)</td>
</tr>
<tr>
<td>5</td>
<td>1418</td>
<td>(1, 3, 9, 6, 10)</td>
<td>(95, 36, 1)</td>
</tr>
<tr>
<td>6</td>
<td>1022</td>
<td>(1, 9, 5, 10, 2)</td>
<td>(99, 96, 1)</td>
</tr>
<tr>
<td>7</td>
<td>835</td>
<td>(1, 2, 6, 3, 3)</td>
<td>(99, 36, 1)</td>
</tr>
<tr>
<td>8</td>
<td>950</td>
<td>(1, 10, 8, 2, 9)</td>
<td>(95, 96, 1)</td>
</tr>
<tr>
<td>9</td>
<td>789</td>
<td>(1, 8, 2, 4, 4)</td>
<td>(59, 96, 1)</td>
</tr>
<tr>
<td>10</td>
<td>611</td>
<td>(1, 7, 7, 5, 1)</td>
<td>(73, 11, 1)</td>
</tr>
<tr>
<td>11</td>
<td>1236</td>
<td>(1, 4, 4, 7, 7)</td>
<td>(37, 110, 1)</td>
</tr>
</tbody>
</table>
Plane intersection type is $4^{18} 3^{148}$
Plane invariant is too big (18 planes)

<table>
<thead>
<tr>
<th></th>
<th>18_1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12_0</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

$C_0 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}_{12}$
$C_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}_{18}$

<table>
<thead>
<tr>
<th></th>
<th>18_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12_0</td>
<td>6</td>
</tr>
</tbody>
</table>

$C_0 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}_{12}$
$C_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}_{18}$

The classification of BLT-sets proceeds via the poset of partial BLT-sets. For more details, see [1, 7, 32]. The following command classifies the BLT-sets in $Q(4,5)$:

```
orbiter.out -v 3 \  
-orbiter_path $(ORBITER_PATH) \  
-define F -finite_field -q 5 -end -end \  
-define G -linear_group -PGO 5 F -end -end \  
-with G -do \  
-group_theoretic_activities \  
-poset_classification_control -problem_label BLT_5 -W -depth 6 \  
-draw_options -rad 250 -end \  
-draw_poset -report \  
-end \  
-BLT_starter 6 \  
-end
```

```
pdflatex BLT_5_poset.tex
```

After shortening, the report is:

**The Action**

Group action PGO(5,5) of degree 156
The group is a matrix group.
The base action is on projective space PG(4,5)
$q = 5$
The Orbits

Number of Orbits By Level

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nb of orbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Summary of Orbit Representatives

N = node  
D = depth or level  
O = orbit with a level  
Rep = orbit representative  
(S,O) = (order of stabilizer, orbit length)  
L = number of live points  
F = number of flags  
Gen = number of generators for the stabilizer of the orbit rep.

Table 57: Orbit Representatives

<table>
<thead>
<tr>
<th>N</th>
<th>D</th>
<th>O</th>
<th>Rep</th>
<th>(S,O)</th>
<th>L</th>
<th>F</th>
<th>Gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{}</td>
<td>(9360000, 1)</td>
<td>156</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>{}</td>
<td>(60000, 156)</td>
<td>125</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---------------------</td>
<td>---------</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>{ 0, 1 }</td>
<td>(960, 9750)</td>
<td>40</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>{ 0, 1, 24 }</td>
<td>(72, 130000)</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>{ 0, 1, 24, 25 }</td>
<td>(96, 97500)</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>{ 0, 1, 24, 114 }</td>
<td>(48, 195000)</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>0</td>
<td>{ 0, 1, 24, 25, 26 }</td>
<td>(240, 39000)</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
<td>{ 0, 1, 24, 25, 26, 27 }</td>
<td>(120, 78000)</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0</td>
<td>{ 0, 1, 24, 25, 26, 27 }</td>
<td>(1440, 65000)</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>1</td>
<td>{ 0, 1, 24, 25, 26, 27, 140 }</td>
<td>(720, 13000)</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Poset of Orbits: Diagram

Poset of Orbits in Detail

Orbits at Level 6
There are 2 orbits at level 6.
Orbit 0 / 2 at Level 6

Node number: 8

\[ \{0, 1, 24, 25, 26, 27\}_{1440} \]

0 : 0 = ( 0, 0, 0, 0, 0, 0 )  
1 : 1 = ( 0, 0, 1, 0, 0 )  
2 : 24 = ( 0, 1, 2, 4, 2 )  
3 : 25 = ( 0, 1, 3, 2, 1 )

4 : 26 = ( 0, 1, 3, 4 )  
5 : 27 = ( 0, 1, 2, 1, 3 )

Strong generators for a group of order 1440:

\[
\begin{bmatrix}
4 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix},
\begin{bmatrix}
2 & 0 & 0 & 1 & 2 \\
0 & 4 & 0 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 \\
1 & 0 & 0 & 1 & 4 \\
3 & 0 & 0 & 1 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 2 & 4 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 4 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 2 & 4 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 4 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 2 & 4 \\
0 & 3 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 4 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 2 & 4 \\
0 & 3 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 \\
4 & 0 & 0 & 4 & 1
\end{bmatrix}
\]

There are 0 extensions

Number of generators 7
Orbit 1 / 2 at Level 6

Node number: 9

\[ \{0, 1, 24, 114, 140, 152\}_{720} \]

0 : 0 = ( 0, 1, 0, 0, 0 )  
1 : 1 = ( 0, 0, 1, 0, 0 )  
2 : 24 = ( 0, 1, 2, 4, 2 )  
3 : 114 = ( 1, 1, 2, 1, 1 )

Strong generators for a group of order 720:

\[
\begin{bmatrix}
3 & 0 & 0 & 4 & 0 \\
0 & 2 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 \\
1 & 0 & 0 & 4 & 1
\end{bmatrix},
\begin{bmatrix}
4 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
3 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 4
\end{bmatrix},
\begin{bmatrix}
1 & 0 & 0 & 4 & 2 \\
0 & 3 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 \\
1 & 0 & 0 & 1 & 4 \\
2 & 0 & 0 & 1 & 1
\end{bmatrix},
\begin{bmatrix}
3 & 0 & 0 & 2 & 1 \\
0 & 4 & 0 & 0 & 0 \\
2 & 2 & 4 & 4 & 2 \\
2 & 4 & 0 & 1 & 1 \\
4 & 3 & 0 & 4 & 1
\end{bmatrix},
\begin{bmatrix}
0 & 0 & 0 & 4 & 4 \\
0 & 1 & 0 & 0 & 0 \\
1 & 3 & 1 & 3 & 2 \\
4 & 1 & 0 & 4 & 1 \\
3 & 0 & 3 & 1 & 1
\end{bmatrix},
\begin{bmatrix}
0 & 1 & 1 & 1 & 3 \\
1 & 3 & 1 & 4 & 4 \\
3 & 4 & 3 & 4 & 1 \\
2 & 1 & 3 & 1 & 3 \\
4 & 4 & 1 & 0 & 0
\end{bmatrix}
\]

1,0,0,3,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,3,0,2,0,0,3,2,  
1,0,0,4,0,0,4,0,0,0,0,0,0,4,0,0,2,0,0,4,4,0,0,0,4,0,  
1,0,0,4,2,0,3,0,0,0,0,3,0,0,1,0,0,1,4,2,0,0,1,1,  
1,0,0,4,2,0,3,0,0,0,4,4,3,4,4,3,0,2,2,3,1,0,3,2,  
0,0,0,1,1,0,4,0,0,0,4,2,4,2,3,1,4,0,1,4,4,3,0,1,4,  
1,0,0,4,0,3,4,3,4,1,0,0,4,0,0,0,3,3,0,1,0,1,2,2,  
0,1,1,1,3,1,4,4,3,4,3,4,3,4,1,2,1,3,1,3,4,4,1,0,0,  
There are 0 extensions  
Number of generators 7

This report shows that there are exactly two BLT-sets of \(Q(4,5)\).
Table 58: Types of graphs

## 48 Graph Theory

Orbiter can construct certain algebraically defined graphs. It can also construct and classify small graphs and tournaments up to isomorphism. Table 58 shows some Orbiter commands to create graphs. For instance,

```
orbiter.out -v 2 -create_graph -Johnson 5 2 0 -end -save graph_J520.bin
```

creates $J(5, 2, 0)$, also known as the Petersen graph.

```
orbiter.out -v 2 -create_graph -Paley 13 -end -save graph_P13.bin
```

creates the Paley graph of order 13. Very small graphs can be encoded manually. For instance, the graph

![Graph](image)

can be created using the command
Table 59: Graph Theoretic Activities

<table>
<thead>
<tr>
<th>Key</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-find_cliques</td>
<td>options</td>
<td>Find all cliques.</td>
</tr>
<tr>
<td>-export_magma</td>
<td></td>
<td>Export to Magma [12].</td>
</tr>
<tr>
<td>-export_maple</td>
<td></td>
<td>Export to Maple [39].</td>
</tr>
<tr>
<td>-export_csv</td>
<td></td>
<td>Export to csv-file.</td>
</tr>
<tr>
<td>-print</td>
<td></td>
<td>Print the graph.</td>
</tr>
<tr>
<td>-sort_by_colors</td>
<td></td>
<td>Sort the vertices by color classes.</td>
</tr>
</tbody>
</table>

orbiter.out -v 2 -create_graph -edges_as_pairs 5 \
"0,1,0,2,0,3,0,4,1,3,1,4,2,4" -end

The graph is stored as file graph_v5_e7.colored_graph.

Table 59 shows the commands for graph theoretic activities. For instance,

orbiter.out -v 2 -create_graph -Johnson 5 2 0 -end \ 
-graph_theoretic_activity -export_csv graph_J520.bin -end
orbiter.out -v 2 -draw_matrix Johnson_5_2_0.csv 20

creates the Petersen graph and writes the adjacency matrix to file. The second command creates a bitmap drawing of the adjacency matrix, shown in Figure 26.

The clique finding command allows for additional commands as shown in Table 60. For instance, the cliques of size 3 in the graph graph_v5_e7.colored_graph can be found using

orbiter.out -v 2 -create_graph -load_from_file graph_v5_e7.colored_graph \
-end -graph_theoretic_activity -find_cliques -target_size 3 -end -end

This command finds three cliques of size 3.

Table 61 lists the commands to classify small graphs and tournaments. For instance,
### Table 60: Clique Finding Options

<table>
<thead>
<tr>
<th>Key</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rainbow</td>
<td></td>
<td>Find all rainbow cliques. The size of the cliques is the number of vertex colors.</td>
</tr>
<tr>
<td>-target_size</td>
<td>s</td>
<td>Find all cliques of size $s$.</td>
</tr>
<tr>
<td>-weighted</td>
<td>s</td>
<td>Find weighted cliques.</td>
</tr>
<tr>
<td>-Sajeeb</td>
<td></td>
<td>Use the implementation by Sajeeb Chowdhury.</td>
</tr>
<tr>
<td>-output_file</td>
<td>fname</td>
<td>Write cliques to the named file.</td>
</tr>
<tr>
<td>-restrictions</td>
<td>$l \ r \ m$</td>
<td>Restricted search: At level $l$, restrict to all branches congruent to $r$ modulo $m$. Here, $0 \leq r &lt; m$.</td>
</tr>
</tbody>
</table>

### Table 61: Options for classifying graphs

<table>
<thead>
<tr>
<th>Option</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-girth</td>
<td>$d$</td>
<td>Girth at least $d$</td>
</tr>
<tr>
<td>-regular</td>
<td>$r$</td>
<td>Regular of degree $r$</td>
</tr>
<tr>
<td>-no_transmitter</td>
<td></td>
<td>Tournament without transmitter (requires -tournament)</td>
</tr>
</tbody>
</table>
orbiter.out -v 2 -graph_classify -n 4

classifies all graphs with 4 vertices. For tournaments, the option -tournament can be added. For example,

orbiter.out -v 2 -graph_classify -n 4 -v 2 -tournament \ 
   -draw_graphs_at_level 6

classifies the tournaments on 6 vertices. The -draw_graphs_at_level 6 command instructs Orbiter to draw all representatives at level 6. Figure 27 shows the resulting list of 4 tournaments.
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-draw_matrix</code></td>
<td>csv-file w b</td>
<td>Creates a colored bitmap drawing of the matrix in the csv file, using w pixels per entry. The color bit depth is b pixels. The color indicates the value of the matrix entry.</td>
</tr>
<tr>
<td><code>-draw_matrix_partition</code></td>
<td>s row-part col-part</td>
<td>Add a partition of the rows and columns. The separating lines have a width of s pixels.</td>
</tr>
</tbody>
</table>

Table 62: Commands to Create Windows Bitmap Graphics

## 49 Graphical Output

Orbiter can produce graphical output in a variety of formats:

1. TikZ / Latex [54],
2. Metapost [24],
3. Windows Bitmap files (bmp) [57],
4. Povray, see Section 50.

For instance, bitmaps can be created using the commands shown in Table 62. For instance, the command

```bash
orbiter.out -v 3 -define F -finite_field -q 7 -end -end \
-with F -do -finite_field_activity -cheat_sheet_GF -end
```

```bash
orbiter.out -v 2 -draw_matrix GF_q7_addition_table.csv 40 24 \
-draw_matrix_partition 3 7 7
```

creates a bitmap for the addition table of the field $\mathbb{F}_7$. The input is taken from the file

`GF_q7_addition_table.csv`

which is created by the `-cheat_sheet_GF` command. It has the following content:

<table>
<thead>
<tr>
<th>Row, C0, C1, C2, C3, C4, C5, C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 1, 2, 3, 4, 5, 6</td>
</tr>
<tr>
<td>1, 1, 2, 3, 4, 5, 6, 0</td>
</tr>
<tr>
<td>2, 2, 3, 4, 5, 6, 0, 1</td>
</tr>
<tr>
<td>3, 3, 4, 5, 6, 0, 1, 2</td>
</tr>
<tr>
<td>4, 4, 5, 6, 0, 1, 2, 3</td>
</tr>
<tr>
<td>5, 5, 6, 0, 1, 2, 3, 4</td>
</tr>
<tr>
<td>6, 6, 0, 1, 2, 3, 4, 5</td>
</tr>
</tbody>
</table>

221
The command

```
orbiter.out -v 2 -draw_matrix GF_q7_addition_table.csv 40 24 \
    -draw_matrix_partition 3 7 7
```

creates the diagram in Figure 28.

The command

```
orbiter.out -finite_field_activity -q 4 -cheat_sheet_PG 2 \
    -decomposition_by_element 1 "0,1,0, 0,0,1, 2,1,1, 0" "PG_2_4_singer" -end
orbiter.out -v 2 -draw_matrix PG_2_4_singer_incma_transitive.csv 20 3 \
    -draw_matrix_partition \n    3 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 \n    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

produces Figure 29, which is a drawing of the incidence matrix of PG(2, 4) ordered cyclically with respect to the Singer cycle given by the matrix

$$
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
2 & 1 & 1
\end{bmatrix},
$$

which is the companion matrix of the irreducible polynomial $X^2 + X + \omega$ over $\mathbb{F}_4$ (using the numerical representation with $\omega = 2$).

The poset classification algorithm from sections 29 and 30 computes partially ordered sets. The posets are created using the `-draw_poset` option in the poset classification control command package, see Table 31. The posets are stored in a file with extension `.layered_graph`.

Figure 28: Addition table of $\mathbb{F}_7$
<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-file</td>
<td>fname</td>
<td>Use the given file name for output files.</td>
</tr>
<tr>
<td>-xin</td>
<td>a</td>
<td>Assume input x-coordinates are in the interval [0, a]. Default value: 10000.</td>
</tr>
<tr>
<td>-yin</td>
<td>a</td>
<td>Assume input y-coordinates are in the interval [0, a]. Default value: 10000.</td>
</tr>
<tr>
<td>-xout</td>
<td>a</td>
<td>Assume output x-coordinates are in the interval [0, a]. Default value: 1000000.</td>
</tr>
<tr>
<td>-yout</td>
<td>a</td>
<td>Assume output y-coordinates are in the interval [0, a]. Default value: 1000000.</td>
</tr>
<tr>
<td>-spanning_tree</td>
<td></td>
<td>Place nodes according to a spanning tree. Default value: off.</td>
</tr>
<tr>
<td>-circle</td>
<td></td>
<td>Circle all nodes. Default value: on.</td>
</tr>
<tr>
<td>-corners</td>
<td></td>
<td>Draw corners at the outside of the drawing. Default value: off.</td>
</tr>
<tr>
<td>-rad</td>
<td>r</td>
<td>Use radius r for drawing circles around nodes. Default value: 50.</td>
</tr>
<tr>
<td>-embedded</td>
<td></td>
<td>Create latex headers for stand-alone latex files. Default value: off.</td>
</tr>
<tr>
<td>-sideways</td>
<td></td>
<td>Create latex figure sideways. Default value: off.</td>
</tr>
<tr>
<td>-label_edges</td>
<td></td>
<td>Label the edges in Schreier trees. Default value: off.</td>
</tr>
<tr>
<td>-x_stretch</td>
<td>s</td>
<td>Apply x-axis scaling by a factor of s. Default value: s = 1.0. This option does not affect the drawing of Schreier trees.</td>
</tr>
<tr>
<td>-y_stretch</td>
<td>s</td>
<td>Apply y-axis scaling by a factor of s. Default value: s = 1.0. This option does not affect the drawing of Schreier trees.</td>
</tr>
</tbody>
</table>

Table 63: Drawing Options for Layered Graph Files (Part 1)
Figure 29: A cyclic ordering of the incidence matrix of PG(2, 4)

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Args</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-scale</td>
<td>$s$</td>
<td>Use tikz global scale-factor of $s$. Default value: $s = 0.45$.</td>
</tr>
<tr>
<td>-line_width</td>
<td>$s$</td>
<td>Set tikz line width to $s$. Default value: $s = 1.5$.</td>
</tr>
<tr>
<td>-nodes_empty</td>
<td></td>
<td>Draw nodes empty. Do not label. Default value: off.</td>
</tr>
<tr>
<td>-select_layers</td>
<td>$S$</td>
<td>Draw layers whose index is given in the list $S$ only.</td>
</tr>
<tr>
<td>-paths_in_between</td>
<td>$l_1$</td>
<td>$l_2$ $i_1$ $i_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Draw all paths from node $(l_1, i_1)$ to node $(l_2, i_2)$. Here, $(l, i)$ is the $i$-th node at layer $l$ (counting from zero). Delete all other edges between layers $l_1$ and $l_2$.</td>
</tr>
</tbody>
</table>

Table 64: Drawing Options for Layered Graph Files (Part 2)
Figure 30: The first basic orbit of \( \mathrm{PGL}(4,2) \) as a subgroup of \( \mathrm{PGO}^+(6,2) \)

These files can be drawn using the \texttt{-draw_layered_graph} command. The commands in Table 63 and Table 64 show ways in which to customize the drawings. Let us consider an example. Suppose we are interested in the Schreier trees of a permutation group represented in a Stabilizer chain. For instance, the command

```
orbiter.out -v 12 -linear_group -PGL 4 2 -wedge_detached -end \
    -group_theoretic_activities \
    -report \
    -draw_options -rad 200 -end \
    -end
```

```
pdflatex PGL_4_2_Wedge_4_0_detached_report.tex
open PGL_4_2_Wedge_4_0_detached_report.pdf
```

creates the group \( \mathrm{PGL}(4,2) \) as a subgroup of \( \mathrm{PGO}^+(6,2) \) using the \texttt{-wedge_detached} command from Table 26 and produces a report. The report contains a drawing of the basic orbits of the stabilizer chain. Figure 30 shows the first basic orbit from that report. The \texttt{-rad} option is used to set the radius of the nodes in the drawing.

Suppose we are interested in drawing a poset of orbits from the poset classification algorithm. Here is an example. The first command creates the poset. The second command draws it:

```
orbiter.out -v 3 -linear_group -PGGL 3 4 -end \
    -group_theoretic_activities \
    -classify_cubic_curves -q 4 -target_size 9 -n 3 -d 3 \
    -poset_classification_control -problem_label cc_4 -W -depth 9 \
```
Figure 31: The poset of orbits

orbiter.out -v 3 -draw_layered_graph cc_4_poset_lvl_9.layered_graph \
-rad 300 -embedded -line_width 1.1 -y_stretch 0.9 -scale 0.25 \
-paths_in_between 6 4 9 0 \n-end

pdflatex cc_4_poset_lvl_9_draw.tex
open cc_4_poset_lvl_9_draw.pdf

The drawing is shown in Figure 31.

The command

orbiter.out -v 4 \
-linear_group -PGL 4 2 \n-end \n-group_theoretic_activities \n-orbits_on_polynomials 3 \n-orbits_on_polynomials_draw_tree 6 \n-draw_options -yout 500000 -rad 15 -nodes_empty \n-line_width 0.5 -embedded -end \n-end
draws the 6th Schreier tree in the classification of orbits of PGL(4,2) on homogeneous polynomials of degree 3 in 4 variables. The drawing is shown in Figure 32. This particular orbit has length 420, so there are 420 nodes in the tree.

Suppose we want to draw the set of points in a graphical representation of the plane. The following command produces a picture of an elliptic curve shown earlier in Figure 12.

```
orbiter.out -v 2 -process_combinatorial_objects -q 11 -n 2 \  
-draw_points_in_plane EC_11_1_3 \ 
-fname_base_out EC_11_1_3 -embedded \ 
-input -file_of_points elliptic_curve_b1_c3_q11.txt -end \ 
-end
```
The Povray Interface

Orbiter can be used to create raytracing 3D-graphics. Orbiter serves as a front end for the raytracing software Povray [46]. This is a multi step process: A 3D scene is defined through orbiter commands. Next, Orbiter produces Povray files. After that, the povray files are processed through povray, and turned into graphics files (png), called frames. The frames can be turned into a video by using tools like ffmpeg (see Section 51). By default, an rotational animation is produced.

The Orbiter Povray interface requires some general information about the animation, the camera position, the boundary box for clipping, the font size for text and others. Tables 65-66 list the commands to control the 3D-povray frontend. The main part in a 3D graphics is the scene description. This tells the system what will be in the picture. A scene is composed of objects. Various types of objects are available: points, lines, planes, faces, algebraic surfaces, reguli, 3D-text, and others. Some complex objects are predefined, for instance the Hilbert, Cohn-Vossen surface. Once the objects are defined, output commands can be invoked to draw them in various colors and with various options. At times, there are many objects in one scene. In order to make drawing easier, it is possible to group objects. All objects in a group must have the same type. One group of object can be drawn with one command. Tables 67 and 68 summarize the Orbiter commands to build objects of a 3D scene. Building the scene itself does not create any graphical output. To this end, the commands in Table 69 are used. Each of these commands applies to a group of objects of the same kind. Groups of objects are created using the commands in Table 68 which start with group_of. Here is a simple example which combines scene building and graphical output. The example creates a cube with vertices, edges and faces:

```
1  orbiter.out -v 2 -povray \
2   -round 0 -nb_frames_default 30 -output_mask cube_%d_%03d.pov \ 
3   -video_options -W 1024 -H 768 -global_picture_scale 0.5 \ 
4     -default_angle 75 -clipping_radius 2.7 \ 
5       -end \ 
6       -scene_objects \ 
7         -obj_file cube_centered.obj \ 
8           -edge "0, 1" \ 
9           -edge "0, 2" \ 
10          -edge "0, 4" \ 
11          -edge "1, 3" \ 
12          -edge "1, 5" \ 
13          -edge "2, 3" \ 
14          -edge "2, 6" \ 
15          -edge "3, 7" \ 
16          -edge "4, 5" \ 
17          -edge "4, 6" \ 
18          -edge "5, 7" \ 
19          -edge "6, 7" \ 
20          -group_of_things_as_interval 0 8 \ 
21          -spheres 0 0.3 "texture{ Polished_Chrome pigment{quick_color
```
<table>
<thead>
<tr>
<th>Option</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-do_not_rotate</td>
<td></td>
<td>No rotation. By default, the animation consists of a full rotation around a vertical axis.</td>
</tr>
<tr>
<td>-rotate_about_z_axis</td>
<td></td>
<td>Rotate around z-axis.</td>
</tr>
<tr>
<td>-rotate_about_111</td>
<td></td>
<td>Rotate around (1,1,1)-axis (default).</td>
</tr>
<tr>
<td>-rotate_about_custom_axis</td>
<td>axis</td>
<td>Rotate around a custom axis. The axis is specified as a vector of length 3.</td>
</tr>
<tr>
<td>-boundary_none</td>
<td></td>
<td>Remove the clipping.</td>
</tr>
<tr>
<td>-boundary_box</td>
<td></td>
<td>Clip using a box shape.</td>
</tr>
<tr>
<td>-boundary_sphere</td>
<td></td>
<td>Clip using a sphere (default).</td>
</tr>
<tr>
<td>-font_size</td>
<td>s</td>
<td>Set font size to s.</td>
</tr>
<tr>
<td>-stroke_width</td>
<td>s</td>
<td>Set text depth to s.</td>
</tr>
<tr>
<td>-omit_bottom_plane</td>
<td></td>
<td>Remove the bottom plane.</td>
</tr>
<tr>
<td>-W</td>
<td>w</td>
<td>Set output dimension to w pixels wide.</td>
</tr>
<tr>
<td>-H</td>
<td>h</td>
<td>Set output dimension to h pixels height.</td>
</tr>
<tr>
<td>-nb_frames</td>
<td>n</td>
<td>Set number of frames to n. One revolution around the axis is split into n frames.</td>
</tr>
<tr>
<td>-zoom</td>
<td>r a_s a_t c_s c_t</td>
<td>Set zoom angle and clipping with in round r to change from a_s to a_t and from c_s to c_t, respectively.</td>
</tr>
<tr>
<td>-pan</td>
<td>r F T C</td>
<td>In round r, pan the camera from location F to location T in a rotational movement with center at C. Each of F,T,C are three dimensional coordinates.</td>
</tr>
<tr>
<td>-pan_reverse</td>
<td>r F T C</td>
<td>Same as -pan, but camera movement is in opposite order.</td>
</tr>
<tr>
<td>-no_background</td>
<td></td>
<td>Remove background.</td>
</tr>
<tr>
<td>-no_bottom_plane</td>
<td>r</td>
<td>Remove bottom plane in round r.</td>
</tr>
<tr>
<td>-camera</td>
<td>r S C L</td>
<td>In round r, set camera location at C, sky at S and pointing towards L. Each of S,C,L are three-dimensional coordinate vectors.</td>
</tr>
</tbody>
</table>

Table 65: Options for Orbiter 3D-graphics (Part 1)
<table>
<thead>
<tr>
<th>Option</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-clipping</code></td>
<td><code>r c</code></td>
<td>In round <code>r</code>, set clipping radius to <code>c</code>.</td>
</tr>
<tr>
<td><code>-text</code></td>
<td><code>r a text</code></td>
<td>In round <code>r</code>, produce running text <code>text</code> with sustain value <code>a</code>.</td>
</tr>
<tr>
<td><code>-label</code></td>
<td><code>r s a g text</code></td>
<td>In round <code>r</code>, produce running text <code>text</code> with start value <code>s</code>, sustain <code>s</code> and gravity <code>g</code>.</td>
</tr>
<tr>
<td><code>-latex</code></td>
<td><code>r s a praemable g text l fname</code></td>
<td>In round <code>r</code>, produce running latex text <code>text</code> with start value <code>s</code>, sustain <code>s</code> and gravity <code>g</code>. Put <code>praemable</code> in the latex source code. Use <code>fname</code> for the latex file names (no extension).</td>
</tr>
<tr>
<td><code>-global_picture_scale</code></td>
<td><code>d</code></td>
<td>Set scaling factor to <code>d</code>.</td>
</tr>
<tr>
<td><code>-picture</code></td>
<td><code>r d fname options</code></td>
<td>In round <code>r</code>, place picture <code>fname</code> scaled by <code>d</code> using options.</td>
</tr>
<tr>
<td><code>-look_at</code></td>
<td><code>L</code></td>
<td>Override camera look-at value to <code>L</code>. <code>L</code> is a three-dimensional vector.</td>
</tr>
<tr>
<td><code>-default_angle</code></td>
<td><code>a</code></td>
<td>Set default camera angle to <code>a</code>.</td>
</tr>
<tr>
<td><code>-clipping_radius</code></td>
<td><code>f</code></td>
<td>Set default clipping radius to <code>f</code>.</td>
</tr>
<tr>
<td><code>-scale_factor</code></td>
<td><code>s</code></td>
<td>Set default scale factor to <code>s</code>.</td>
</tr>
<tr>
<td><code>-line_radius</code></td>
<td><code>s</code></td>
<td>Set default line radius to <code>s</code>.</td>
</tr>
</tbody>
</table>

Table 66: Options for Orbiter 3D-graphics (Part 2)
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-cubic_lex</td>
<td>coeffs</td>
<td>Cubic surface given by 20 coefficients in lexicographic ordering</td>
</tr>
<tr>
<td>-cubic_orbiter</td>
<td>coeffs</td>
<td>Cubic surface given by 20 coefficients in Orbiter ordering</td>
</tr>
<tr>
<td>-cubic_Goursat</td>
<td>$A\ B\ C$</td>
<td>Cubic surface with tetrahedral symmetry given by 3 Goursat coefficients as $Axyz + B(x^2 + y^2 + z^2) + C = 0$</td>
</tr>
<tr>
<td>-quadric_lex_10</td>
<td>coeffs</td>
<td>Quadric surface given by 10 coefficients in lexicographic ordering</td>
</tr>
<tr>
<td>-quartic_lex_35</td>
<td>coeffs</td>
<td>Quartic surface given by 35 coefficients in lexicographic ordering</td>
</tr>
<tr>
<td>-octic_lex_165</td>
<td>coeffs</td>
<td>Octic surface given by 165 coefficients in lexicographic ordering</td>
</tr>
<tr>
<td>-point</td>
<td>coeffs</td>
<td>Point given by three coordinates</td>
</tr>
<tr>
<td>-point_list_from_csv_file</td>
<td>fname</td>
<td>List of points with coordinates given in a csv file</td>
</tr>
<tr>
<td>-line_through_two_points_recentered_from_csv_file</td>
<td>fname</td>
<td>List of lines through two points with point coordinates given in a csv file</td>
</tr>
<tr>
<td>-line_through_two_points_from_csv_file</td>
<td>fname</td>
<td>List of lines through two points with point coordinates given in a csv file</td>
</tr>
<tr>
<td>-point_as_intersection_of_two_lines</td>
<td>$i_1\ i_2$</td>
<td>Create a point from the intersection of two lines $i_1$ and $i_2$</td>
</tr>
<tr>
<td>-edge</td>
<td>$i_1\ i_2$</td>
<td>Create an edge (line segment) between points $i_1$ and $i_2$</td>
</tr>
<tr>
<td>-text</td>
<td>$i_1\ s$</td>
<td>Create a label $s$ located at the point $i$</td>
</tr>
<tr>
<td>-triangular_face_given_by_three_lines</td>
<td>$i_1\ i_2\ i_3$</td>
<td>Create a triangular face give by three lines $i_1, i_2, i_3$</td>
</tr>
<tr>
<td>-face</td>
<td>pts</td>
<td>Create a face through the vertices pts, ordered cyclically</td>
</tr>
<tr>
<td>-quadric_through_three_skew_lines</td>
<td>$i_1\ i_2\ i_3$</td>
<td>Create a quadric through three skew lines</td>
</tr>
<tr>
<td>-plane_defined_by_three_points</td>
<td>$i_1\ i_2\ i_3$</td>
<td>Create a plane through three noncollinear points</td>
</tr>
<tr>
<td>-line_through_two_points_recentered</td>
<td>pt-coords</td>
<td>Create a line through two points given by 6 coordinates, recentered</td>
</tr>
</tbody>
</table>

Table 67: Scene definition commands (part 1)
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-line_through_two_points</td>
<td>pt-coords</td>
<td>Create a line through two points given by 6 coordinates</td>
</tr>
<tr>
<td>-line_through_twoexisting_points</td>
<td>i1 i2</td>
<td>Create a line through two points</td>
</tr>
<tr>
<td>-line_through_point_with_direction</td>
<td>x y z u_x u_y u_z</td>
<td>Create a line through a point ((x, y, z)) with a given direction ((u_x, u_y, u_z))</td>
</tr>
<tr>
<td>-plane_by_dual_coordinates</td>
<td>a b c d</td>
<td>Create the plane (ax + by + cz + d = 0) given in dual coordinates</td>
</tr>
<tr>
<td>-dodecahedron</td>
<td></td>
<td>Create a Dodecahedron centered at the origin ((20) points, 30 edges, 12 faces)</td>
</tr>
<tr>
<td>-Hilbert_Cohn_Vossen_surface</td>
<td></td>
<td>Create the Hilbert, Cohn-Vossen surface ((1) cubic surface, 45 tritangent planes, 27 lines)</td>
</tr>
<tr>
<td>-obj_file</td>
<td>fname</td>
<td>Read points and faces from the given .obj file</td>
</tr>
<tr>
<td>-group_of_things</td>
<td>list</td>
<td>Create a group of things from the given list</td>
</tr>
<tr>
<td>-group_of_things_with_offset</td>
<td>list offset</td>
<td>Create a group of things from the given list, each value is increase by offset</td>
</tr>
<tr>
<td>-group_of_things_as_interval</td>
<td>a b</td>
<td>Create a group of things indexed by the interval (a, \ldots, a + b - 1)</td>
</tr>
<tr>
<td>-group_of_things_as_interval_with_exceptions</td>
<td>a b ex</td>
<td>Create a group of things indexed by the interval (a, \ldots, a + b - 1) with the exceptional elements in the list ex removed</td>
</tr>
<tr>
<td>-group_of_all_points</td>
<td></td>
<td>Create a group of things from all points currently defined</td>
</tr>
<tr>
<td>-group_of_all_faces</td>
<td></td>
<td>Create a group of things from all faces currently defined</td>
</tr>
<tr>
<td>-group_subset_at_random</td>
<td>i f</td>
<td>Create a group of things from the existing group (i) by picking a random subset with probability (f)</td>
</tr>
<tr>
<td>-create_regulus</td>
<td>i N</td>
<td>Create a regulus for quadric (i) with (N) lines</td>
</tr>
</tbody>
</table>

Table 68: Scene definition commands (part 2)
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-spheres</td>
<td>$i \ r \ prop$</td>
<td>For each element in point group $i$, create a sphere of radius $r$ with given Povray properties.</td>
</tr>
<tr>
<td>-cylinders</td>
<td>$i \ r \ prop$</td>
<td>For each element in edge group $i$, create a cylinder of radius $r$ with given Povray properties.</td>
</tr>
<tr>
<td>-prisms</td>
<td>$i \ d \ prop$</td>
<td>For each element in face group $i$, create a prism of half-thickness $d$ with given Povray properties.</td>
</tr>
<tr>
<td>-planes</td>
<td>$i \ prop$</td>
<td>For each element in plane group $i$, create a plane with given Povray properties.</td>
</tr>
<tr>
<td>-lines</td>
<td>$i \ r \ prop$</td>
<td>For each element in line group $i$, create a line of radius $r$ with given Povray properties.</td>
</tr>
<tr>
<td>-cubics</td>
<td>$i \ prop$</td>
<td>For each element in group $i$ of cubics, create a surface with given Povray properties.</td>
</tr>
<tr>
<td>-quadrics</td>
<td>$i \ prop$</td>
<td>For each element in group $i$ of quadrics, create a surface with given Povray properties.</td>
</tr>
<tr>
<td>-quartics</td>
<td>$i \ prop$</td>
<td>For each element in group $i$ of quartics, create a surface with given Povray properties.</td>
</tr>
<tr>
<td>-octics</td>
<td>$i \ prop$</td>
<td>For each element in group $i$ of octics, create a surface with given Povray properties.</td>
</tr>
<tr>
<td>-texts</td>
<td>$i \ d \ s \ prop$</td>
<td>For each element in group $i$ of labels, create a text element with half-thickness $d$ and size $s$ with given Povray properties.</td>
</tr>
</tbody>
</table>

Table 69: Graphical output commands
This command instructs Orbiter to create 30 povray files (extension .pov), one for each frame of a rotating scene. The scene contains a cube whose vertices are shown in chrome, whose edges are in red, and whose faces are yellow and transparent. The cube turns around a vertical axis of symmetry. Here is the first frame of the result:

The coordinates of the cube are stored in an object file `cube_centered.obj`. The content of this file is:

```
v  -1  -1  -1
v   1  -1  -1
v  -1   1  -1
v   1   1  -1
v  -1  -1   1
v   1  -1   1
v  -1   1   1
v   1   1   1
f  1  2  4  3
f  1  2  6  5
f  1  3  7  5
f  2  4  8  6
f  3  4  8  7
f  5  6  8  7
```
The monkey saddle is a cubic surface, given by the equation
\[ z = x^3 - 3xy^2 \]
The next example plots the monkey saddle surface together with the tangent plane at (0, 0, 0). An animation is created by rotating the scene around the z-axis.

Here is one of the frames that are created:

The Hilbert Cohn-Vossen surface is given by the equation
\[ \frac{5}{2}xyz - (x^2 + y^2 + z^2) + 1 = 0. \]
We use the following code to plot it:
orbiter.out -v 2 -povray
   -round 0 -nb_frames_default 30 -output_mask HCV_%d_%03d.pov \
   -video_options -W 1024 -H 768 -global_picture_scale 0.9 \ 
   -default_angle 75 -clipping_radius 2.4 \ 
   -camera 0 "1,1,1" "-3,1,3" "0.12,0.12,0.12" \ 
   -end \ 
   -scene_objects \ 
      -Hilbert_Cohn_Vossen_surface \ 
      -group_of_things "0" \ 
      -cubics 0 "texture{ pigment{ White*0.5 transmit 0.5 } finish 
      {ambient 0.4 diffuse 0.5 roughness 0.001 reflection 0.1 specular 
      .8} }" \ 
      -group_of_things_as_interval 0 6 \ 
      -group_of_things_as_interval 6 6 \ 
      -group_of_things_as_interval_with_exceptions 12 15 "14,19,23 " \ 
      -lines 1 0.02 "texture{ pigment{ color Red } finish { diffuse 
      e 0.9 phong 1})" \ 
      -lines 2 0.02 "texture{ pigment{ color Blue } finish { diffu 
      se 0.9 phong 1})" \ 
      -lines 3 0.02 "texture{ pigment{ color Yellow } finish { dif 
      fuse 0.9 phong 1})" \ 
      -label 0 "a1" -label 2 "a2" -label 4 "a3" \ 
      -label 6 "a4" -label 8 "a5" -label 10 "a6" \ 
      -label 12 "b1" -label 14 "b2" -label 16 "b3" \ 
      -label 18 "b4" -label 20 "b5" -label 22 "b6" \ 
      -label 24 "c12" -label 26 "c13" -label 30 "c15" \ 
      -label 32 "c16" -label 34 "c23" -label 36 "c24" \ 
      -label 40 "c26" -label 42 "c34" -label 44 "c35" \ 
      -label 48 "c45" -label 50 "c46" -label 52 "c56" \ 
      -group_of_things_as_interval 0 6 \ 
      -texts 4 0.2 0.15 "texture{ pigment{Black} } no_shadow" \ 
      -group_of_things_as_interval 6 6 \ 
      -texts 5 0.2 0.15 "texture{ pigment{Black} } no_shadow" \ 
      -group_of_things_as_interval 12 12 \ 
      -texts 6 0.2 0.15 "texture{ pigment{Black} } no_shadow" \ 
      -scene_objects_end \ 
   -povray_end \ 

Figure 33 shows the final product. The Schlaefli labeling of lines can be seen.

The Endrass octic [18] is the algebraic surface given by the equation

\[
X^8 := 64 \left( -w^2 + x^2 \right) \left( -w^2 + y^2 \right) \left( (x+y)^2 - 2 w^2 \right) \left( (x-y)^2 - 2 w^2 \right) - \left( -4 \left( 1 + \sqrt{2} \right) \left( x^2 + y^2 \right) \right)^2 + \left( 8 \left( 2 - \sqrt{2} \right) \right) z^2 + \left( 2 + 7 \sqrt{2} \right) \left( x^2 + y^2 \right) - 16 z^4 + 8 \left( 1 - 2 \sqrt{2} \right) z^2 w^2 - \left( 1 + 12 \sqrt{2} \right) w^4 \right] \\\n\]

The following Orbiter command creates a povray graphics of the octic, shown in Figure 34:
Figure 33: The Hilbert Cohn-Vossen surface
orbiter.out -v 2 -povray \
   -round 0 -nb_frames_default 30 -output_mask endrass_octic_%d_%03d.pov \n   -video_options -W 1024 -H 768 -global_picture_scale 0.75 default \n   -camera 0 "1,1,1" "6,6,3" "0,0,0" \n   -rotate_about_111 \n   -end \n   -scene_objects \n     -line_through_two_points_recentered_from_csv_file_coordinate_grid.csv \n     -group_of_things "0" \n     -group_of_things "1" \n     -group_of_things "2" \n     -group_of_things_as_interval 3 39 \n     -lines 0 0.15 "texture{ pigment{ color Red } finish { diffuse 0.9 phong 1}}" \n     -lines 1 0.15 "texture{ pigment{ color Green } finish { diffuse 0.9 phong 1}}" \n     -lines 2 0.15 "texture{ pigment{ color Blue } finish { diffuse 0.9 phong 1}}" \n     -lines 3 0.05 "texture{ pigment{ color Black } finish { diffuse 0.9 phong 1}}" \n     -octics 165 "-93.2548,0,0,0,-309.019,0,0,0,0,0,0,0,0,-1055.06,0,-1582.59,0,-593.47,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-256,0,-468.077,0,-789.019,0,-525.726,0,0.941125" \n     -plane_by_dual_coordinates "0,0,1,0" \n     -group_of_things "0" \n     -group_of_things "0" \n     -octics 4 "texture{ pigment{ White*0.5 transmit 0.5 } finish { ambient 0.4 diffuse 0.5 roughness 0.001 reflection 0.1 specular .8 } }}" \n     -planes 5 "texture{ pigment{ color Blue transmit 0.5 } finish { diffuse 0.9 phong 1}}"\n     -scene_objects.end \n     -povray_end

This illustration includes coordinate axes and the x, y-plane.
Figure 34: The Endrass Octic
<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>s l mask</td>
<td>Specify the input file names by running a printf command with the given mask applied to the index $i$ where $i$ goes from $s$ to $s+l-1$. This option can be repeated.</td>
</tr>
<tr>
<td>-step</td>
<td>s</td>
<td>Increment the index in steps of size $s$.</td>
</tr>
<tr>
<td>-o</td>
<td>mask</td>
<td>Create the output file using the given mask.</td>
</tr>
<tr>
<td>-output_starts_at</td>
<td>i</td>
<td>Start output file indices at $i$ (default is 0).</td>
</tr>
</tbody>
</table>

Table 70: Prepare frames commands

51 Creating Animations

Orbiter can be used to create animations. This relies on the software ffmpeg. In a first step, all frames (i.e. individual graphics files) are created using Orbiter’s povray interface. After that, the frames are used to create the animation. In order to use ffmpeg, the frames should have a uniform file naming scheme, using a consecutive numbering to arrange the files in order. This is achieved by using a printf style mask, with %d representing the number of the current frame. In order to do so, Orbiter can be used to copy and rename files. A temporary directory can be used to collect the files. The Orbiter command prepare_frames can be used. For a list of commands, see Tables 70. For instance, the command

```bash
mkdir FRAMES
orbiter.out -prepare_frames \
   -i 0 30 monkey_0_%03d.png \
   -output_starts_at 0 \
   -o FRAMES/frame%04d.png \
   -end
ffmpeg -r 5 -f image2 -i FRAMES/frame%04d.png -f mp4 -q:v 0 \
   -vcodec mpeg4 monkey.mp4
```

creates a video monkey.mp4 from a set of 30 files. The individual filenames are created using the printf format string monkey_0_%03d.png, with an integer index that is drawn from the interval [0, 29]. The part that starts with a percent sign and ends with a “d” character defines the way in which the integer is formatted. The number three before the “d” indicates that three characters will be printed. The zero indicates the use of leading zeros. So, the first file would be monkey_0_000.png and the very last file is monkey_0_029.png. The description of the printf format string can be found in the documentation of the C standard library [29].
Orbiter can plot functions using a built-in function tracker. The functions must be continuous apart from a finite number of poles. The function can have multiple components, each described using an expression. Each expression is specified in Reverse Polish Notation (RPN). Consider an example. A Lissajous curve is defined using coordinate functions of the form
\[ x = r \sin(at + c), \quad y = r \sin(bt), \quad a, b, c, r \in \mathbb{R}. \]
The terms
\[ r \sin(at + c), \quad r \sin(bt) \]
are the expressions of the two coordinate functions. RPN means that the operator is listed after the operands. A stack data structure is used to hold temporary values. Operators are pushed to the top of the stack using the push commands. A binary operator pops the two elements from the stack, performs the operation, and pushes the resulting value back onto the stack. For a unary operator, only one element is popped and replaced by the result. Here are some examples of expressions rewritten in RPN:
\[ \sin(x) \mapsto \text{push } x \text{ sin}, \]
\[ a + b \mapsto \text{push } a \text{ push } b \text{ add}, \]
\[ a \cdot b \mapsto \text{push } a \text{ push } b \text{ mult}. \]
The coordinate functions are enclosed between \texttt{-code} and \texttt{-code_end} commands. Each coordinate function is described in RPN and terminated using a \texttt{return} keyword. By the time the \texttt{return} keyword is reached, the RPN expression must have exactly one value on the stack which is considered the value of the expression. Constants are declared between the \texttt{-const} and \texttt{-const_end} keywords. Likewise, variables are declared between the \texttt{-var} and \texttt{-var_end} keywords. Picking \( a = 3 \), \( b = 2 \), \( c = \pi/2 \) and \( r = 7 \), the function is computed using
```
orbiter.out -v 2 -smooth_curve "lissajous" 0.07 2000 15 0 18.85 \ 
-const a 3 b 2 c 1.57 r 7 -const_end \ 
-var t -var_end \ 
-code \ 
    push t push a mult push c add sin push r mult return \ 
    push t push b mult sin push r mult return \ 
-code_end
```
The sequence
\[ \text{push } t \text{ push } a \text{ mult push } c \text{ add sin push } r \text{ mult} \]
is \( r \sin(at + c) \) expressed in RPN. The constants are defined in the line
```
-const a 3 b 2 c 1.57 r 7 -const_end
```
The input variable is defined using the line
```
-var t -var_end
```
The sequence
-\text{smooth\_curve} \ "lissajous" 0.07 2000 15 0 18.85
defines the name of the output file, the fact that two consecutive points are never further than $\epsilon = 0.07$ away, the fact that points that are 15 or more away from the origin should be ignored, and the fact that the variable $t$ loops over the range $[0, 18.85]$ with a default of 2000 steps. The evaluator automatically reduces the step-size if consecutive image points are more than $\epsilon$ apart. The code to produce the plot is

```plaintext
orbiter.out -v 2 -povray \
-round 0 -nb_frames_default 1 -output_mask lissajous_N%d_N%03d.pov \
-video_options -W 1024 -H 768 -global_picture_scale 0.40 \
-default_angle 45 -clipping_radius 5 -omit_bottom_plane \
-camera 0 "0,-1,0" "0,0,12" "0,0,0" \
-rotate_about_z_axis \
-end \
-scene_objects \
-line_through_two_points_recentered_from_csv_file coordinate_grid.csv \
-group_of_things "0" \
-group_of_things "1" \
-group_of_things "2" \
-lines 0 0.09 "texture{ pigment{ color Yellow } }" \
-lines 1 0.09 "texture{ pigment{ color Yellow } }" \
-lines 2 0.09 "texture{ pigment{ color Yellow } }" \
-group_of_things_as_interval 3 39 \
-lines 3 0.02 "texture{ pigment{ color Black } }" \
-point_list_from_csv_file function_lissajous_N2000_points.csv \
-group_of_things_as_interval 0 6524 \
-spheres 4 0.1 "texture{ pigment{ color Red } } finish { diffuse 0.9 phong 1}" \
-plane_by_dual_coordinates "0,0,1,0" \
-group_of_things "0" \
-planes 5 "texture{ pigment{ color Blue*0.5 transmit 0.5 } }"
\scene_objects_end \
povray_end
```

The plot is shown in Figure 35. We can turn it into a 3D plot by using the $t$ value for the $z$ coordinate. The code to produce the 3D plot is

```plaintext
orbiter.out -v 2 -povray \
-round 0 -nb_frames_default 1 -output_mask lissajous_N%d_N%03d.pov \
-video_options -W 1024 -H 768 -global_picture_scale 0.40 \
-default_angle 45 -clipping_radius 5 -omit_bottom_plane \
-camera 0 "0,-1,0" "0,0,12" "0,0,0" \
-rotate_about_z_axis \
-end \
-scene_objects \
-line_through_two_points_recentered_from_csv_file coordinate_grid.csv \
-group_of_things "0" \
-group_of_things "1" \
-group_of_things "2" \
-lines 0 0.09 "texture{ pigment{ color Yellow } }" \
-lines 1 0.09 "texture{ pigment{ color Yellow } }" \
-lines 2 0.09 "texture{ pigment{ color Yellow } }" \
-group_of_things_as_interval 3 39 \
-lines 3 0.02 "texture{ pigment{ color Black } }" \
-point_list_from_csv_file function_lissajous_N2000_points.csv \
-group_of_things_as_interval 0 6524 \
-spheres 4 0.1 "texture{ pigment{ color Red } } finish { diffuse 0.9 phong 1}" \
-plane_by_dual_coordinates "0,0,1,0" \
-group_of_things "0" \
-planes 5 "texture{ pigment{ color Blue*0.5 transmit 0.5 } }"
\scene_objects_end \
povray_end
```
Figure 35: Lissajous figure

```
-round 0 -nb_frames_default 30 -output_mask lissajous_3d_%d_%03d.pov

-video_options -W 1024 -H 768 -global_picture_scale 0.40

-default_angle 45 -clipping_radius 5 -omit_bottom_plane

-camera 0 "0,0,1" "7,7,5" "0,0,1"

-rotate_about_z_axis

-end

-scene_objects

-line_through_two_points_recentered_from_csv_file coordinate_grid.csv

-group_of_things "0"

-group_of_things "1"

-group_of_things "2"

-lines 0 0.09 "texture{ pigment{ color Yellow } }"

-lines 1 0.09 "texture{ pigment{ color Yellow } }"

-lines 2 0.09 "texture{ pigment{ color Yellow } }"

-group_of_things_as_interval 3 39

-lines 3 0.02 "texture{ pigment{ color Black } }"

-point_list_from_csv_file function_lissajous_3d_N2000_points.csv

-group_of_things_as_interval 0 6538

-spheres 4 0.1 "texture{ pigment{ color Red } finish { diffuse 0.9 phong 1} }"

-plane_by_dual_coordinates "0,0,1,0"

-group_of_things "0"

-scene_objects_end

-povray_end
```
The function is computed using the command

```c
orbiter.out -v 2 -smooth_curve "lissajous_3d" 0.07 2000 50 0 18.85 \
    -const a 3 b 2 c 1.57 r 7 -const_end \n    -var t -var_end \n    -code \n        push t push a mult push c add sin push r mult return \n        push t push b mult sin push r mult return \n        push t return \n    -code_end \n```

The 3D curve is shown in Figure 36.
<table>
<thead>
<tr>
<th>Option</th>
<th>Arguments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-csv_file_select_rows</td>
<td>fname R</td>
<td>Selects rows listed in $R$ from the csv-file fname.</td>
</tr>
<tr>
<td>-csv_file_select_cols</td>
<td>fname R</td>
<td>Selects columns listed in $R$ from the csv-file fname.</td>
</tr>
<tr>
<td>-csv_file_select_rows_and_cols</td>
<td>fname $R$ $C$</td>
<td>Selects rows listed in $R$ and columns listed in $C$ from the csv-file fname.</td>
</tr>
<tr>
<td>-csv_file_join</td>
<td>fname col-label</td>
<td>Joins csv file fname according to column with label col-label. This option is given once for each file that should be joined.</td>
</tr>
<tr>
<td>-csv_file_latex</td>
<td>fname</td>
<td>Produces a latex table from the given csv-file.</td>
</tr>
</tbody>
</table>

Table 71: Miscellaneous Orbiter Commands

53 Miscellaneous

Table 71 list miscellaneous Orbiter commands. The command `-csv_file_select_rows` can be used to select rows from a csv file. The command `-csv_file_select_cols` can be used to select columns from a csv file. The command `-csv_file_select_rows_and_cols` selects rows and columns. Here is an example. We create the multiplication table of the finite field $\mathbb{F}_7$, ordered according to the powers of a primitive element:

$$\alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5.$$  

After that, we pull the rows and columns corresponding to even powers $\alpha^0, \alpha^2, \alpha^4$.

```
orbiter.out -v 3 -define F -finite_field -q 7 -end -end \   
  -with F -do -finite_field_activity -cheat_sheet_GF -end \   
  orbiter.out -v 4 -csv_file_select_rows_and_cols \   
    GF_q7_multiplication_table_reordered.csv \   
    "0,2,4" "0,2,4" \   
orbiter.out -v 2 -draw_matrix GF_q7_multiplication_table_reordered.csv \   
    40 24 -draw_matrix_partition 3 6 6 \   
orbiter.out -v 2 -draw_matrix \   
    GF_q7_multiplication_table_reordered_select.csv \   
    40 24 -draw_matrix_partition 3 3 3
```

The even powers of $\alpha$ create a multiplicative subgroup. Figure 37 shows the table of the multiplicative group $\mathbb{F}_7^*$ and the subgroup of squares (compare with Figure 5 in Section 8). Here is the file `GF_q7_multiplication_table_reordered.csv`
Figure 37: Cyclic multiplication table of $\mathbb{F}_7$ and subgroup of squares

<table>
<thead>
<tr>
<th>Row</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

and next the file that is created by selecting rows and columns 0, 2, 4:

<table>
<thead>
<tr>
<th>Row, &quot;C0&quot;, &quot;C2&quot;, &quot;C4&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, &quot;1&quot;, &quot;2&quot;, &quot;4&quot;</td>
</tr>
<tr>
<td>1, &quot;2&quot;, &quot;4&quot;, &quot;1&quot;</td>
</tr>
<tr>
<td>2, &quot;4&quot;, &quot;1&quot;, &quot;2&quot;</td>
</tr>
</tbody>
</table>

END
54 Limitations

Several limitations exist in Orbiter. Here is a list:

1. Field elements are encoded as int. This limits the size of fields that can be handled to $2^{8s-1}$ where $s = \text{sizeof(int)}$.

2. The ranks of elements in the permutation domain are encoded as long int. This limits the size of permutation domains that can be handled. The degree of a permutation group must be less than $2^{8s-1}$ where $s = \text{sizeof(long int)}$.

3. The finite field class builds tables for the addition and multiplication of field elements. This restricts the size of the fields that can be created.

4. The projective geometry class tries to build a bitmatrix for the adjacency matrix if the number of lines is less than MAX_NUMBER_OF_LINES_FOR_INCIDENCE_MATRIX which is defined in src/lib/foundations/geometry/projective_space.cpp. If the number of lines is too big, the table is not created. In this case, the projective geometry class may behave slower.

5. The projective geometry class tries to build a table for the lines if the number of points is less than MAX_NUMBER_OF_POINTS_FOR_POINT_TABLE and the number of lines is less than MAX_NUMBER_OF_LINES_FOR_LINE_TABLE, both of which are defined in src/lib/foundations/geometry/projective_space.cpp. If the number of points is too big, the table is not created. In this case, the projective geometry class may behave slow.

6. The projective geometry class tries to build a table for the lines through any two points if the number of points is less than MAX_NB_POINTS_FOR_LINE_THROUGH_TWOPOLNTS_TABLE which is defined in src/lib/foundations/geometry/projective_space.cpp. If the number of points is too big, the table is not created. In this case, the projective geometry class may behave slow.

7. The projective geometry class tries to build a table for the intersection points of pairs of lines if the number of points is less than MAX_NB_POINTS_FOR_LINE_INTERSECTION_TABLE which is defined in src/lib/foundations/geometry/projective_space.cpp. If the number of points or lines is too big, the table is not created. In this case, the projective geometry class may behave slow.

8. For Windows users: Cygwin by default uses 32 bit integers for both int and long int. Using Cygwin 64 to compile Orbiter recommended.

9. A limited list of primitive polynomials are hard-coded in Orbiter. For large fields, the user must provide their own primitive polynomial. The polynomials encoded in orbiter are not guaranteed to be compatible with the subfield relationship.
55 Contributors

Orbiter contains the work of many collaborators. Here is a partial list.

Anton Betten. Main author of Orbiter.


Dieter Betten. Dieter contributed to algorithms for configurations and linear spaces.
**Brendan McKay.** Brendan contributed the graph theory package Nauty and a solver for positive systems of diophantine equations. Brendan pioneered the technique of poset classification in combinatorics.

**Bernd Schmalz.** Bernd invented the variant of the poset classification algorithm used in orbiter.

**Stefaan De Winter.** Stefaan helped develop algorithms for orthogonal geometries.

**Vladimir Tonchev.** Vladimir helped develop algorithms for unitals and codes.
Sajeeb Roy Chowdhury. Sajeeb contributed a parallel clique finding algorithm and parallel CRC search. He also worked on shallow Schreier trees.

Fatma Karaoglu. Fatma contributed to the algorithms for cubic surfaces.

Awss Alogaidi, Eun-Ju Cheon and Tatsuya Maruta all contributed to the search for arcs in projective planes.
Alissa Braun, Brendan Looi, Joel Barraza-Nova and ZiYao Cheng helped with cubic surfaces. Alissa also worked on CRC search.

Xi Chen. Xi contributed a DLX solver.

Paul Macklin. Paul contributed software to write bmp bitmap files.

Nick Gammon. Nick contributed a parser for expressions, based on earlier work of Bjarne Stroustrup in “The C++ Programming Language.”
References


[27] Fatma Karaoglu and Anton Betten. The number of cubic surfaces with 27 lines over a finite field. To appear in Journal of Algebraic Combinatorics.


253


[41] Brendan McKay. Possolve, A program to solve positive integer systems, Australian National University, private communication, around 1997.


[48] L. Schläfli. An attempt to determine the twenty-seven lines upon a surface of the third order and to divide such surfaces into species in reference to the reality of the lines upon the surface, Quart. J. Math. 2 (1858), 55–110.


