An Introduction to polymake 2.12

Michael Joswig*

Abstract

polymake is is a tool to study the combinatorics and the geometry of convex polytopes and polyhedra. It is also capable of dealing with simplicial complexes, matroids, polyhedral fans, graphs, some objects from tropical geometry, and more.

1 Introduction

This text is meant as a short introduction to the system. It refers to features of the version 2.12, released March 19, 2012. While polymake is capable of dealing with a number of different mathematical objects the bulk of the code still deals with convex polytopes. Therefore we begin with polytopes, that is, convex hulls of finitely many points in Euclidean space. The idea is to show a typical use case.

The following people contributed to this version of polymake: Benjamin Assarf, Ewgenij Gawrilow, Katrin Herr, Sven Herrmann, Lars Kastner, Benjamin Lorenz, Silke Möser, Andreas Paffenholz, Thomas Rehn, Thilo Rörig, Benjamin Schröter.

2 An Example Session

In the following commands in the polymake shell are preceded with a '>' symbol. the language is a dialect of Perl. We start by producing a polytope which is the Cartesian product of a pentagon and a heptagon (both regular), and we want to know the f-vector, which records the number of faces per dimension.

```
> $p57=product(n_gon(5),n_gon(7));
> print $p57->F_VECTOR;
35 70 47 12
```

This means that our polytope called $p57 has 35 vertices, 70 edges, 47 two-dimensional faces and twelve three-dimensional faces. The latter are the *facets* of $p57.

A key concept in polymake is that mathematical objects, such as our polytope $p57$, have *properties* which the user is interested in. In the example above the user wants to learn about the property F_VECTOR. For basic usage the user can deal with objects and properties in a naive way: If a user asks for a specific property the system decides how to compute it and tells the result. Relevant intermediate information is kept and does not need to be re-computed if requested later. The list of available properties depends on the type of the object as well as on the individual user's installation. The latter holds since polymake can be extended in ways which are transparent to the user.

We proceed with the example from above. By construction the polytope $p57 has a great deal of symmetry, and this is what we want to exhibit in the next few commands. The group of *combinatorial automorphisms* is the automorphism of the face lattice. In this case it is generated by four elements, each of which is written as a pair of permutations. The first in each pair gives the action on the twelve facets, numbered from 0 to 11, while the second gives the action on the 35 vertices. Each line of the output is truncated slightly.

```
> $Aut=automorphisms($p57->VERTICES_IN_FACETS);
> print $Aut;
(<7 6 2 3 4 5 1 0 9 8 10 11> <0 6 5 ...>)
(<0 1 2 5 4 3 6 7 8 9 11 10> <0 1 2 ...>)
(<1 2 6 3 4 5 7 8 9 0 10 11> <1 2 3 ...>)
(<0 1 2 4 5 10 6 7 8 9 11 3> <7 8 9 ...>)
```

*Fachbereich Mathematik, TU Darmstadt, 64289 Darmstadt, Germany, joswig@mathematik.tu-darmstadt.de

This is an abstract related to a tutorial or demo given at GTS 2012. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear in a conference with formal proceedings and/or in a journal.

It is a new feature of version 2.12 that, via an interface to `permlib` [2], `polymake` can treat finite permutation groups as objects.

```
> $G=new group::Group(GENERATORS=>[map { $_->first } @$Aut]);
> print $G->ORDER;
140
```

The whole `polymake` project is subdivided into *applications* which center around one (or slightly more than one) kind of object. We dealt with the application `polytope` so far, and now we do a context switch to the application for groups. This way we can access functions from that application without the need to qualify the application such as in `group::Group` above.

```
> application 'group';
> $orbits=compute_domain_orbits($G);
> print $orbits;
{0 1 2 6 7 8 9}
{3 4 5 10 11}
```

This is the orbit structure on the facets. So we got two types (namely, prisms over pentagons and prisms over heptagons). The indices refer to the implicit numbering of the facets in the object `$p57`; this can be made explicit as follows.

```
> print rows_numbered(convert_to<Float>($p57->FACETS));
0:4.0489173395223 0 0 -1 -4.38128626753482
1:1.44504186791263 0 0 1 -1.2539603376627
2:1 0 0 1.10991626417474 0
3:2.6180339887499 1 -3.07768353717525 0 0
4:1 1.23606797749979 0 0 0
5:2.6180339887499 1 3.07768353717525 0 0
6:1.44504186791263 0 0 1 1.2539603376627
7:4.0489173395223 0 0 -1 4.38128626753482
8:2.07652139657234 0 0 -2.07652139657234 1
9:2.07652139657234 0 0 -2.07652139657234 -1
10:1.37638192047117 -1.37638192047117 1 0 0
11:1.37638192047117 -1.37638192047117 -1 0 0
```

Let us create a picture. The *dual graph* of a polytope is the abstract graph whose nodes are the facets (that is, the faces of codimension one) and whose edges correspond to the faces codimension two shared by two facets. Nodes corresponding to facets in the same orbit receive the same color. `polymake` uses a spring embedding algorithm to draw this graph without making use of geometric information. This way the combinatorial symmetry is automatically reflected, as shown in Figure 1.

```
> @color = map {
             $orbits->[0]->contains($_) ? "red" : "blue"
           } 0..($p57->N_FACETS-1);
> $p57->VISUAL_DUAL_GRAPH(NodeColor=>\@color);
```

3 Getting `polymake`

`polymake` runs in a UNIX like environment. A C++ compiler (preferably `gcc` 4.6.x) and a Perl interpreter are necessary in order to run the system. Java support is necessary for most visualization features. Notice that a C++ compiler is required even if you do not write C++ code yourself. Ubuntu 11.10 is recommended, but all other recent Linux versions should work, too. FreeBSD is supported as well. The new C++ compiler project `clang` is not supported yet but will be in the near future.

It is recommended to download the `polymake` tar ball from `http://polymake.org/doku.php/download/start` and to install it via

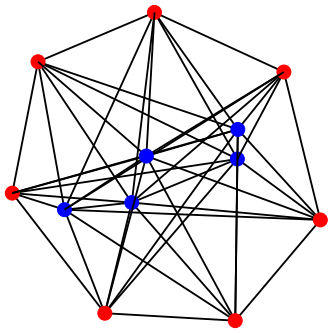

Figure 1: Dual graph of the product of a pentagon and a heptagon

```
> ./configure
> make
> make install
```

Installation on MacOS X from scratch is slightly more involved, hence we recommend the bundle (`polymake` 2.12 versions available for Mac OS 10.6 and 10.7).

For those with a Intel based machine but without any of the operating systems mentioned above there is a live CD version based on Linux Mint 12 LXDE (probably too large for one CD) and another smaller one based on Ubuntu Mini Remix 11.10.

Recent versions of `Cygwin` on various flavors of Windows might be able to run some portions of `polymake` but will invariably crash upon hitting the first exception (frequently used in `polymake`, and not properly supported in `Cygwin`).

References

[1] Ewgenij Gawrilow and Michael Joswig. polymake: a framework for analyzing convex polytopes. In *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, volume 29 of *DMV Sem.*, pages 43–73. Birkhäuser, Basel, 2000.

[2] Thomas Rehn. permlib — a callable C++ library for permutation computations. `http://www.math.uni-rostock.de/~rehn/software/permlib.html`.