PALP:

A Package for Analyzing Lattice Polytopes with Applications to Toric Geometry

Maximilian KREUZER[*]

Institut für Theoretische Physik, Technische Universität Wien
Wiedner Hauptstraße 8–10, A-1040 Wien, AUSTRIA

and

Harald SKARKE[#]

Mathematical Institute, University of Oxford
24-29 St. Giles', Oxford OX1 3LB, ENGLAND

ABSTRACT

We describe our package PALP of C programs for calculations with lattice polytopes and applications to toric geometry, which is freely available on the internet. It contains routines for vertex and facet enumeration, computation of incidences and symmetries, as well as completion of the set of lattice points in the convex hull of a given set of points. In addition, there are procedures specialised to reflexive polytopes such as the enumeration of reflexive subpolytopes, and applications to toric geometry and string theory, like the computation of Hodge data and fibration structures for toric Calabi-Yau varieties. The package is well tested and optimised in speed as it was used for time consuming tasks such as the classification of reflexive polyhedra in 4 dimensions and the creation and manipulation of very large lists of 5-dimensional polyhedra. While originally intended for low-dimensional applications, the algorithms work in any dimension and our key routine for vertex and facet enumeration compares well with existing packages.

[*]e-mail: kreuzer@hep.itp.tuwien.ac.at
[#]e-mail: skarke@maths.ox.ac.uk

1 Introduction

PALP is a package that contains a number of procedures for computing various data on lattice polytopes. It is freely available from our website [1] under the GNU licence terms. It is written in C and provides the general purpose program *poly.x*, as well as a number of further main programs for special tasks (these include *class.x*, which was created for classifying reflexive polyhedra, and *nef.x*, which analyzes complete intersection Calabi-Yau manifolds; both are useful in the context of toric geometry). While these programs cover many applications relevant for someone working with lattice polytopes or toric Calabi–Yau spaces, a variety of basic routines from the package can be used to create new main programs for other purposes.

Basic tasks of the package include the enumeration of vertices and facet equations of the convex hull of a given set of points, an efficient completion of the set of lattice points in this convex hull, and computation of the incidence structure of the face lattice. It is also possible to analyze symmetries of a polytope in terms of $GL(\mathbb{Z})$ lattice automorphisms that leave it invariant; here it is important to note that we work in a linear setting (i.e., the lattice origin is a distinguished point). In the same setting the computation of a normal form is provided; this is a specific ordering of the vertices and choice of coordinate system such that the normal forms of any two polytopes related by a lattice automorphism are the same.

PALP places special emphasis on reflexive polytopes (i.e., lattice polytopes whose duals are again lattice polytopes) and contains (or rather, has evolved from) routines for their classification. Reflexive polytopes play an important role in algebraic geometry where they encode the combinatorial data of Calabi–Yau hypersurfaces in toric varieties, leading to applications also in string theory and conformal field theory. PALP contains a number of procedures for applications in these contexts, like the evaluation of Hodge data or the search for reflexive sections of reflexive polytopes (corresponding to fibration structures of the Calabi–Yau spaces). We have tried, wherever possible, to keep the description in this paper at the level of lattice polytopes such that background knowledge in toric geometry and/or confomal field theory is required only for understanding parts of section 4.

Our original aim in constructing the package was optimal performance in terms of calculation time, memory efficiency and numerical stability for reflexive polytopes in four dimensions. We also used it to produce a number of new results on 5- and 6-dimensional reflexive polyhedra. Testing our vertex and facet enumeration routines on higher (e.g. 12, 15, 20) dimensional polytopes revealed that they compete well with those of existing packages [2–5]. Like "cdd", PALP is based on a double description type of algorithm [6] and uses a simple pivoting strategy which is turned on for large dimension with a parameter. Similarly, progress in the polyhedron analysis is monitored only in high dimensions (for both parameters the defaults are set in *Vertex.c*).

Turning to the limitations of the package, we first have to point out that we work with fixed precision because speed was one of our main concerns when designing the package. We are using two data types, "Long" and "LLong", which are usually set to "long" (32 bit) and "long long" (64 bit) in the file *Global.h*. "LLong" is routinely used in numerically critical circumstances.[1] In many examples with small coordinates (also in high dimensions) one may try so save CPU

---

[1] The most notable example is, perhaps, the function "EEV_To_Equation", which computes a new facet equation by joining the intersection of two equations with a newly found vertex. Here it seems to be unavoidable that intermediate expressions can be of the order of squares of both input and output numbers.

time by using only 32 bit for "LLong". More importantly, the analysis of very large reflexive polytopes typically requires 64 bit in all operations in dimension 5 or higher. In any case, we implemented many tests that should alert the user in almost all cases where an overflow invalidates the result of a calculation.

Maximal dimensions, numbers of points, vertices and faces, etc., also have to be fixed at compilation time. This is so because the whole package was originally designed for polytopes of low dimensions (typically 4 to 6), where reflexivity puts a priori limits to these numbers. While facet enumeration still works well in higher dimensions, some parts of the package, like the computation of incidences, may run into problems with stack size, because the number of faces may soon get very large.

The package in its present status assumes the input polyhedron to span the dimension. This, and many other limitations, could be avoided by combining routines that already exist in the package. Already now, the user can obtain a set of equations that define the affine subspace that is spanned by the set of input points, which should be sufficient to proceed in many applications. While PALP contains powerful routines for dealing with very large numbers of polytopes (sorting, compression, data base structures etc.) these applications are currently restricted to the reflexive case. We hope that the present form of the package is only the starting point and that many other applications will be created. We will be happy to provide some help in this process and/or to include user contributions in an appropriate way.

The structure of this paper is such that it should reflect growing specialisation from general aspects of lattice polytopes via reflexive polytopes to applications in algebraic geometry and conformal field theory. For many users it may be sufficient to look at section 2 where we describe basic I/O formats and options of the main program *poly.x*. The formats are designed such that the main programs can be combined in pipelines for iterating certain operations. How this works is illustrated in section 3, where we present two examples: the search for minimal reflexive polyhedra that contain a given polytope, and the classification of reflexive polyhedra in 3 dimensions. Section 4 is mainly intended for readers who are interested in applications to algebraic geometry and conformal field theory. Here we provide some basics of toric geometry and discuss some options that can be used to compute Hodge data and Poincaré polynomials related to transversal polynomials. In section 5 we explain the structure of the package, as it is reflected in the dependences given in the makefiles and the definitions and routines in the main header file *Global.h*. This should provide a basic understanding of how the parameters have to be set in various contexts, and should also enable more ambitious users to design their own programs for applications that are not covered in the present version of PALP.

2   IO formats and options for *poly.x*

After obtaining our package from our web site [1] and following the instructions given there for unpacking, it is possible to use *poly.x* more or less immediately, simply by typing "make poly" and calling *poly.x* with its help option. In the following, as in all of our examples, user input is marked by bold face.

```
$ poly.x -h
This is ``poly.x'':  computing data of a polytope P
Usage:   poly.x [-<Option-string>] [in-file [out-file]]
Options (concatenate any number of them into <Option-string>):
  h  print this information         | n  do not complete polytope or
  f  use as filter                  |      calculate Hodge numbers
  g  general output:                | i  incidence information
     P reflexive: numbers of (dual) | s  check for span property
       points/vertices, Hodge numbers |    (only if P from CWS)
     P not reflexive: numbers of    | I  check for IP property
       points, vertices, equations  | S  number of symmetries
  p  points of P                    | T  upper triangular form
  v  vertices of P                  | N  normal form
  e  equations of P/vertices of P-dual | t  traced normal form computation
  m  pairing matrix between vertices | V  IP simplices among vertices of P*
       and equations                | P  IP simplices among points of P*
  d  points of P-dual               |      (with 1<=codim<=# when # is set)
       (only if P reflexive)        | Z  lattice quotients for IP simplices
  a  all of the above except h,f    | #  #=1,2,3  fibers spanned by IP
  l  LG-`Hodge numbers' from single |      simplices with codim<=#
       weight input                 | ## ##=11,22,33,(12,23): all (fibered)
  r  ignore non-reflexive input     |      fibers with specified codim(s)
  D  dual polytope as input (ref only) |   when combined: ### = (##)#
Input:    degrees and weights `d1 w11 w12 ... d2 w21 w22 ...'
          or `d np' or `np d' (d=Dimension, np=#[points]) and
            (after newline) np*d coordinates
Output:   as specified by options
```

Thus, if we want to find out about the vertices and equations of the hyperplanes bounding a polytope, we can use *poly.x* and specify the options "-ve". If no input file is specified the program asks for input, either in the CWS form of degrees and weights (see below), or in terms of a matrix of points whose convex hull determines the polytope. The program first reads the size of the matrix (number of lines and number of columns) and interprets the smaller of the two as the dimension. The matrix can therefore be given as a line of column vectors or as its transposed, a column of line vectors:

```
$ poly.x -ve
Degrees and weights  `d1 w11 w12 ... d2 w21 w22 ...'
  or `#lines #colums' (= `PolyDim #Points' or `#Points PolyDim'):
3 4
Type the 12 coordinates as dim=3 lines with #pts=4 colums:
1 0 0 0
0 1 0 0
0 0 1 0
3 4  Vertices of P
    1    0    0    0
    0    1    0    0
    0    0    1    0
4 3  Equations of P
  -1   -1   -1      1
```

```
      0    0    1      0
      0    1    0      0
      1    0    0      0
```

The resulting facet equations are given as lines "$a_1 \ldots a_n \ c$" normalized such that the a_i have no common divisor and the inequalities $\vec{a} \cdot \vec{x} + c \geq 0$ are fulfilled for all points of P. If the inhomogeneous terms c are positive for all equations then the origin is in the interior of P. In that case we say that P has the 'IP property'. For such an 'IP polytope' the dual polyhedron P^* is, by definition, the convex hull of its vertices $\vec{a}/c$.

Reflexivity of a lattice polytope P is equivalent to P^* being a lattice polytope, i.e. to $c = 1$ for all facet equations. In that situation the lines $\vec{a}$ can be interpreted as vertices of P^* and *poly.x* omits the numbers c, indicating that the resulting matrix can be interpreted as vertices of the dual polytope (in the transposed input format). As an example we consider

```
$ poly.x -e
Degrees and weights  'd1 w11 w12 ... d2 w21 w22 ...'
  or '#lines #colums' (= 'PolyDim #Points' or '#Points PolyDim'):
3 4
Type the 12 coordinates as dim=3 lines with #pts=4 colums:
 3 -1 -1 -1
-1  3 -1 -1
-1 -1  3 -1
4 3  Vertices of P-dual <-> Equations of P
    1   0   0
    0   1   0
    0   0   1
   -1  -1  -1
```

While the omission of $c = 1$ in the reflexive case may be confusing in the beginning, the advantage is that the resulting output can directly be used as input of a calculation for the dual polytope.

This example brings us to the second input format. A simple way to obtain lattice polytopes is as the set of non-negative solutions $\vec{X} \in \mathbb{Z}^N$ to a system of k linear diophantine equations $\vec{w}_j \cdot \vec{X} = d_j$ with non-negative $(w_j)_i = w_{ji}$ and $\sum_j w_{ji} > 0$ for all i. If these equations are independent, the resulting polytope has support in the sublattice of codimension k in $\mathbb{Z}^N$ defined by the equations. Restricting the allowed equations to the case $d_j = \sum_i w_{ji}$ implies the existence of at most one interior point, namely $\vec{X}$ with $X_i = 1$, which is always a solution to this system of equations. By shifting this generic solution to the origin and working with the points $x_l = \sum_i B_l^i (X_i - 1)$ transformed to some basis of the $N - k$ dimensional lattice that supports the polyhedron, we obtain a polytope whose interior point (if it has one) is at the origin. This construction provides our alternative input format in terms of lists of numbers $d_1 \ w_{11} \ \ldots \ w_{1N} \ d_2 \ w_{21} \ \ldots \ w_{kN}$, which we call (combined) weight systems (for $k > 1$) or (C)WS. It can be shown that all reflexive polytopes are subpolytopes of polytopes of this kind [7]. While it may require some exercise to feel comfortable with this format, it is easy to see that our last example is the simplex defined by the weight system "4 1 1 1 1". An example of a combined weight system is "2 1 1 0 0 2 0 0 1 1", which describes a square. More details can be found in ref. [8].

We now give a description of all the options of *poly.x* that is more detailed than the one in the help screen. Note that if no option is specified the program by default chooses the generic option "-g" (not recommended for large dimensions; see below).

-h   The help screen is displayed.

-f   The filter flag switches off the prompt for the input data also in case where input is read from the terminal (i.e. when no input file is given). This is useful for building pipelines.

-g   The following data is displayed. First the numbers $\#p$ and $\#v$ of lattice points and vertices, respectively, in the format "M: $\#p$ $\#v$". The remaining output depends on whether P is reflexive. In this case, the numbers $\#d$ and of $\#e$ of dual lattice points and vertices are displayed in the format "N: $\#d$ $\#e$", followed by information on the Hodge numbers of the corresponding Calabi–Yau manifold (in the case of a three dimensional polytope corresponding to a K3 surface, where the Hodge numbers are determined anyway, information on the Picard number and the 'correction terms' is given instead). Otherwise the number $\#e$ of facets is shown as "F: $\#e$". Using this option implies the completion of the set of lattice points in the convex hull ("points" in the help screen always means "lattice points"). In the reflexive case it also leads to the completion of the dual polytope and the computation of the complete incidence structure which is required for the calculation of the Hodge numbers. For large dimensions (say, beyond 9) these tasks may result in an extremely long response time or in a crash of the program. In such a case, therefore, it is strongly advisable to use other options, e.g. -nve, if informations on the number of lattice points or Hodge numbers are not required.

-p   The lattice points of the polytope are displayed.

-v   The vertices of the polytope are displayed.

-e   The equations of the hyperplanes bounding the polytope are displayed.

-m   One gets the $n_v \times n_e$ matrix with entries $\vec{a}_j \cdot \vec{v}_i + c_j$, $1 \leq i \leq n_v$, $1 \leq j \leq n_e$, where n_v, n_e are the numbers of vertices and equations, respectively. The elements of this 'pairing matrix' represent the lattice distances between the respective vertices and facets.

-d   If the polytope is reflexive the lattice points of the dual polytope are displayed.

-a   This is a shortcut for "-gpvemd"; it can be combined with any other options.

-l   This option is relevant for applications in the context of Landau-Ginzburg models and will be explained in section 4.

-r   Any input that does not correspond to a reflexive polytope will be ignored. This is useful for filtering out reflexive polytopes from a larger list and saves calculation time if one is interested only in reflexive polytopes.

-D   The input is regarded as the dual polytope. As this makes sense only in the reflexive case there is an error message (but no exit from the program) for non-reflexive input. This option is useful, in particular, if one wants to have control over the order of the points in the N lattice.

-n   The completion of the set of lattice points is suppressed. Hence the Hodge numbers cannot be calculated and the output will look like the one for non-reflexive polytopes even in the reflexive case. In particular, if the input is not of the (C)WS type, the number of points may be displayed wrongly. For large dimensional polytopes this option saves a lot of calculation time.

-i   Information on the incidence structure is displayed.

-s   This option refers to the following property of (combined) weight systems. As we saw above, in the higher dimensional embedding defined by a weight system, the polytope is bounded by the inequalities $X_i \geq 0$. We say that the polytope has the span property if the pullbacks of the equations $X_i = 0$ to the subspace carrying the polytope are spanned by vertices of the polytope, i.e. if these equations correspond to facets. With "-s" a message is given if the (combined) weight systems does not have this property.

-I   There is a message if the polytope does not have the origin of the coordinate system in its interior.

-S   The output contains the following two numbers. The first is the number of lattice automorphisms (elements of $GL(n, \mathbb{Z})$) that leave the polytope invariant; note that each such automorphism acts as a permutation on the set of vertices. The second one is the number of permutations of the set of vertices that leaves the vertex pairing matrix invariant (after taking into account the induced permutations of facets). This number can also be interpreted as the number of automorphisms of the lattice generated by the vertices; it may be larger than the number of symmetries in the given lattice.

-T   A coordinate change is performed that makes the matrix of coordinates of the points specified in the input upper triangular, with minimal entries above the diagonal. This may be useful for representing the polytope in a specific lattice basis consisting of points of P, or for finding the volume of a specific cone (if the generators of the cone are the first input points, the volume will be the product of the entries in the diagonal after the transformation).

-N   This option leads to the calculation of a normal form of the polytope, i.e. a matrix containing the vertices in a specific order in a particular coordinate system, such that this output is the same for any two polytopes related by a lattice automorphism. This is useful, for example, if two polytopes are suspected to be isomorphic because they are isomorphic if and only if their normal forms are identical.

-t   The calculation of the normal form involves determining the pairing matrix, a normal form for the pairing matrix, an analysis of which symmetries leave this normal form invariant, a preferred ordering of the vertices and a conversion of the resulting vertex coordinate matrix to upper triangular form. The results of these steps, which may provide further useful information on the structure of P, are displayed.

-V   The IP simplices whose vertices are also vertices of the dual polytope are displayed (an IP simplex is a possibly lower dimensional simplex with the lattice origin in its relative interior; for more information see, e.g., [8]).

-P   The IP simplices whose vertices are lattice points of the dual polytope are displayed. This option should only be used if it is fairly clear that the dual polytope does not have too many lattice points.

-Z This option only has an effect if combined with V or P. Then the program computes a quotient action that defines the sublattice of the intersection with the N lattice with the respective linear subspace that is spanned by the vertices of an IP simplex.

-1, -2, -3, -11, -22, -33, -12, -23 Information on fibration structures of the Calabi–Yau manifold corresponding to a reflexive polytope is displayed. These structures correspond to reflexive subpolytopes of the dual (N lattice) polytope P^* that are intersections of the dual polytope with a linear subspace of $N_{\mathbb{R}}$; see, e.g., [8–10]. As this is a time consuming task and the desired output format may vary, we have programmed two different versions.

If a single number $\# \in \{1, 2, 3\}$ is specified, the intersections of P^* with all linear subspaces spanned by IP simplices are checked for reflexivity. This misses fibrations whose corresponding subspaces are only spanned by a combination of two or more IP simplices. The codimension of the IP simplices is restricted to $1 \leq \text{codim} \leq \#$ (in contrast to our usual policy this option has a side effect on P if combined with that option).

If two numbers are specified, all fibration structures of the given type are computed. For 11, 22 and 33 all reflexive sections of codimension 1, 2 and 3, respectively, are constructed. For 12 and 23 all reflexive subpolytopes of codimension 1 and 2 that themselves contain a reflexive subpolytope with relative codimension 1 are constructed. The output is a polytope in the N lattice whose choice of bases and whose order of points reflects the fibration structure.

Finally we turn to the the question of what can be done if something goes wrong in the application of *poly.x*. We have designed our package in such a way that it should exit with an error message rather than crash or display wrong results. The two main sources for possible problems are inappropriately set parameters and numerical overflows. The most important settings of parameters all occur at the beginning of *Global.h* which is described in more detail in section 5.

Here are some typical error messages. If we want to analyze the Calabi–Yau sixfold that is a hypersurface in $\mathbb{P}^7$, the following may happen.

```
$ 8 1 1 1 1 1 1 1 1
Please increase POLY_Dmax to at least 7
```

In this case one should edit *Global.h*, setting

```
#define               POLY_Dmax      7      /* max dim of polytope    */
```

and compile again. Similarly the program may ask for changes of other basic parameters, all of which are defined within the first 60 lines of *Global.h*.

In many cases we have implemented checks with the help of the "assert" routine, leading to error messages such as the following.

```
poly.x: Vertex.c:572: int Finish_IP_Check(PolyPointList *, VertexNumList *,
EqList *, CEqList *, INCI *, INCI *): Assertion '_V->nv<32' failed.
Abort
```

In this case one should look up line 572 of *Vertex.c*,

```
assert(_V->nv<VERT_Nmax);
```

This means that the value of `_V->nv` has risen above the value 32 assigned to `VERT_Nmax` in *Global.h* and that the value of `VERT_Nmax` should be changed correspondingly. An assertion failure that does not refer to an inequality involving a parameter or an allocation failure is very likely to point to a numerical overflow.

While the possible problems mentioned above are related to parameters set to values that are too low, excessively high values may also lead to problems. This may result in slowing down the calculation (in particular, calculation time depends very sensitively upon whether `VERT_Nmax` is larger than 64) or in running out of memory which may lead to messages such as `Allocation failure in Find_Equations`. Excessively high parameter values may even cause an overflow of the stack, which often only results in a `Segmentation fault`. On many systems stack overflow is indistinguishable from real program errors and the precise location of the crash would have to be found with a debugger. But first one can try to recompile with, say, a smaller `POINT_Nmax`.

3   Other programs and examples

In addition to *poly.x* our package contains the further main programs *class.x*, *cws.x* and *nef.x*. In the following we give a brief description and examples related to *class.x* and in subsection 3.2 we also mention all we need to know about *cws.x*, which creates weight systems and combined weight systems. *nef.x* (by Erwin Riegler) will be discussed briefly in the second paragraph of section 4.

The program *class.x* provides an implementation of the algorithm for the classification of reflexive polyhedra that is described in refs. [7,8] (except for the very first steps of the algorithm which are implemented in *cws.x*). In contrast to *poly.x* this requires the program to work not on one polyhedron at a time but rather on a set of polyhedra of the same dimension, typically related by inclusion (possibly on sublattices). This task required the definition of new data and I/O structures. In particular, because of the huge number of reflexive polytopes in 4 dimensions, we had to implement some data compression and a corresponding binary I/O format. Moreover, since the data exceeded 2GB even in their compressed form, it was necessary to devise a data base structure for storing the data in a set of smaller files on the hard disk in a sorted form, with the top of a binary tree kept in the program memory to enable a fast search for individual polytopes. Fortunately, the user does not need to know anything about this except that the relevant file names have to be provided in combination with the respective flags.

Because of the potentially extremely large data that have to be processed the problem of stack overflow is more serious for *class.x*. While *poly.x* can easily cope with polytopes that have serveral million points, it does not make sense to consider the set of subpolytopes of such a huge polytope, and hence *class.x* should be compiled with a more moderate setting of the relevant parameters.

With the help option of *class.x* detailed information on the various items is directly provided by the executable on typing the letter for the respective option.

```
$ class.x -h
This is  'class.x', a program for classifying reflexive polytopes
Usage:     class.x  [options] [ascii-input-file [ascii-output-file]]
Options:   -h          print this information
           -f or -     use as filter; otherwise parameters denote I/O files
           -m*         various types of minimality checks (* ... lvra)
           -p* NAME    specification of a binary I/O file (* ... ioas)
           -d* NAME    specification of a binary I/O database (DB) (* ... ios)
           -r          recover: file=po-file.aux, use same pi-file
           -o[#]       original lattice [omit up to # points] only
           -s*         subpolytopes on various sublattices (* ... vphm)
           -k          keep some of the vertices
           -c          check consistency of binary file or DB
           -M[M]       print missing mirrors to ascii-output
           -a[2b]      create binary file from ascii-input
           -b[2a]      ascii-output from binary file or DB
           -H*         applications related to Hodge number DBs (* ...cstfe)

    Type one of [m,p,d,r,o,s,k,c,M,a,b,H] for help on options,
    'g' for general help, 'I' for general information on I/O or 'e' to exit:
```

Rather than duplicating that information we therefore proceed with two examples that illustrate how a combination of *poly.x* and *class.x* can be used to accomplish certain goals. Further examples are provided in the next section on applications, which however uses notions from toric geometry and conformal field theory more freely.

3.1   A search for maximal reflexive subpolytopes

In our first example we consider the weight system "7 1 1 1 1 1 2" which describes a non-reflexive polytope and construct its largest reflexive subpolytope.[2] We first put the weight system on a file and compute the generic data that are obtained with the option "-g" (this is the default if no option is specified):

```
$ echo "7 1 1 1 1 1 2" > w.in
$ poly.x w.in
7 1 1 1 1 1 2 M:496 10 F:7
```

The polyhedron thus has 496 points, 10 vertices and 7 facets. It is not reflexive, because then the output line would contain the number of dual points and vertices and the Hodge data instead of the "F" information.

Next we use the program *class.x* to search for reflexive subpolytopes:

```
$ class.x -o5 -po zw.5 w.in
```

---

[2] In geometrical terms this corresponds to a refexive blow-up of a weighted projective space, which in this case admits a transversal Calabi-Yau hypersurface whose Newton polytope is not reflexive. This can only occur in 5 or more dimensions.

```
        rec-dep<=5
        0kR-0 0MB 0kIP 0kNF-0k 491_491 v14r10 f9r7 4b7 0s 0u 0n
        7 1 1 1 1 1 2 R=1 +0sl hit=0 IP=5837 NF=1 (0)
        Writing zw.5: 1+0sl 0m+0s 7b done: 0s
```

The option "-o5" requests that up to (recursion depth) 5 points are omitted in searching for reflexive subpolytopes. The program tells us (some diagnostic information and) that it found one reflexive subpolytope. In order to complete its task it had to check 5837 polytopes for the existence of an interior point. The result is written in compressed binary format onto the file "zw.5". In order to get it in readable form we need another run of *class.x*:

```
$ class.x -b2a -pi zw.5
5 10
1 1 1 1 1 -1 -1 -1 -1 -1
0 3 0 0 0 -2 -2 5 -2 -2
0 0 3 0 0 -2 -2 -2 5 -2
0 0 0 3 0 -2 -2 -2 -2 5
0 0 0 0 3 5 -2 -2 -2 -2
np=1+0sl  5d  1v<=7 n<=7  1nv  1 0 1 7  0 0 0 0
```

(the last line is statistical information on the content of the binary file, which we ignore). More information on this polytope can be obtained with *poly.x*:

```
$ class.x -b2a -pi zw.5 | poly.x -fg
M:491 10 N:8 7 H:2,0,450 [2760]
```

so indeed we found a reflexive polytope that has 5 points less.[3] The structure of that polytope can be analyzed further by computing the IP simplices, i.e. the linear relations among the vertices of the dual polytope

```
$ class.x -b2a -pi zw.5 | poly.x -fV
5 7  vertices of P-dual and IP-simplices
    1   -1   -1   -1   -1    2   -1
    0    0    0    1    0   -1    0
    0    0    1    0    0   -1    0
    0    1    0    0    0   -1    0
    0    0    0    0    1   -1    0
--------------------------------  #IP-simp=2
    2    1    1    1    1    1    0    7=d  codim=0
    1    0    0    0    0    0    1    2=d  codim=4
```

We thus recover the original weight system and an additional vertex that is located opposite to the first vertex. By displaying the facet equations with "poly.x -e" one can see that the facet corresponding to this new dual vertex is parallel to the facet at distance 2 that spoiled reflexivity of our starting polyhedron, but now it is at distance 1. The IP simplices tell us that the maximal reflexive subpolytope can actually be described by a combined weight system:

---

[3] The Hodge numbers of the corresponding Calabi–Yau 4-fold are $h_{11} = 2$, $h_{12} = 0$, $h_{13} = 450$, and the Euler number is 2760.

```
$ echo "7 2 1 1 1 1 1 0  2 1 0 0 0 0 0 1" | poly.x -fg
7 2 1 1 1 1 1 0  2 1 0 0 0 0 0 1 M:491 10 N:8 7 H:2,0,450 [2760]
```

Remaining doubts about the equality of these polytopes could be eliminated by comparing the normal forms that are computed by "poly.x -N".

3.2   Classification of reflexive polytopes in three dimensions

We now want to know how to use our programs for classifying all reflexive polytopes in three dimensions [8, 11]. As a first step we have to find the relevant weight systems with 4 weights. The algorithm for finding a finite set of candidates for such weight systems was found in [12] and is also described in [8]. It is implemented in the program *cws.x*, which creates weight systems with the option "-w" and combined weight systems with the option "-c" (help is available with "cws.x -h"). Extra information on the resulting (C)WS can be obtained by using *class.x* with the option "-ma" (*m* for minimality, *a* for all types of minimality checks), which is designed in such a way that it returns only those (C)WS that have the IP property, together with information on the minimality type. So the first steps are

```
$ cws.x -w3 > ws.3d
$ cws.x -c3 > cws.3d
$ cat cws.3d ws.3d | class.x -ma -f > wK3.ma
```

This leads to the complete list of 116 IP (C)WS for $d = 3$:

```
$ wc -l wK3.ma
    116 wK3.ma
```

Every line in the resulting file contains extra information on the minimality type which can be used to extract the 15 r-minimal (C)WS:

```
$ grep r wK3.ma > wK3.r
$ wc -l wK3.r
     15 wK3.r
```

Now we are ready to use *class.x* for the construction of all subpolytopes of these 15 polytopes:

```
$ class.x -po zK3 wK3.r
...
Writing zK3: 4318+0sl 2119m+79s 11549b  u60 done: 0s
```

This means that *class.x* has found 4318 subpolytopes on the original lattices and no subpolytope that is reflexive only on a sublattice but not on the original one. The 4318 subpolytopes are encoded in the binary file *zK3*. We can convert this file into a database,

```
$ class.x -pi zK3 -do zzK3
Read zK3 (4318poly +0sl 11549b)  write zzK3.* (11 files)  done (1s)
$ ls zz*
zzK3.info  zzK3.v05  zzK3.v07  zzK3.v09  zzK3.v11  zzK3.v13
zzK3.v04  zzK3.v06  zzK3.v08  zzK3.v10  zzK3.v12
```

and look for polytopes in this database that are reflexive on a sublattice such that the sublattice-polytope is not in the database.

```
$ class.x -sv -di zzK3 -po zzK3.s
...
Writing zzK3.s: 1+0sl 0m+0s 3b done: 0s
```

There is precisely one such polytope (written to *zzK3.s*) and we convert it to ascii-format by using the "-b[2a]" ("binary to ascii") option of *class.x*:

```
$ class.x -b -pi zzK3.s
3 4
1 1 1 -3
0 2 0 -2
0 0 4 -4
```

By computing the IP simplices among the vertices of the dual polytope and the group action that defines the sublattice spanned by these vertices

```
$ class.x -b -pi zzK3.s | poly.x -fVZ
3 4  vertices of P-dual and IP-simplices
    3   -1   -1   -1
   -2    2    0    0
   -1    0    1    0
-------------------   #IP-simp=1
    1    1    1    1    4=d  codim=0 /Z2: 1 0 1 0
```

we see that the 4319*th* polytope is a $\mathbb{Z}_2$ quotient of the "quartic". Our program, actually, accepts an extended version of the CWS input, where a group action for selecting a sublattice can be added in the format produced by the option Z:

```
$ echo "4 1 1 1 1 /Z2: 1 0 1 0" | poly.x -fgN
4 1 1 1 1 /Z2: 1 0 1 0 M:19 4 N:7 4 Pic:9 Cor:6
3 4  Normal form of vertices of P
    1    1    1   -3
    0    2    0   -2
    0    0    4   -4
```

The normal form indeed coincides with the one extracted from the file zzK3.s above.

A complete list of all 4319 polytopes in ascii format can be generated with the following steps:

```
$ class.x -pi zK3 -pa zzK3.s -po zK3.all
Data on zK3:  4318+0sl  11825b  (3d)
Data on zzK3.s:  1+0sl  51b  (3d)
SL: 0nf 0sm 0nm 0b ->
d=3 v10 v<=13 n<=10 vn43  2199 79 0 11549  0 0 0 0
Writing zK3.all: 4319+0sl 2120m+79s 11549b  [p^2/2m=0M]
$ class.x -b -pi zK3.all > zK3.all.as
```

With the data "2120m+79s" in the last output line mirror symmetry of the result is easily checked, as twice the number 2120 of mirror pairs and the number 79 of selfdual polytopes add up to 4319.

4   Some applications in toric geometry and conformal field theory

In the present section we present some further examples and results that we produced while preparing the package for publication. In particular we analyzed the list of 5-dimensional Newton polytopes of the transversal weight systems for Calabi–Yau 4-folds that were classified in [13]. This set contains polytopes with up to $355\,785$ points and with degrees (and coordinates) as large as $6\,521\,466$, and thus provided stringent tests on the numerical quality of our algorithms.[4] We discuss some new results on weight systems and what kind of data for superconformal field theories (Landau Ginzburg models) can be computed with *poly.x*. Then we give some examples of free quotients and fibration structures in toric Calabi–Yau hypersurfaces.

The package PALP also contains procedures that construct mirror pairs of toric complete intersections by enumerating nef partitions of a reflexive polytope [14,15] and by evaluating the Hodge data of the corresponding manifolds [16–18]. Since this application, written by Erwin Riegler, is still in an experimental state, we do not provide any documentation. It is, however, already quite reliable and the information that is provided by the help option of the executable *nef.x* should be sufficient at least for the computation of some basic data.

4.1   Landau–Ginzburg models and weighted projective spaces

The first large list of Calabi–Yau manifolds with approximate mirror symmetry consisted of hypersurfaces in weighted projective spaces [19]. To avoid singularities that are not inherited from the ambient space one requires transversality of the polynomial equations that define the varieties. Together with the Calabi–Yau condition $d = \sum \omega_i$ this makes the set of appropriate weight systems finite [20–22] .

The same quasi-homogeneus polynomials can be used as superpotentials of Landau Ginzburg models [23], and it turns out that the Hodge data of the manifold correspond to the charge

---

[4] For these polytopes, single precision (i.e. "Long" set to 32 bit integers) suffices until the square of the degree of the weight system approaches 10^{31}. The only place where 64 bit are required much earlier is the routine "EEV_To_Equation" in Vertex.c, which combines the intersection of two hyperlane equations and a newly found Vertex to a new hyperplane equation. This problem cannot be solved by an improvement of the algorithms and therefore some intermediate numbers in that routine are always computed with maximal precision.

degeneracies of the chiral ring of the canonical LG orbifold [24, 25]. The basic object in this context is the Poincaré polynomial [23]

$$P(t) = \prod \frac{1 - t^{d - \omega_i}}{1 - t^{\omega_i}},$$ (1)

in terms of which all orbifold data can be computed [24–26].

While it turned out that this class of models is incomplete because of its lack of mirror symmetry (even if orbifolds and discrete torion are included [27, 28]), Batyrev discoverd that the above construction can be generalized to hypersurfaces in toric varieties [29]. He showed that the Calabi–Yau condition translates into reflexivity of the polytopes, in terms of which the toric varieties are defined [30–32], and that the Hodge data can be computed in terms of the incidence data and of the numbers of lattice points of dual pairs of faces [29]. Starting from a weighted projective space, the corresponding toric variety is found by considering the Newton polytope of the most general quasi-homogeneous polynomial of degree $d = \sum \omega_i$. That polytope is automatically reflexive in up to 4 dimensions [12]. (The simplest counterexample in 5 dimension was the subject of our example in section 3.1.)

In a certain aspect, on the other hand, the Landau Ginzburg models are more general. The resulting superconformal field theories can be used for constructing consistent string vacua even if the condition of vanishing canonical class is violated provided that the central charge $c/3 = \sum_i (1 - 2\omega_i/d)$ is equal to the complex dimension of the "internal space". Accordingly, the program *poly.x* admits a different kind of input when called with the Landau–Ginzburg option "-l". While the input is restricted to single weight systems, the degree need not be equal to the sum of the weights (moreover, the degree may be given as the last instead of the first number). If the resulting value $c/3$ for the central charge is an integer, the analogue of the Hodge numbers is computed. Otherwise only the data of the Newton polytope and the central charge are evaluated:

```
$ echo "3 1 1 1 1 1" | poly.x -flg
1 1 1 1 1 3=d M:35 5 F:5 LG: c/3=5/3
$ echo "3 1 1 1 1 1 1" | poly.x -flg
1 1 1 1 1 1 3=d M:56 6 F:6 LG: H0:1,0,1 H1:0,20 H2:1
```

In the second case, the complete Hodge diamond is computed but only part of it is displayed because the rest is fixed by duality under the Hodge star operation; the output format for $c/3 = n \in \mathbb{N}$ is `LG: H0:`$h_{00}, \ldots, h_{0n}$ `H1:`$h_{10}, \ldots, h_{1,n-1}$ $\ldots$ `Hn:`h_{nn}.

If the Newton polytope is reflexive, Vafa's formula for the CFT data is compared with the geometrical data of the corresponding Calabi–Yau variety. If there were a discrepancy the program would exit with a diagnostic message. The fact that this never happend, even for the 252 933 reflexive among the 1 100 055 transversal weight systems for Calabi–Yau 4-folds, is quite non-trivial. This coincidence, while natural, is not proven nor really understood. From the computational point of view Vafa's formula becomes quite slow at large degrees because the evaluation of the Poincaré polynomial requires a division of polynomials whose degree ranges up to 26 million in the case of 4-folds. Accordingly,

```
$ echo "6521466 1805 1806 151662 931638 2173822 3260733" | poly.64 -lf

1805 1806 ... 3260733 6521466=d M:355 6 N:355785 6 V:303148,0,252 [1820448]
```

takes almost 10 minutes on an AMD XP1900 processor, while

```
$ echo "6521466 1805 1806 151662 931638 2173822 3260733" | poly.64 -f
6521466 1805 1806 ... 3260733 M:355 6 N:355785 6 H:303148,0,252 [1820448]
```

is finished in 8 seconds. With the "-l" flag the Hodge data are preceded by a "V:" to indicate that the numbers were computed with Vafa's formulas (and compared with Batyrev's in the reflexive case, indicated by the "N:", while a combination of "F:" and "V:" characterizes a nonreflexive transversal weight system); moreover the degree of the weight system is written after the weights because of the Landau–Ginzburg flag.

As one can verify with *class.x*, distinct weighted projective spaces need not lead to different reflexive polytopes. With

```
$ cws.x -w4 -t > tws.4d
```

we create, as an example, a file containing all 7555 transversal weight systems for 4-dimensional Newton polyhedra (this takes a few hours, but the data can also be fetched from our web page [1]). Then

```
$ class.x -a2b -po ztws.4d  tws.4d twsred.4d
...
Writing ztws.4d: 5961+0sl 404m+32s 76243b  u32 done: 0s
```

writes the binary file *ztws.4d* encoding a sorted list of 5961 distinct polytopes and the ascii file *twsred.4d* containing a reduced list of 5961 weight systems for these polytopes. Among the 252,933 reflexive polytopes coming from 5-dimensional weighted projective spaces obeying the transversality condition, 202,746 are different. Similarly, the 184,026 'IP weight systems' with 5 weights correspond only to 58,690 different 4-dimensional polytopes.

4.2   Quotients and fibrations

In toric geometry a refinement of the N lattice, with the corresponding change to a sublattice of the dual M lattice, corresponds to the modding out of an abelian group action. The resulting quotient of a Calabi–Yau hypersurfaces is free, and thus leads to a non-trivial fundamental group, if it does not introduce singularities. This is the case if the refinement of the N lattice does not lead to additional lattice points for the dual polytope $\Delta^* \subset N_{\mathbb{R}}$. With the option "-sp" the program *class.x* searches either polytopes in ascii-format or the content of a data base for such objects.[5] By searching the database of all reflexive 4-dimensional polytopes with

---

[5] In fact a slightly weaker condition is sufficient: additional interior points of facets correspond to divisors that do not intersect the hypersurface. With the option "-sh" *class.x* omits these points when checking for free quotients, but so far we did not find an example where this leads to additional solutions.

```
$ class.x -sp -di zzdb
```

we found that there are 16 free quotiens, for 13 of which the index of the lattice quotient is 2.
The remaining 3 quotients are the well-know free $\mathbb{Z}_5$ quotient of the quintic, a $\mathbb{Z}_3$ quotient of
$\mathbb{P}^2 \times \mathbb{P}^2$, and a $\mathbb{Z}_3$ quotient of weighted $\mathbb{P}^4_{33111}$.

In string theory fibration structures play an important role in dualities. In particular, K3
fibrations are essential for heterotic–type II duality [33–35] and elliptic fibrations are essential
in F-theory [36]. In the context of toric geometry such fibrations correspond to reflexive sections
of codimension 1 or 2 of 4-dimensional polytopes in the N lattice [9, 10, 37]. In the remainder
of this section we show some examples of such fibrations that have a non-trivial fundamental
group.[6] The only polytope in the list of 16 free quotients that does not have any fibration is
the $\mathbb{Z}_5$ quotient. The two $\mathbb{Z}_3$ quotients both are elliptically fibered:

```
$ echo "3 1 1 1 0 0 0  3 0 0 0 1 1 1 /Z3: 0 1 2 0 1 2"| poly.x -fg2P
3 1 1 1 0 0 0  3 0 0 0 1 1 1 /Z3: 0 1 2 0 1 2 M:34 9 N:7 6 H:2,29 [-54]
4 7  points of P-dual and IP-simplices
   -3    0    0    0    0    3    0
    1    1    0    0    0   -2    0
    1    0    0   -1    1   -1    0
    2    0    1   -1    0   -2    0
----------------------------      #IP-simp=2
    1    1    0    0    0    1   3=d  codim=2
    0    0    1    1    1    0   3=d  codim=2
----------------------------      #fibrations=2
    v    v    _    _    _    v  cd=2  m:10 3 n:4 3
    _    _    v    v    v    _  cd=2  m:10 3 n:4 3
$ echo "9 3 3 1 1 1 /Z3: 1 2 1 2 0"| poly.x -fg22PZ
9 3 3 1 1 1 /Z3: 1 2 1 2 0 M:49 5 N:7 5 H:2,38 [-72]
4 7  points of P-dual and IP-simplices
   -3    0    3    0    0    0    0
    1    0   -2    0    3    1    0
    1    0   -1    1   -1    0    0
    2    1   -2    0   -1    0    0
----------------------------      #IP-simp=2
    3    1    3    1    1    0   9=d  codim=0 /Z3: 1 2 2 1 0 0
    1    0    1    0    0    1   3=d  codim=2
4 6  m:10 3 n:4 3  M:49 5 N:7 5  p=025134
    1   -2    1    0    0    3
    0    1   -1   -1    0   -2
    0    0    0    0    3   -3
    0    0    0   -1   -1    2
```

In the first case we used IP simplices to search for fibrations of codim≤ 2. In the second case
we can be sure that all elliptic fibrations were found because of the 22 flag; the upper-left 2x3
block of the coordinate matrix describes the boundary points of the fiber polytope in the N
lattice, whose dual has 10 points and 3 vertices (the information on the fiber polytopes is given

---

[6] It turned out that these quotient spaces admit an elliptic K3 fibration if and only if the group is $\mathbb{Z}_2$; this
fact might find an explanation in the context of heterotic–type II duality [38].

by "m:10 3 n:4 3"). The string "p=025134" at the end of the first line of the polytope in the fiber basis describes the permutation that brings the points from the original order, to which the IP simplex information refers, into an order where the points of the fiber come first; as usual in C, counting begins with 0.

As our last example we consider a polytope with different elliptic K3 fibrations, where none of the K3 fibers is spanned by a single IP simplex:

```
$ poly.x -12PD
Degrees and weights  'd1 w11 w12 ... d2 w21 w22 ...'
  or '#lines #colums' (= 'PolyDim #Points' or '#Points PolyDim'):
4 7
Type the 28 coordinates as dim=4 lines with #pts=7 colums:
   1  -1   0   0   0   1  -1
   0   2  -1   0   2   1  -1
   0   0   0   1  -1   1  -1
   0   0   0   0   0   2  -2
4 9  points of P-dual and IP-simplices
     1   -1    0    0    0    1   -1    0    0
     0    2   -1    0    2    1   -1    1    0
     0    0    0    1   -1    1   -1    0    0
     0    0    0    0    0    2   -2    0    0
---------------------------------------    #IP-simp=4
     1    1    2    0    0    0    0    0  4=d  codim=2
     0    0    2    1    1    0    0    0  4=d  codim=2
     0    0    1    0    0    0    0    1  2=d  codim=3
     0    0    0    0    0    1    1    0  2=d  codim=3
4 8  Em:9 3 n:5 3  Km:35 5 n:7 5  M:53 10 N:9 7  p=01273456
     1    1   -1    1    0    2    0    0
     0    2   -1    1    0    2    1   -1
     0    0    0    0    1   -1    1   -1
     0    0    0    0    0    0    2   -2
4 8  Em:9 3 n:5 3  Km:27 6 n:7 5  M:53 10 N:9 7  p=01275634
     1    1   -1    1    0    0    0    2
     0    2   -1    1    0    0    1    1
     0    0    0    0    1   -1    1   -1
     0    0    0    0    0    0    2   -2
...
4 8  Em:9 4 n:5 4  Km:27 6 n:7 5  M:53 10 N:9 7  p=25673401
     1    0    0   -1    0   -2    1   -3
     0    1   -1    0    0    0    1   -1
     0    0    0    0    1   -1    1   -1
     0    0    0    0    0    0    2   -2
```

The fibers can be checked to be elliptic curves in $\mathbb{P}^1 \times \mathbb{P}^1$ (for "Em:9 4 n:5 4") or in $\mathbb{P}^2_{112}$, and the data of the elliptic K3 surfaces are "M:35 5 N:7 5 Pic:2 Cor:0" or "M:27 6 N:7 5 Pic:4 Cor:1", as one can find out by entering the upper-left 3x6 blocks of K3 coordinates into *poly.x* with flags "-gD".

The dots in the above result indicate 3 further fibrations, with corresponding permutations "p=23470156", "p=23475601", "p=25670134", that can be checked to be related to the

displayed fibrations by a symmetry of the polytope: The IP simplex structure is invariant under the four independent permutations (01), (34), (56) and (03)(14) of order 2, which induce the required relations. Indeed, "poly.x -t" displays the resulting 16 vertex permutations that represent all GLZ-symmetries of the polytope. Note that in this case the double cover

```
$ poly.x -Dg
Degrees and weights  'd1 w11 w12 ... d2 w21 w22 ...'
  or '#lines #colums' (= 'PolyDim #Points' or '#Points PolyDim'):
4 7
Type the 28 coordinates as dim=4 lines with #pts=7 colums:
   1  -1   0   0   0   1  -1
   0   2  -1   0   2   1  -1
   0   0   0   1  -1   1  -1
   0   0   0   0   0   1  -1
M:105 10 N:9 7 H:5,85 [-160]
```

has a larger Picard number $h_{11} = 5$, which indicates that not all divisors of the double cover can be toric. The number of points in the N lattice is the same and the Euler number doubles, as it must be for a free quotient.

5   Structure of the package

In addition to *Makefile* which controls the compilation of executables, our package consists of program files with extension *.c* and header files (extension *.h*).

Makefile contains various different sets of options which are optimized for different types of compilers (including GNU C++, Silicon Graphics and the DEC Alpha compilers). For elaborate applications it may certainly be useful to experiment with some of the options. Optimization level -O3 may result in faulty compilation (we included several work-arounds for the GNU compiler and for the Silicon Graphics), while -O2 is usually safe and only marginally less effective.

The files in our package with extension *.c* come in two varieties: the files starting with lower case letters (i.e. *class.c*, *cws.c*, *nef.c*, *poly.c*) correspond to main programs whereas all other *.c*–files start with capitals and contain routines used by the main programs.

Coord.c, *Polynf.c* and *Vertex.c* are used by all main programs. Definitions and routines from these files that are globally available are all listed in the header file *Global.h*, which is probably the most interesting file for anyone trying to understand the structure of our package or to do some programming with our routines. As there is a lot of documentation within this file, we describe it here only briefly and recommend also to study a printout of it.

Global.h starts with basic parameter settings. This part is important not only for programmers but also for users: none of the parameters should be chosen too small, but choosing them too large may result in bad performance or even a crash due to lack of memory. Usually it should be sufficient to make sure that the definition of POLY_Dmax (the maximal dimension of a polytope to be analyzed) matches the application, because *Global.h* contains default settings for other relevant parameters depending on POLY_Dmax. For POLY_Dmax ≤ 4 these settings are

such that every reflexive polytope fulfills them. Next there are most of the type definitions that are used globally in our package. Finally, the globally available routines from *Coord.c*, *Polynf.c* and *Vertex.c* are listed.

Coord.c contains routines mainly connected with I/O; these are routines for reading coordinates (or CWS and converting these to coordinates) as well as routines for various types of output. Whenever possible, the output is structured in such a way that it can be used as input for another run of our programs.

Polynf.c contains routines for analyzing the symmetries of a polytope, calculating its normal form, identifying IP simplices, finding fibrations, transforming bases and other advanced tasks.

Vertex.c contains the core routines for handling polytopes. These include basic tasks such as evaluating an equation on a point, comparing points, converting a codimension 2 subspace (defined by two equations) and a vertex into the equation of the hyperplane spanned by them, etc. Next there are the routines of polytope analysis such as determining the vertices and facets of the convex hull of a finite number of lattice points and the computation of the complete set of lattice points of a given polytope; there is also a routine for calculating the Hodge numbers of a Calabi–Yau hypersurface determined by a reflexive polytope. Finally there are routines related to incidence structures. In our programs these structures are represented by bit patterns which are determined either by unsigned (long long) integers or by arrays of unsigned integers. This is useful because many of the bitwise operations provided by C correspond to meaningful operations in terms of the analysis of a polytope (see sec. 3 of [39]).

Subadd.c, *Subdb.c* and *Subpoly.c* are only used by *class.x*. They share a common header file *Subpoly.h*. *Subpoly.c* contains the routines necessary for constructing all reflexive subpolytopes of a given polytope as well as many other routines necessary for the classification of reflexive polytopes. *Subadd.c* and *Subdb.c* both have to do with data handling. In *Subdb.c* there are the routines concerning databases consisting of binary files whereas *Subadd.c* deals with other tasks such as sorting lists of normal forms of polyhedra etc.

LG.c and the corresponding header file *LG.h* are used by *poly.x* and *cws.x*, but not by *class.x*. *LG.c* contains the routines necessary for a CFT type analysis as described in section 4. *nef.x* uses routines from *Nefpart.c* for creating nef partitions and routines from *E_Poly.c* for Hodge numbers; the corresponding header file is *Nef.h*.

There is also a file *Rat.c* providing routines for dealing with rational numbers and certain tasks concerning manipulations of integers. The corresponding header file *Rat.h* is included wherever these applications are needed.

References

[1] M.Kreuzer, H.Skarke, `http://hep.itp.tuwien.ac.at/~kreuzer/CY.html`

[2] http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html

[3] http://www.geom.umn.edu/software/qhull/

[4] http://www.zib.de/Optimization/Software/Porta/

[5] http://cgm.cs.mcgill.ca/~avis/C/lrs.html

[6] D.Avis, D.Bremner, R.Seidel, *How good are convex hull algorithms?* Computational Geometry: Theory and App. 7 (1997) 265 [http://www.cs.unb.ca/~bremner/papers/abs-hgach-97.ps.gz]

[7] M.Kreuzer, H.Skarke, *On the classification of reflexive polyhedra,* Commun. Math. Phys. **185** (1997) 495 [hep-th/9512204]

[8] M.Kreuzer, H.Skarke, *Reflexive polyhedra, weights and toric Calabi-Yau fibrations,* [math.ag/0001106]

[9] A.Avram, M.Kreuzer, M.Mandelberg, H.Skarke, *Searching for K3 fibrations,* Nucl. Phys. **B494** (1997) 567 [hep-th/9610154]

[10] M.Kreuzer, H.Skarke, *Calabi–Yau fourfolds and toric fibrations,* J. Geom. Phys. **466** (1997) 1 [hep-th/9701175]

[11] M.Kreuzer, H.Skarke, *Classification of reflexive polyhedra in three dimensions,* Adv. Theor. Math. Phys. **2** (1998) 847 [hep-th/9805190]

[12] H. Skarke, *Weight Systems for Toric Calabi–Yau Varieties and Reflexivity of Newton Polyhedra,* Mod. Phys. Lett. **A11** (1996) 1637, alg-geom/9603007.

[13] M. Lynker, R. Schimmrigk, A. Wisskirchen, *Landau-Ginzburg vacua of string, M- and F-theory at c = 12,* Nucl. Phys. **B550** (1999) 123, hep-th/9812195.

[14] V.V.Batyrev, L.A.Borisov, *Dual cones and mirror symmetry for generalized Calabi–Yau manifolds,* in *Mirror Symmetry II* (ed. S.-T. Yau) [alg-geom/9402002].

[15] V.V.Batyrev, L.A.Borisov, *On Calabi-Yau complete intersections in toric varieties,* in *Proceedings of Trento Conference (1994)* [alg-geom/9412017].

[16] V.V.Batyrev, L.A.Borisov, *Mirror duality and string theoretic Hodge numbers* [alg-geom/9509009].

[17] M. Kreuzer, E. Riegler, D. Sahakyan, *Toric complete intersections and weighted projective space,* math.AG/0103214.

[18] M. Kreuzer, *Strings on Calabi-Yau spaces and toric geometry,* Nucl. Phys. Proc. Suppl. **102** (2001) 87, hep-th/0103243.

[19] P.Candelas, M.Lynker, R.Schimmrigk, *Calabi-Yau manifolds in weighted $\mathbb{P}^4$,* Nucl. Phys. **B341** (1990) 383.

[20] M.Kreuzer, H.Skarke, *On the classification of quasihomogeneous functions,* Commun. Math. Phys. **150** (1992) 137, hep-th/9202039.

[21] M.Kreuzer, H.Skarke, *No mirror symmetry in Landau-Ginzburg spectra!,* Nucl. Phys. **B388** (1992) 113 [hep-th/9205004]

[22] A.Klemm, R.Schimmrigk, *Landau–Ginzburg string vacua,* Nucl. Phys. **B411** (1994) 559 [hep-th/9204060]

[23] W.Lerche, C.Vafa, N.Warner, *Chiral rings in N=2 superconformal theories,* Nucl. Phys. **B324** (1989) 427

[24] C.Vafa, *String Vacua and Orbifoldized LG Models,* Mod. Phys. Lett. **A4A** (1989) 1169; Superstring Vacua, HUTP-89/A057 preprint

[25] K.Intriligator and C.Vafa, *Landau–Ginzburg Orbifolds,* Nucl. Phys. **B339** (1990) 95

[26] M.Kreuzer, C.Schweigert, *On the extended Poincaré polynomial,* Phys. Lett. **B352** (1995) 276 [hep-th/9503174]

[27] M.Kreuzer, H.Skarke, *All abelian symmetries of Landau–Ginzburg potentials,* Nucl. Phys. **B405** (1993) 305 [hep-th/9211047]

[28] M.Kreuzer, H.Skarke, *ADE string vacua with discrete torsion,* Phys. Lett. **B318** (1993) 305 [hep-th/9307145]; *Landau-Ginzburg orbifolds with discrete torsion,* Mod. Phys. Lett. **A10** (1995) 1073 [hep-th/9412033]

[29] V.V.Batyrev, *Dual polyhedra and mirror symmetry for Calabi–Yau hypersurfaces in toric varieties,* J. Alg. Geom. **3** (1994) 493 [alg-geom/9310003]

[30] W.Fulton, *Introduction to toric varieties* (Princeton Univ. Press, Princeton 1993)

[31] D. Cox, *Recent developments in toric geometry* [alg-geom/9606016].

[32] D.Cox, S.Katz, *Algebraic geometry and mirror symmetry,* Math. Surveys Monogr. 68 (Amer. Math. Soc., Providence 1999)

[33] A.Klemm, W.Lerche and P.Mayr, *K3 fibrations and heterotic type II string duality,* Phys. Lett. **B357** (1995) 313 [hep-th/9506112].

[34] P.S.Aspinwall and J.Louis, *On the ubiquity of K3 fibrations in string duality,* Phys. Lett. **B369** (1996) 233 [hep-th/9510234]

[35] P.Candelas, A.Font, *Duality between the webs of heterotic and type II vacua,* Nucl. Phys. **B511** (1998) 295, hep-th/9603170.

[36] C.Vafa, *Evidence for F-theory,* Nucl. Phys. **B469** (1996) 403 [hep-th/9602022]

[37] Y.Hu, C.-H.Liu, S.-T.Yau, *Toric morphisms and fibrations of toric Calabi-Yau hypersurfaces* [math.AG/0010082]

[38] A.Klemm, M.Kreuzer, E.Riegler, E.Scheidegger, work in progress

[39] M.Kreuzer, H.Skarke, *Complete classification of reflexive polyhedra in four dimensions* [hep-th/0002240]