Computing convex hulls and counting integer points with \textsc{polymake}

Benjamin Assarf\textsuperscript{1} · Ewgenij Gawrilow\textsuperscript{2} · Katrin Herr\textsuperscript{3} · Michael Joswig\textsuperscript{1} · Benjamin Lorenz\textsuperscript{1} · Andreas Paffenholz\textsuperscript{3} · Thomas Rehn\textsuperscript{4}

Received: 8 September 2014 / Accepted: 5 April 2016 / Published online: 23 May 2016
© Springer-Verlag Berlin Heidelberg and The Mathematical Programming Society 2016

Abstract  The main purpose of this paper is to report on the state of the art of computing integer hulls and their facets as well as counting lattice points in convex polytopes. Using the \textsc{polymake} system we explore various algorithms and implementations. Our experience in this area is summarized in ten “rules of thumb”.

M. Joswig, B. Lorenz and A. Paffenholz are partially supported by the DFG within the Priority Program 1489. M. Joswig is additionally supported by Einstein Foundation Berlin.

Michael Joswig
Joswig@math.tu-berlin.de
Benjamin Assarf
Assarf@math.tu-berlin.de
Ewgenij Gawrilow
egawrilow@gmail.com
Katrin Herr
herr@mathematik.tu-darmstadt.de
Benjamin Lorenz
Lorenz@math.tu-berlin.de
Andreas Paffenholz
paffenholz@opt.tu-darmstadt.de
Thomas Rehn
thomas.rehn@research.initos.com

1 Institut für Mathematik, TU Berlin, Str. des 17. Juni 136, 10623 Berlin, Germany
2 TomTom International BV, De Ruijterkade 154, 1011 AC Amsterdam, The Netherlands
3 Fachbereich Mathematik, TU Darmstadt, Dolivostr. 15, 64293 Darmstadt, Germany
4 initOS GmbH & Co. KG, Hegelstr. 28, 39104 Magdeburg, Germany
1 Introduction

In integer and linear optimization the software workhorses are solvers for linear programs (based on simplex or interior point methods) as well as generic frameworks for branch-and-bound or branch-and-cut schemes. Comprehensive implementations are available both as Open Source, like SCIP [2], as well as commercial software, like CPLEX [24] and Gurobi [38]. While today it is common to solve linear programs with millions of rows and columns and, moreover, mixed integer linear programs with sometimes hundreds of thousands of rows and columns, big challenges remain. For instance, the 2010 version of the MIPLIB [30] lists the mixed-integer problem liu with 2178 rows, 1156 columns, and a total of only 10,626 non-zero coefficients; this seems to be impossible to solve with current techniques. One way to make progress in the field is to invent new families of cutting planes, either of a general kind or specifically tailored to a class of examples. In the latter situation the strongest possible cuts are obviously those arising from the facets of the (mixed) integer hull. A main purpose of this note is to report on the state of the art of getting at such facets in a brute force kind of way. And we will do so by explaining how our software system polymake [55] can help.

Here we focus on integer linear programming (ILP); mixed integer linear programming (MILP) will only be mentioned in passing. To avoid technical ramifications we assume that all our linear programs (LP) are bounded. The brute force method for obtaining all facets of the integer hull is plain and simple, and it has two steps. First, we compute all the feasible integer points. Since we assumed boundedness these are only finitely many. Second, we compute the facets of their convex hull. Of course, the catch is that neither problem is really easy. Deciding if an ILP has an (integer) feasible point is known to be NP-complete [34]. So, we may not even hope for any efficient algorithm for the first step. Most likely, the situation for the second is about equally bad. While it is open whether or not there is a convex hull algorithm which runs in polynomial time measured in the combined sizes of the input and the output, recent work of Khachiyan et al. [46] indicates a negative answer. They show that computing the vertices of an unbounded polyhedron is hard; the difference to the general convex hull problem is that their result does not say anything about the rays of the polyhedron.

Our paper is organized as follows. We start out with a very brief introduction to the polymake system and its usage. In Sect. 3 we explore how various convex hull algorithms and their implementations behave on various kinds of input. Our input is chosen according to typical scenarios motivated by questions in optimization. Deliberately we picked data within a manageable range; our goal is to give a feel for what can be done within about one hour of CPU time on a current standard desktop machine. For particularly hard convex hull problems see [4,42]. Section 4 is devoted to enumerating lattice points in polytopes, which is actually the first step in the procedure
sketched above. Like for the convex hull computations there are several methods which behave quite differently depending on the input. We would like to stress that all the software systems mentioned have their preferred types of input for which they are (often vastly) superior to the others. There are no globally optimal algorithms known, neither for convex hull computations nor for lattice point enumeration. Even worse, in general, it is very difficult to say which method works best on which input. Only a thorough geometric and combinatorial analysis a posteriori allows for precise statements. Yet we will try to sum up our experience in this area in several “rules of thumb”, all of which have to be taken with a grain of salt. The software systems tested are interfaced to by polymake, and some of them are even shipped with it. All our experiments are run through polymake, which does create very little overhead. The current version of polymake does not directly make use of our rules of thumb. Instead the user needs to pick the best method according to her or his judgment (or rely on some default). We close the paper with additional references to the literature and related experimental results. See the Appendix for details on our experimental setup.

Computers of today and the foreseeable future are faster than their predecessors just because they provide more cores. Hence the parallelization of algorithms and software will play an increasingly important role. Yet here we largely restrict our attention to single-threaded computations in order to not stretch our —already quite involved— test scenarios beyond any manageable limits. As an exception we report on one parallel test with normaliz [19]. The very recent version 6.0 of lrs [3,5] offers single machine shared memory multicore parallelization as well as MPI based multi-machine parallelization. See Avis and Jordan [8] for a computational test of several large examples.

## 2 Example polymake session

In this section we will give a few examples on how one can use polymake for various tasks, e.g., convex and integer hull computations and linear programming. The main application area of polymake are polytopes and polyhedra, but the software can also deal with a number of other kinds of objects. That is why polymake is split into different so-called applications, for instance, for graphs, fans, ideals, matroids, simplicial complexes, or toric varieties. We will only work with the application polytope here, which is indicated by the corresponding prompt polytope> in front of each code line. The polymake system employs a rule based mechanism for deciding which algorithms satisfy user requests; this is explained in [35]. However, one may also force the usage of specific algorithms. Thus, a big advantage of polymake is that the user can use and compare various algorithms through a common interface. Our special object model, which here is used for the polytopes and the linear programs, is the topic of [36]. The software is available from polymake.org [55].

For our first example, a fractional knapsack polytope $P$ is the set of points in the non-negative orthant of, say $\mathbb{R}^d$, which satisfies one extra linear inequality, i.e.,

$$
P = \left\{ x \in \mathbb{R}^d \mid \langle a, x \rangle \leq b, \ x_i \geq 0 \ \text{for all} \ i \right\} \tag{1}$$
for some integral normal vector $a \in \mathbb{Z}^d$ and $b \in \mathbb{Z}$. Such polytopes are easy to construct in polymake. For instance, the command

```
polytope> $k = fractional_knapsack([40,-2,-3,-5,-8,-13]);
```

produces the fractional knapsack polytope for the inequality

$$2x_1 + 3x_2 + 5x_3 + 8x_4 + 13x_5 \leq 40$$

in $\mathbb{R}^5$. If all coefficients are positive, as in our example, the polytope is bounded. In this case the combinatorics is not so exciting, as all fractional knapsack polytopes are simplices. Each line in the following output corresponds to one of the six vertices.

```
polytope> print $k->VERTICES;
1 0 0 0 0 0
1 20 0 0 0 0
1 0 40/3 0 0 0
1 0 0 8 0 0
1 0 0 0 5 0
1 0 0 0 0 40/13
```

First, note that the coordinates contain rational numbers as polymake uses GMP’s arbitrary precision rational arithmetic [31] by default. Second, notice the leading 1 of each vertex. This is due to the fact that polyhedra in polymake are modeled as the intersection of a cone with the affine hyperplane defined by $x_0 = 1$. With this homogenization many algorithms can deal with unbounded polyhedra without much extra effort; e.g., see [44, Sect. 3.4].

At this point polymake already used a (dual) convex hull algorithm to find the vertices of the polytope which was given by an outer description. There are several convex hull implementations to choose from within polymake; see Sect. 3 for more details. This particular fractional knapsack polytope belongs to the Fibonacci fractional knapsack polytopes. Their integer hulls, the Fibonacci knapsack polytopes, will be investigated in Sect. 3.4 below.

Now we want to present how to optimize over this polytope with polymake. Like for the convex hull algorithms, there are various (simplex type) implementations for linear programming available. The first step is to create a LinearProgram object with an objective function. Then we can ask for the maximal/minimal value and an optimal solution.

```
polytope> $lp = $k->LP(LINEAR_OBJECTIVE=>new Vector ([0,1,2,1,2,1]));
polytope> print $lp->MAXIMAL_VERTEX;
1 0 0 0 0 0
polytope> print $lp->MAXIMAL_VALUE;
80/3
```

Observe that the maximal vertex is not integral. To find an integral optimal solution one can repeat this process with the integer hull of that polytope.

```
polytope> $ik = integer_hull($k);
polytope> print $ik->N_POINTS;
1366
polytope> print $ik->N_VERTICES;
16
```
Computing convex hulls and counting integer points with polymake

Fig. 1 Visualization of the two example polytopes, made with polymake’s interface to TikZ [60]. a Visualization of the integer points of the projection of $\mathbb{R}^k$ onto the first three coordinates with a maximal linear and a maximal integer solution. b The Klee-Minty-Cube in dimension 3 with $t = \frac{1}{4}$

Here polymake enumerated all integral points inside the polytope. For this task there are again several different algorithms available, see Sect. 4. Afterwards we define a new polytope with all those integral points as input points, and which is stored in $\mathbb{R}^k$. Notice that polymake also provides a function mixed_integer_hull with the obvious interpretation.

Solving an integer linear program is the same as doing linear optimization over the integer hull. So we can proceed as before. A projection of this Fibonacci polytope, the integral points, and the optimal vertices is shown in Fig. 1a.

```
polytope> $ilp = $ik->LP(LINEAR_OBJECTIVE=>new Vector([0,1,2,1,2,1]));
polytope> print $ilp->MAXIMAL_VERTEX;
121 2000
polytope> print $ilp->MAXIMAL_VALUE;
26
```

As already mentioned, polymake usually works with exact rational coordinates. However, it is also possible to work with other coordinate types, which includes certain extensions of rationals. For instance, fields of Puiseux fractions are relevant in optimization since they can be used to model coordinate perturbations. The (rational) Puiseux fractions form a field whose elements are univariate rational functions with rational coefficients and rational exponents; they can be equipped with a natural ordering. Computing with Puiseux fractions amounts to computing with the usual rational numbers and an infinitesimally small transcendental number; see [43] for the details. We illustrate this with the 3-dimensional Klee–Minty cube [47] which can be constructed as follows.

```
polytope> $monomial=new UniMonomial<Rational,Rational>(1);
polytope> $t=new PuiseuxFraction<Min>($monomial);
polytope> $c = klee_minty_cube(3,$t);
polytope> print_constraints($c);
Facets:
0: x1 >= (0)
1: -x1 >= (-1)
```
2: -(t) x1 + x2 >= (0)
3: -(t) x1 - x2 >= (-1)
4: -(t) x2 + x3 >= (0)
5: -(t) x2 - x3 >= (-1)

The parentheses indicate that the coefficients are Puiseux fractions, rather than rational numbers. Yet it makes sense to contemplate evaluating these expressions for special values. For instance, \( t = 0 \) yields the ordinary \( 0/1 \)-cube; and for \( 0 < t < 1/2 \) we obtain the Klee–Minty cube. Notice that, by construction, the latter inequality always holds in the field of Puiseux fractions. As a benefit from working over Puiseux fractions we can obtain the volume as a rational function in \( t \) (which is a polynomial in this case):

```
polytope> print $c->VOLUME;
(1 -2*t + t^2)
```

In addition to the various applications shipped with \texttt{polymake}, there are also third-party extensions, which add further functionality. Of particular interest in the context of integer linear programming is the \texttt{unimodularity-test} extension by Matthias Walter \cite{64}, which implements an algorithm of Walter and Trümper \cite{65}. Our wiki page at \texttt{polymake.org} contains several tutorials, some of which specialize on topics in optimization. In particular, it features a proof-of-concept implementation of the branch-and-bound algorithm, to solve integer linear programs without relying on computing all lattice points within the feasible domain, which is useful for educational purposes. Via the interface to \texttt{normaliz} \cite{19} our software can also compute Hilbert bases. For instance, this allows for computing Gomory-Chvátal closures by calling the function \texttt{gc_closure}.

\texttt{polymake} also allows to directly read and write data files in the classical \texttt{lp}-format for linear programs and in \texttt{porta} format for convex hull problems in primal and dual form. This can be done with the commands \texttt{lp2poly}, \texttt{poly2lp}, \texttt{poly2porta}, and \texttt{porta2poly}. See the corresponding help text for the correct calling syntax and options (e.g., \texttt{help “lp2poly”}; inside the \texttt{polymake} shell).

The programming language of the \texttt{polymake} shell is essentially \texttt{Perl}. However, we extended the language in several ways. The most important modification is related to our implementation of an interface between \texttt{Perl} and \texttt{C++}, which is based on \texttt{Perl}'s \texttt{XS} interface. For instance, calling the function \texttt{new Vector(...)} creates an object via \texttt{C++} code, and the \texttt{Perl} shell keeps a reference to an opaque object. The \texttt{C++} code has a template parameter for the coefficients which defaults to \texttt{Rational}, which is our wrapper to the \texttt{GMP}. We modified \texttt{Perl} to allow for template parameters in the \texttt{C++} data type. For instance, \texttt{new Vector<Integer>(...)} is legal in \texttt{polymake}. New instantiations of data types due to an uncommon choice of template parameters trigger the automatic compilation of the appropriate \texttt{C++} code. That code is kept such that it does not need to be recompiled again.

### 3 Convex hull computations

The \textit{convex hull problem} in its primal form asks to compute the facets of the convex hull of finitely many given points in \( \mathbb{R}^d \). By cone polarity this is equivalent to computing
the vertices and rays of a polyhedron which is given in terms of finitely many linear equations and inequalities. The latter form is the dual convex hull problem. Both forms occur naturally in the context of linear and integer programming: e.g., to compute the facets of the integer hull of a given polytope is a primal convex hull problem. Dual convex hull problems arise, e.g., in the computation of Voronoi diagrams. Out of the four types of convex hull problems we will investigate two, namely computing the facets of integer hulls and computing Voronoi diagrams. Other applications of convex hull algorithms aim at certain parameterized optimization problems. An example is the search for biologically correct alignments of mRNA data [28].

3.1 Complexity status

The precise complexity status of the convex hull problem is unsettled: It is not known whether or not there exists an algorithm whose complexity is bounded by a polynomial in the combined size of the input and the output. This concept is called polynomial total time; see [62]. As a measure for the complexity of an algorithm it differs from what is common in theoretical computer science, where algorithms are typically measured in the size of the input only. To evaluate convex hull algorithms from a practical point of view restricting the attention to the input size only is somewhat unsatisfying as the size of the output can vary over several orders of magnitude. To give just one basic example: a cube in dimension $d$ has $2^d$ vertices but only $2d$ facets, while the dual cross polytope has $2d$ vertices but as many as $2^d$ facets. By McMullen’s Upper Bound Theorem [66, Sect. 8] the number of facets of a $d$-polytope with $n$ vertices is bounded by $O(n^{d/2})$, and this bound is tight. For fixed dimension $d$ the bound $O(n^{d/2})$ becomes polynomial in $n$, and for this setup Chazelle gave a fixed dimension polynomial time algorithm which is worst-case optimal with respect to the size of the input [20]. However, by taking the dimension as a constant, such an algorithm is allowed to “waste” an exponential amount of time on many small convex hull problems.

More recently, Khachiyan et al. showed that it is #P-hard to enumerate the vertices of an unbounded polyhedron given in terms of inequalities [46]. However, this result does not say anything about the rays of that polyhedron, whence it does not have a direct implication for the complexity status of the convex hull problem. Yet this is a strong hint that a polynomial total time convex hull algorithm does not exist.

3.2 Common algorithms

In practice several algorithms and their implementations are used; see [4,42] for previous computational studies. One purpose of this paper is to point out that essentially all of them are relevant for various convex hull problems arising in linear and integer optimization.

The beneath-and-beyond method is best described as a primal convex hull algorithm; it occurs in work of Seidel [59], see also [29,42]. So the goal is to compute the facets of the convex hull, say $P$, of finitely many given points. We may assume that the points affinely span the entire space for otherwise we can restrict our attention to the affine span. Picking an affine basis among the input points yields a simplex $P_1$ of
full dimension \(d\) which is contained in \(P\). By solving systems of linear equations we obtain the facets of \(P_1\). Now the remaining input points are added inductively, yielding a sequence of polytopes \(P_1, P_2, \ldots, P_{n-d} = P\). In the inductive step triangulations of the polytopes \(P_i\) are maintained, and these allow for the computations of the facets. The size of the final triangulation of the polytope \(P\) is the decisive factor in the running time; see [42, Proposition 3.2]. Notice that there are polytopes with few vertices and few facets for which nonetheless all triangulations are large; see [4] and [42, Sect. 3.2]. In our tests we use the implementation included in polymake, which is marked as “bb” below.

Algorithms which inductively compute the intermediate convex hulls \(P_i\) for all \(i\) are called incremental. Bremner showed that no incremental convex hull algorithm can have a polynomial total running time [16].

A dual method, which is also incremental, is double description. It was first described by Motzkin et al. [51]; see also [33] and [44, Sect. 5.2]. This is essentially the same as Fourier–Motzkin elimination and Chernikova’s algorithm [21]. Given finitely many inequalities, one starts with a subset of the inequalities which enclose a full-dimensional simplex, \(Q_1\). Up to solving a linear programming feasibility problem and up to a projective transformation, this is not a restriction. The vertices of that simplex are again computed by solving systems of linear equations. In the inductive step, to obtain \(Q_{i+1}\), the hyperplane defined by a new inequality separates the vertices of \(Q_i\) into those which are also vertices of \(Q_{i+1}\) and those which are not. The new vertices of \(Q_{i+1}\) arise as intersection of that hyperplane with the edges of \(Q_i\). The extra data structure which needs to be maintained is the vertex-edge graph of the intermediate polytopes. The double description method is implemented, e.g., in cdd [32,33], ppl [9] and porta [23]. Those software packages are tested below. See [15] for an average-case analysis of the beneath-and-beyond and double description methods.

From a certain point of view one can even consider beneath-and-beyond and double description as similar; e.g., see [33]. However, there is a choice which kind of data structures to maintain and how to maintain them during the computation. In our terminology, beneath-and-beyond primarily uses a triangulation (in the primal setting), while double description primarily uses the graph (in the dual setting). As our experiments below show this leads to a very different behavior on various kinds of input.

A third convex hull algorithm which we consider here is reverse search by Avis and Fukuda [6]. The only non-incremental algorithm in our selection is based on the simplex method for linear programming. Any linear objective function which is in general position with respect to a given polytope \(R\), induces a direction on each edge of \(R\). Starting from some vertex the optimum (which is unique as the objective function is generic) can be reached by following a directed path. The reverse search algorithm in its natural form computes dual convex hulls. It solves one linear program to determine one vertex of the feasible region from the given list of finitely many linear inequalities. By pivoting backwards all vertices can be traced. The non-trivial ingredient in this algorithm is that this can be accomplished with constant memory after the initialization. The reverse search method is implemented in lrs [3,5]. There is a multithreaded version of lrs, which we did not test; see [8].
There are more convex hull algorithms and implementations. Bruns, Ichim and Söger suggested an interesting hybridization of the beneath-and-beyond method with Fourier–Motzkin elimination, which they called pyramid decomposition [18]. As a special feature their normaliz implementation supports multithreading; we report on some tests with normaliz, and these are marked ‘nmz’. The CGAL library [61] also offers convex hull implementations, but we did not test them here; the reason is that the output is somewhat different. Joswig and Ziegler described a convex hull algorithm based on simplicial homology computation [45], which is interesting from a theoretical point of view only.

Our first two rules of thumb serve as a first approximation, and this is consistent with what had previously been observed by others.

Rule of Thumb 1 If you do not know anything about your input to a convex hull problem, try double description.

While this is useful to keep in mind, a large part of this paper is devoted to discuss natural situations in which this simple minded strategy leads astray. A conceptual disadvantage of double description, being an incremental algorithm, is that its entire output is only available at the very end of the computation. Non-incremental techniques, such as reverse search, do not share this disadvantage. This immediately leads to the next rule of thumb.

Rule of Thumb 2 Use reverse search if you expect the output to be extremely large, and if partial information is useful.

lrs features options for obtaining an unbiased estimate of the output size (and lrs running time), from looking into the first few levels of the search tree. By comparing the number of vertices (or facets) with the number of bases one can also get an idea of how degenerate the input is. Notice, however, that polymake’s current interface does not support these advanced features of lrs.

3.3 Non-symmetric cut polytopes

Let us start out with a class of examples where the first rule above works to our advantage. Consider a finite simple graph $G = (V, E)$ which is undirected, i.e., the nodes form a finite set $V$ and the set $E$ of edges consists of two-element subsets of $V$. A (possibly trivial) partition $V = A + B$ of the node set induces a cut $C(A, B) := \{e \in E | |e \cap A| = |e \cap B| = 1\}$ in $G$. We can encode such a cut via its incidence vector $\chi^C \in \{0, 1\}^{|E|}$, i.e., the 0/1-vector of length $|E|$ with $\chi^C_e = 1$ if and only if $e \in C(A, B)$. The cut polytope of $G$ is the polytope

$$\text{Cut}(G) := \text{conv}(\chi^C | C \text{ is a cut in } G).$$

A classical line of research in combinatorial optimization asks to determine the facets of cut polytopes; see Schrijver [58, Sect. 75.7].

Remark 1 Each automorphism of $G$ gives rise to a linear automorphism of its cut polytope $\text{Cut}(G)$.
For our first experiment we look at graphs with few nodes and which are completely asymmetric, i.e., their automorphism group is trivial. In Sect. 3.6 below we will discuss cut polytopes of graphs which are symmetric. Let $G_k$ be the connected graph with $k + 6$ nodes and $k + 6$ edges shown in Fig. 2. For $k \geq 1$ the graph $G_k$ is completely asymmetric, while $G_0$ admits one involutory automorphism.

The relevant parameters for the cut polytopes of the graph $G_k$ are easy to determine. The dimension, $d = \dim(\text{Cut}(G_k))$, equals the number of edges, which is $k + 6$. There are $n = 2^{k+5}$ many cuts (including the empty cut), all of which form vertices, as cut polytopes are 0/1-polytopes. The values $d$ and $n$ describe the size of the input. In this case, since the graph $G_k$ does not contain a $K_5$-minor, the facets of $\text{Cut}(G_k)$ can be determined via a result of Barahona [11]; see also [58, Corollary 75.4f]. This yields $m = 2d + 8 = 2k + 20$ facets.

Our experiment investigates the primal convex hull computations for the polytopes $\text{Cut}(G_k)$, where $k$ ranges from 0 to 14, and for which we tried the six implementations bb, cdd, lrs, nmz, porta, and ppl. The polytopes were given to polymake in reduced vertex description via VERTICES and LINEALITY_SPACE. See the accompanying web page [54] for full details.

polymake has default options for computing convex hulls etc. but one can directly influence the choice of algorithm for various tasks within the polymake shell by issuing prefer statements. To choose, e.g., ppl as the default method for convex hull computations one can use

```
polytope> prefer "ppl";
```

This choice will be stored in the preferences file of polymake (.polymake/prefer.pl in the home folder) and is valid until one sets another default. To switch only temporarily one can use prefer_now, which is only valid on the line it is contained in, e.g.,

```
polytope> prefer_now "lrs"; print $p->FACETS;
```

This can be fine tuned with minor keys in the form ppl.convex_hull, see the polymake wiki for more details. By the way, omitting the “print” in the second command above will trigger the convex hull computation without producing output. This is instrumental, e.g., for proper timings.

The programs cdd, lrs, nmz and ppl are interfaced via their library versions, hence there is virtually no overhead from testing through the polymake system. However, our interface to porta is based on an exchange of text files. This does create some overhead, which is visible in porta’s curve in Fig. 3, for small input. For large input, however, this effect can be neglected (at least on a logarithmic scale).

Every code was tried ten times on each input with the same parameters. The average running times are displayed in Table 3 in the Appendix and Fig. 3. Note that bb runs
Fig. 3 Timings (in seconds, logarithmically scaled) for convex hull computations of non-symmetric cut polytopes $\text{Cut}(G_k)$; see Table 3 in the Appendix. The curves for nmz $6$ and ppl almost agree. (bb: beneath-beyond, nmz $6$: libnormaliz with 6 cores)

out of memory quite early. More precisely, our imposed memory limitation to 4GB does not suffice for $\text{Cut}(G_5)$ and beyond; see the Appendix for details on our experimental setup. The reason is that the triangulations grow very quickly. For instance, the triangulation of $\text{Cut}(G_4)$ which was computed by bb already has 1,040,517 facets. For $\text{Cut}(G_5)$ we even get 11,199,900 facets within like 200s but requiring about 10GB of main memory. The asymptotic behavior of cdd is quite good, but it is clearly outperformed by ppl and nmz. The only parallel code is normaliz, which here we tested with six cores. Our experiments (undocumented here) suggest a factor of three in performance versus the single-threaded version of nmz.

**Rule of Thumb 3** Do use double description for computing the facets of 0/1-polytopes.

The curves for the six-threaded nmz and the single-threaded ppl match quite well. For practical purposes this seems to suggest that nmz parallelization, which employs the OpenMP protocol, is particularly interesting for more than six cores.

### 3.4 Knapsack integer hulls

For our second test we look at the kind of knapsack polytopes introduced in Sect. 2. Consider the Fibonacci sequence

$$a_1 = 2 \quad a_2 = 3 \quad a_i = a_{i-2} + a_{i-1}.$$ 

Then the **Fibonacci fractional knapsack polytope** in dimension $d$ for the parameter $b > 0$ is defined as

$$F_d(b) = \{ x \in \mathbb{R}_{\geq 0}^d \mid a^T x \leq b \},$$

where $a = (a_1, \ldots, a_d)$ is the vector of the first $d$ elements of the Fibonacci sequence. As pointed out before, as mere polytopes these are boring as all of them are $d$-dimensional simplices. What makes a more interesting computational experiment is to
Table 1 Properties of the Fibonacci fractional knapsack polytopes $F_d(b)$

| $d\backslash b$ | 40  | 50  | 60  | 70  | 80  | 90  | 100 |
|----------------|-----|-----|-----|-----|-----|-----|-----|
| Number of integer points | 1021 | 2145 | 4008 | 6879 | 11,069 | 16,929 | 24,853 |
| Number of facets of the integer hull | 46  | 7   | 7   | 8   | 6   | 6   | 8   |
| Number of vertices of the integer hull | 48  | 1   | 1   | 9   | 1   | 288 | 1   |

look at the set of integer points in $F_d(b)$ and to compute their convex hull, which yields the corresponding Fibonacci knapsack polytope. So, as in Sect. 3.3, we are computing the facets of sets of integer points. As a key difference, however, here the vast majority of the input points is redundant, i.e., they do not form vertices of the integer hull. See Sect. 4.3 below for a discussion about how to compute these integer points. Each polytope for this set of experiments was defined in redundant vertex description via the property POINTS, see [54] for details.

To get an idea about the input and output sizes Table 1 lists the number of lattice points (which provide the input), the number of facets (which form the output) and the number of vertices depending on $d$ and $b$. By a result of Hayes and Larman [39] the number of vertices of the integer hull of any knapsack $d$-polytope is bounded by $\sigma^d$, where $\sigma$ is the length of the binary encoding of the knapsack inequality; our examples are rather far from this bound.

Looking at the timings in Fig. 4a and Table 4 in the Appendix this experiment seems to be showing something similar to the previous one: ppl outperforms the other codes, and lrs is far behind. As a major difference to the previous experiment here porta can nearly compete with ppl. For proper comparison, notice that in Fig. 3 the dimension of the input increases with $k$, which is the parameter on the $x$-axis, while Fig. 4 reports on results in fixed dimension 5. The most interesting aspect of the Fibonacci knapsack examples, however, is that they support the following.

Rule of Thumb 4 For iterative convex hull algorithms the insertion order often makes a difference. This is particularly true if the input contains redundant points. If it is easy to recognize redundant input, it usually pays to remove that first. This is not restricted to convex hull codes.

An extreme approach to remove redundant input would be to solve one linear program per input point to decide whether or not this is a vertex. In most circumstances, this is more expensive than computing the convex hull with redundancies.
The key to understanding the computational results is the observation that, for \( d = 6 \) and \( b = 100 \), less than 0.5\% of the input points are vertices. The best possible scenario for any incremental convex hull algorithm is to process a sequence of points with the vertices coming first. In more complicated cases the ordering among the vertices may also matter.

To illustrate the situation we ran extensive tests on the Fibonacci knapsack examples with various insertion orders; see the rather lengthy Table 4 in the Appendix. The cdd and ppl implementation are almost insensitive to this variation since they employ heuristics to process the input in their preferred order. Also normaliz orders the input in an initial step, so we did not run tests with varying order for it. All other codes are very sensitive to the insertion order, even lrs, which is not an iterative algorithm. However, the ordering of the input affects the traversal of the search tree. Figure 4b is an overly condensed version of Table 4 in the Appendix; for each implementation we took the timing of the best ordering that we found. porta and bb quite clearly want the vertices first, while lrs prefers a randomized order. In practice, for a specific input, it will often be unrealistic to tell in advance which ordering will work best for which method. The purpose of this test is to underline that, in addition to choosing the proper algorithm, it may be worth-while to spend a thought on the input order, too.

### 3.5 Voronoi diagrams

A completely different convex hull set up comes about as follows. Consider a configuration \( S \) of \( m \) points in \( \mathbb{R}^{d-1} \). The **Voronoi region** of a site \( s \in S \) is the set of points in \( \mathbb{R}^{d-1} \) whose distance to \( s \) does not exceed the distance to any other point in \( S \). Conceptually, the distance can be measured in terms of any fixed metric, but here we are looking at the Euclidean case only. Then the Voronoi region of \( s \) is a convex polyhedron which may be bounded or not. An immediate inequality description of a Voronoi region arises from bisecting hyperplanes between pairs of sites. Ranging over all sites in \( S \) the Voronoi regions form a polyhedral complex, i.e., a set of polyhedra meeting face-to-face, and this polyhedral complex is the **Voronoi diagram** of \( S \). By
construction the Voronoi diagram covers the entire space $\mathbb{R}^{d-1}$. Voronoi diagrams are an indispensable tool in computational geometry and its applications.

The (Euclidean) Voronoi diagram of $S$ can be computed by solving a dual convex hull problem. This is due to the fact that the Voronoi diagram of $S$ is the orthogonal projection of the unbounded polyhedron in $\mathbb{R}^d$ whose $m$ facets are those tangent hyperplanes to the standard paraboloid $\{(x, \|x\|^2) \mid x \in \mathbb{R}^{d-1}\}$ in $\mathbb{R}^d$ which lie above the sites in $S$. This way the 0-dimensional cells of the Voronoi diagram of $S$ are the projections of the vertices of the unbounded polyhedron

$$V(S) := \left\{ (x, \delta) \in \mathbb{R}^{d-1} \times \mathbb{R} \mid 2\langle s, x \rangle - \|s\|^2 \leq \delta \text{ for all } s \in S \right\},$$

which is full-dimensional. From the vertices and the facets (which are given by the sites) of $V(S)$ one can, for instance, compute a facet description of each Voronoi region by a combinatorial procedure. The same is true for other additional information that might be desired. In this sense, solving the dual convex hull problem for the polyhedron $V(S)$ is the key step; e.g., see [44, Sect. 6.3].

For our experiment we choose $m$ points uniformly at random within the cube $[-1, 1]^{d-1}$. The polytopes $V(S)$ were given to polymake in reduced facet description via FACETS, derived from the sites given by the random points. Except for $\text{bb}$ where we added an explicit polarization step because this implementation originally worked for primal convex hull only, thus the input was VERTICES. See the accompanying web page [54] for full details. For each choice of parameters we do this experiment ten times. The timings in Table 5 in the Appendix give the average values. We tried $m \in \{500, 1000, 1500, \ldots, 3000\}$ and $3 \leq d \leq 7$, but we restricted our attention to those cases which could be computed within one hour of single-threaded CPU time. While Table 5 in the Appendix also lists a few timings which are longer, everything is truncated to one hour in Fig. 5. Notice that the polyhedron $V(S)$ is simple (i.e., each vertex is contained in $d$ facets) almost surely.

On this non-degenerate input cdd does not work so well and can only solve a few small problems. Reverse search is a method which is designed for non-degenerate input, and so its implementation lrs performs quite well. In fact, the worst case running time of reverse search is bounded by $O(dmn)$ on non-degenerate input; and this fits with our experiment. It is plausible (but not proved, as far as we know) that $n$ grows only slightly larger than linearly in $m$ in this case, and so the running time in fixed dimension approximately grows quadratically. Yet this is outperformed by the beneath-and-beyond method $\text{bb}$, implemented in polymake. Its worst case running time was estimated at $O(d^5mt^2)$ in [42], where $t$ is the size of a triangulation of the polar of $V(S)$ which $\text{bb}$ implicitly computes on the way. While it is plausible (but again not proved, as far as we know) that $t$ is small in this case, possibly even about linear in $m$, the above bound seems to be too pessimistic. The empirical running times on this input grow almost linearly in fixed dimension. Notice also that, for any $d$ and very small $m$ lrs, normaliz and ppl beat $\text{bb}$.

**Rule of Thumb 5** On random input the beneath-and-beyond algorithm often behaves very well.

Springer
Computing convex hulls and counting integer points with polymake

We conclude the report on this experiment with an explanation why it suffices to list the average over only ten runs per parameter set via a simple statistical analysis. To this end we fix one set of parameters for which we do the random sampling one thousand times. We pick $d = 5$ and $m = 500$ because all algorithms terminate within a reasonable time-span.

The first random variable to look at is the number $n$ of vertices of the Voronoi diagram, whose distribution is plotted in Fig. 6. The numbers range from 12,538 to 13,130 with average value 12,816.352, median 12,816 and standard deviation 93.037. For comparison the diagram also shows a normal distribution with the given expected value and standard deviation. To get an idea, it follows from the Generalized Lower Bound Theorem of Murai and Nevo [52] that $n \geq 1982$ (provided that the input is in
general position). On the other hand McMullen’s Upper Bound Theorem [50] yields $n \leq 246,512$.

The other natural random variables are the running times for the various convex hull codes. Like for the output sizes, these are strongly concentrated around their respective mean. We omit the details.

Remark 2 We do not report porta timings here since porta dies with an error message on each input of this kind. This seems to be related to a flaw in porta’s implementation of the arithmetic. The most recent version [23] of porta was published in 2009.

3.6 Symmetric cut polytopes

We continue to examine primal convex hull computations arising from cut polytopes. The difference to Sect. 3.3 is that here we look into graphs with non-trivial symmetry. A symmetric convex hull problem can be solved in various ways by graph based search techniques up to symmetry which need to be combined with some standard convex hull algorithm applied to subproblems of smaller size; see Bremner et al. [17]. This has been implemented in sympol by Rehn [57], and this is also available in polymake, through which sympol can currently be combined with bb, cdd, lrs and ppl.

Our first example is the path $P_k$ on $k$ nodes, which has $k-1$ edges. The cut polytope $\text{Cut}(P_k)$ is the 0/1-cube of dimension $k-1$. It has $n = 2^{k-1}$ vertices and $m = 2k - 2$ facets. The graph $P_k$ is symmetric with respect to exchanging both ends, which yields an automorphism group of order two. Clearly, the 0/1-cube has a much larger group of automorphisms (of order $2^{k-1} \cdot (k - 1)$), but most of its elements are not induced by graph automorphisms.

The second example is the cycle $C_k$ of length $k$. The cut polytope $\text{Cut}(C_k)$ is the convex hull of all 0/1-vectors of length $k$ with an even number of 1’s. It has $n = 2^{k-1}$ vertices and $m = 2k + 2^{k-1}$ facets. The automorphism group of the graph $C_k$ is the dihedral group of order $2k$.
The third example is the complete graph $K_k$ on $k$ nodes, which has $\binom{k}{2} = k(k - 1)/2$ edges. The number of vertices of $\text{Cut}(K_k)$ equals $2^{k-1}$. The facet descriptions are known for $k \leq 8$; see [7,14,22]. For $k = 6$ the cut polytope has 368 facets, for $k = 7$ it has 116,764 facets, for $k = 8$ it has 217,093,472 facets, and at least $12,246,651,158,320$ facets for $k = 9$ [14]. The latter number is conjectured to be the true number of facets. The automorphism group of $K_k$ is the full symmetric group of degree $k$.

As for the case of non-symmetric cut polytopes the input was given as a reduced list of vertices. However, in this case we also passed a list of generators of the symmetry group to polymake for the computations. See [54] for the complete setup.

Table 2 gives the average time in seconds used by the algorithms in bb, cdd, lrs, ppl and their combinations with sympol for the convex hull computations of the three classes of cut polytopes. Independent of the algorithm, at least for large input the advantage of taking symmetry into account is evident. The larger the group of automorphisms the greater is the gain.

**Rule of Thumb 6** If the input is large but symmetric do compute convex hulls up to symmetry.

The differences in performance of the convex hulls codes on this input are consistent with the non-symmetric cut polytopes from Sect. 3.3. We take this as an indication that the rule of thumb above is independent of the choice of the basic algorithm.

## 4 Integer points

In this section we compare various methods to count the integer points in a polytope. This is a fundamental task which appears in various different areas of mathematics, among them number theory, statistics, algebraic geometry, and representation theory (see e.g. [25] for an overview). In the context of optimization this most frequently occurs as a first step if one is interested in computing the facets of integer hulls.

### 4.1 Complexity status

While this was already mentioned in the introduction, we wish to repeat: Even to decide if a polyhedron contains some integer point is NP-complete [34]. On top of this there is no polynomial bound on the number of lattice points with respect to the input size. Hence all algorithms that enumerate the lattice points easily show an (at least) exponential worst case behavior with respect to time and memory. Also note that in dimensions at least three, a lattice polytope without any lattice points, except for the vertices, can have arbitrarily large volume. In contrast, for a fixed positive number of interior lattice points the volume is bounded.

Despite these obstacles the number of lattice points can be determined in polynomial time in fixed dimension with Barvinok’s algorithm [12]. For a short explanation of the algorithm see the next section. Variations of this algorithm have been implemented in two different software packages. This leads to the first rule of thumb for this section.
Table 2 Computation times (in seconds) for the convex hull of several cut polytopes where the symbol □ stands for out of memory and * means that we have only done one iteration of the test.

| G   | d  | n   | m   | bb   | cdd  | lrs  | ppl  | bb\_sym | cdd\_sym | lrs\_sym | ppl\_sym |
|-----|----|-----|-----|------|------|------|------|---------|---------|---------|----------|
| P_9 | 8  | 256 | 16  | 0.38 | 0.18 | 5.36 | 0.07 | 0.29    | 0.23    | 1.36    | 0.08     |
| P_{10} | 9  | 512 | 18  | 3.09 | 0.52 | 90.80| 0.07 | 2.04    | 0.95    | 26.12   | 0.25     |
| C_9 | 9  | 256 | 274 | 2.05 | 0.88 | 29.46| 0.09 | 1.07    | 0.97    | 2.86    | 0.56     |
| C_{10} | 10 | 512 | 532 | 24.99| 4.70 | 602.04| 0.20 | 6.76    | 4.22    | 62.70   | 2.40     |
| K_6 | 15 | 32  | 368 | 6.80 | 0.25 | 1.74 | 0.09 | 1.04    | 0.20    | 0.20    | 0.10     |
| K_7 | 21 | 64  | 116,764 | □ | 14,329.75* | 31,309.01* | 2212.73* | □ | 129.98 | 1217.96 | 32.00   |
Rule of Thumb 7  If you do not know anything about your input and you are only interested in the number of lattice points, try LattE [26] or barvinok [63].

The reason for this general recommendation is that in practice any method which explicitly enumerates the lattice points will often feel the limit of the amount of memory available.

4.2 Common algorithms

Algorithms to solve the enumeration or counting problem in practice basically fall into four different categories.

The basic form of Barvinok’s algorithm computes the rational multivariate generating function for the lattice points in an affine polyhedral cone $C$ using a signed decomposition of $C$ into simplicial unimodular cones whose generating functions are easy to determine. The key feature of this algorithm is that we can bound the number of unimodular cones and their faces in this decomposition by a polynomial in the input size if the dimension is fixed. In its original form the algorithm has to care for lattice points in intersections of cones in the decomposition using inclusion-exclusion. There are two commonly used variations that avoid this, either working with the dual of the cone or using irrational decomposition [49]. The first approach is based on the observation that duals of low-dimensional cones are non-pointed and the rational generating function of a non-pointed cone vanishes. In the second approach one translates the cone by a small vector in such a way that no lattice point is contained in a lower-dimensional face (this can be done without explicitly computing the shift). See [12] for details.

By Brion’s Theorem the generating function of the integer points in a rational polytope $P$ is given by the the sum of the generating functions of all vertex cones. Via Barvinok’s algorithm we obtain an algorithm to compute a polynomial size representation of the generating function of $P$ in polynomial time, if the dimension is fixed. Evaluation at 1 returns the number of lattice points. This algorithm has been implemented in the software packages LattE [26,27] and barvinok [63]. The latter package also contains several modifications of the original algorithm (see, e.g., [48]). Here we only test LattE, since currently there is no interface between polymake and barvinok.

Historically, LattE uses decomposition in dual space as the default method for triangulating the vertex cones. However, the current version LattE integrale also implements irrational decomposition and a mix of dual and irrational decomposition. It also allows to stop the signed decomposition if the determinant of a cone falls below a threshold. This is motivated by the observation that determinants initially drop fast in the decomposition process, but breaking down cones with small determinant into unimodular ones often takes a considerable amount of time. LattE also allows to compute the generating function of the homogenization cone $\sigma_P := \text{cone}(P \times \{1\})$ via Barvinok’s algorithm.

By default, polymake calls the default version of LattE using decomposition in dual space. This can be changed by setting command line options for the call to LattE. In polymake, this can be done with the command set_custom, e.g.
For more options see the manual of \textit{LattE integrale}. The parameters chosen may have a tremendous influence on the running time. As for the convex hulls there is no a priori way to determine which algorithm works best on a given instance. The manual of \textit{LattE integrale} gives some hints. We demonstrate this in our experiments by running them once with the default options and once with the above choice (denoted by \textit{LattE}$^\vee$).

The second type of algorithm directly enumerates the lattice points of a polytope. Two such methods are implemented in \textit{polymake}.

1. The algorithm \texttt{bbox} encloses a polytope into a scaled cube with edges parallel to the coordinate axes. It then checks for every point in the cube whether it is contained in the polytope.

2. The algorithm \texttt{projection} recursively enumerates lattice points in the fibers over lattice points in the projection of the polytope into one of the coordinate hyperplanes. This uses an optimized Fourier–Motzkin elimination to create irredundant descriptions for the polytopes which are the images of the projection.

The third method that can be used to enumerate lattice points in fact solves the more general problem of finding a Hilbert basis in the cone $\sigma_P$. As all lattice points in $P \times \{1\}$ must be in the Hilbert basis we can filter out all generators with a higher last coordinate to obtain a list of lattice points. Various algorithms exist for enumeration of Hilbert bases, but there are only two large software projects that have implemented some of the methods. To both software projects \textit{polymake} provides an interface.

1. The program \texttt{4ti2} \cite{4ti2} uses a project-and-lift approach to compute Hilbert bases of polyhedral cones intersected with arbitrary lattices \cite{40}. Computation of integer points in polyhedra is treated as a special case by bounding the height of the generators.

2. The program \texttt{normaliz} \cite{normaliz} computes (among other things) Hilbert bases of cones and enumerates lattice points in polytopes. Roughly speaking, the algorithm constructs a triangulation of a cone (\textit{e.g.}, via a placing triangulation similar to the beneath-and-beyond method explained above), then computes a Hilbert basis of each simplicial cone and finally reduces the union of all Hilbert bases of the simplicial cones to a Hilbert basis of the original cone. Computation of a Hilbert basis of a simplicial cone is done by enumerating all lattice points in the fundamental parallelepiped using a transformation to the positive orthant. For the enumeration of the lattice points of a polytope \texttt{normaliz} considers the homogenization cone and discards all generators of height 2 or above in the computation of the Hilbert basis.

There exists an option to run \texttt{normaliz} in a \textit{dual mode} which is based on an algorithm by Pottier \cite{56}. As for the convex hulls there is no a priori way to determine which algorithm works best on a given instance. Still, the authors recommend this mode when only an outer description is given as input. We demonstrate this in our experiments by running them once with the default options and once in dual mode (denoted by \texttt{normaliz}$^\vee$).
Generally, as Barvinok’s algorithm is polynomial in fixed dimension it should be used if only the number of lattice points needs to be determined. We will see an example supporting this recommendation in the following section, where we pick up the fractional knapsack polytopes from Sect. 3.4 again and discuss the computation of the lattice points used as input above. However, in the subsequent section we will introduce random box polytopes and show that LattE fails on those, seemingly simply structured, polytopes. At the same time the theoretically inferior methods work quite well. We will shortly discuss further examples in Sect. 4.6.

Clearly, one can tailor implementations to address special cases. For purposes related to integer linear programming it is most interesting to count or to enumerate the 0/1-points in a polytope which is given in terms of inequalities. The prototypical implementation is azove [13], which is also interfaced in polymake (via text file exchange). azove traverses all paths in a binary decision diagram to enumerate the 0/1-points.

Remark 3 The latest version of azove was published in 2007. To use it with a modern Linux kernel requires some patching, as otherwise the initial memory allocation fails.

4.3 Knapsack polytopes

We evaluate the different algorithms for counting and enumerating the lattice points in Fibonacci fractional knapsack polytopes as introduced in Eq. (1) of Sect. 2. This enumeration is the first step of the integer hull computation in Sect. 3.4. According to our Rule of Thumb 7 we would expect LattE to perform significantly better than the other algorithms, but as we will see, the picture is slightly more complex.

Remark 4 By adding a slack variable a fractional knapsack polytope can also be written as the set of non-negative solutions to a single linear equation. In this context Baldoni et al. [10] described a polynomial time algorithm to compute the top coefficients of a quasi-polynomial which counts the integer solutions.

We performed two series of experiments. In the first we fixed the dimension to \( d = 5 \) and varied the right hand side of the knapsack equation. Geometrically speaking, this corresponds to shifting one of the facets of a simplex. In the second series we fixed the right hand side \( b = 60 \) and varied the dimension. In both cases, we provided the full lists of vertices and facets of the polytope as input to the algorithms, see [54]. All timings are listed in Tables 6 and 7 in the Appendix and plotted in Fig. 7. Note that in both cases the projection algorithm performs pretty well.

We also determined what our memory restriction of 4 GB implies as an upper limit on the number of points depending on the dimension. This is corresponds roughly to 17 million points in \( F_5(350) \) and 5 million points in \( F_8(200) \). In both cases the projection algorithm finishes in about a minute.

4.3.1 Varying the right hand side

First, note that when increasing the right hand side the number of lattice points \( \ell \) in the polytope will increase significantly. The values of \( \ell \) for our examples are also given
in Table 6 in the Appendix. Hence, the amount of work naturally increases for all algorithms but \textsc{LattE}. More precisely, the running time for the Barvinok algorithm as implemented in \textsc{LattE} seems constant. This is plausible, since changing the right hand side only gives a dilation of the same polytope, the amount of work to compute the generating functions at each vertex stays exactly the same.

This experiment, displayed in Fig. 7a, shows the strength of Barvinok’s algorithm, and this is why we suggest the Rule of Thumb 7.

4.3.2 Varying the dimension

In this series of tests, we fix the right hand side, \textit{i.e.}, the size of the knapsack, and the coefficients, \textit{i.e.}, the object sizes, are the Fibonacci numbers. At first sight the behavior of the implementations tested may seem erratic; see Fig. 7b. In fact, it correlates with various strategies to avoid the “curse of dimension”. If the dimension gets high enough the size of the additional objects will be bigger than the knapsack. For our test this means that the number of lattice points, \( \ell \), in the Fibonacci fractional knapsack polytopes becomes constant for \( d \geq 8 \); see Table 7 in the Appendix. Moreover, all lattice points lie in the subspace corresponding to the first seven coordinates.

The lifting approach used in \textsc{4ti2} detects this and the running times are constant starting from \( d = 8 \). The \textsc{bbox} algorithm determines zero – rounded down from some rational value less than one – as the upper bound for each further coordinate and thus the only additional work comes from the increasing number of zero-entries in the lattice vectors. The slight growth in the running time of the \textsc{projection} algorithm mainly comes from the number of lifting steps that are needed to generate the lattice points.

Here, one can see that the dimension has a strong impact on the running time of \textsc{LattE}, because of the increasing number of vertices and the more complicated generating functions, both with all primal and default parameters. For high dimension, \textsc{4ti2}, \textsc{bbox}, \textsc{normaliz}, and \textsc{projection} perform a lot better. Since version 2.99.4, \textsc{normaliz} employs a special strategy to reduce the size of the funda-
mental parallelepipeds, which is triggered above a certain threshold. This is the reason for the bumps at very low input parameters. The jump at dimension 14 is caused by an automatic coordinate upgrade that switches from 64 bit long arithmetic to arbitrary precision. Further, the dual mode of normaliz performs extremely well for this type of input.

**Rule of Thumb 8** If most of the lattice points lie in a low-dimensional subspace then normaliz (both in default and dual mode) or projection can still find them all.

### 4.4 Random box

We also considered a second class of polytopes that show a completely different behavior with respect to the different algorithms. For parameters $d$ and $n$ we define a random box polytope $R(d, n)$ as the convex hull of $n$ lattice points chosen uniformly at random from the cube $C^d_5 := [0, 5]^d$.

For each of the algorithms bbox, LattE, normaliz, and projection we did the following experiment: for each dimension $d$ between 4 and 9, and for each $n$ in the set $\{20, 30, 40, 50, 60, 70\}$ we generated 10 random polytopes $R(d, n)$ and tried to count the lattice points in the convex hull with each of the algorithms. Again, we provided the full lists of vertices and facets of the polytope as input to the algorithms, see [54]. The average running times are plotted in Fig. 8 and are listed in Table 8 in the Appendix. We did not include 4ti2 and normaliz in dual mode in these experiments; for details see Remark 5.

In this experiment only normaliz and projection finished all tests within the given time and memory restrictions.

bbox failed on some of the 9-dimensional examples, while LattE basically only worked for dimension 4 and 5. With the all primal mode, LattE was able to compute some of the 7-dimensional examples.

Observe that the number of lattice points inside $C^d_5$ grows exponentially in $d$ (in fact, there are $6^d$ lattice points). Hence, the fraction of the lattice points chosen decreases with the dimension. Yet, we expect that coordinate projections of the polytope $R(d, n)$ use the full range $[0, 5]$ for its coordinates, which forces bbox to enumerate many lattice points that do not contribute to the final result. This clarifies why this algorithm runs into problems on this class of polytopes for higher dimensions.

We see from the experiments that LattE basically only works for the low dimensional examples. This particularly bad behavior can be explained by the specific version of Barvinok’s algorithm employed.

We have already outlined the basic idea of this algorithm above. In its original form it starts with some initial triangulation of the vertex cones of the polytope. The index of such a simplicial cone in the triangulation is the determinant of its generators. The algorithm now successively lowers the maximum index of a cone in the triangulation by replacing cones with a signed decomposition into cones of lower index. The number of steps used by the algorithm heavily depends on the initial maximal index of a cone. Once we have a signed decomposition we can add up the rational generating functions for each cone. A drawback of this approach is that we count lattice points in intersections of cones multiple times, once for each cone they
Fig. 8 Running times (in s) for counting lattice points in random polytopes depending on the dimension $d \in [4, 9]$ and the number of points $m \in [20, 70]$, a bbox, b projection, c LattE, d LattE∨, e normaliz. See Table 8 in the Appendix for exact timings.

are in. LattE implements both options discussed above to avoid this problem. We can choose to work with the dual cone, or we can use *irrational decomposition*.

By default, LattE chooses the first of the two variants and does a triangulation in dual space. For our experiments we can now observe that the index of an initial triangulation of the dual of a random polytope $R(d, n)$ is quite large. This explains that LattE won’t finish on these examples within our given time limit.

We see, however, that the index of an initial triangulation in primal space is much lower. So the performance of LattE significantly improves on these examples if we
ask **LattE** to use *irrational decomposition* and do a signed triangulation in primal space; compare Fig. 8c, d.

Since we are dealing with random polytopes we did a similar statistical analysis as at the end of Sect. 3.5, to explain why it suffices to list the average over only ten runs per parameter set. The natural random variables are the running times for the various algorithms. For **bbox**, **LattE**, **normaliz**, and **projection**, they are strongly concentrated around their respective mean. We omit the details.

**Remark 5** We omit the timings for **4ti2** and **normaliz** in dual mode because for this experiment the statistical analysis found that they behave erratically, **normaliz** to a lesser extent than **4ti2**; see Fig. 9. We do not have an explanation for this phenomenon.

### 4.5 Matching polytopes

Consider a finite simple undirected graph $G = (V, E)$. For a vertex $v \in V$ we let $\delta(v) \subseteq E$ be the set of edges incident with $v$. A vector $x \in \mathbb{R}^E$ which satisfies

$$
\sum_{e \in \delta(v)} x_e \leq 1 \quad \text{for all } v \in V \quad \text{and} \quad x_e \geq 0 \quad \text{for all } e \in E
$$

is called a *fractional matching*. This gives rise to the *fractional matching polytope*

$$
M(G) := \text{conv}(x \in \mathbb{R}^E \mid x \text{ is a fractional matching of } G)
$$

of the graph $G$. The dimension $d$ of $M(G)$ is easy to determine, it is the same as the number of edges $d = |E|$. Further the number $m$ of constraints on $M(G)$ is equal to $|V| + |E|$. A *matching* of $G$ is a subset of the edge set $E$ which covers each vertex at most once. By construction the matchings are in bijection with the integer points in $M(G)$, all of which have 0/1-coordinates. Finding matchings, e.g., of maximum cardinality, has a variety of applications. Thus, it is generally desirable to solve integer linear
programs over $M(G)$. This is our motivation to look into counting and enumerating matchings, i.e., the lattice points in fractional matching polytopes.

Our test graph is the complete graph $K_n$ with $n$ nodes. In this case $d = \binom{n}{2}$ is the dimension of $M(K_n)$, and the number of inequalities equals $m = \binom{n}{2} + n$. In addition to the five implementations used before, we also tested azove. This is a special implementation for enumerating 0/1-vectors. Each code was tried ten times on each input with the same parameter. In each experiment, the polytopes were given by their list of facets and, for the projection algorithm, also the vertices, see [54].

The average running times are displayed in Fig. 10 and Table 9 in the Appendix.

Note that projection could go slightly further but this algorithm does not only require the inequalities of the polytope but also the vertices. As $M(K_9)$ already has as many as 79,892 vertices it was not possible in our setup to compute the vertices of $M(K_{10})$: we expect around 700,000 vertices which exceed our memory constraint of 4 GB.

Since we start with an outer description, the dual algorithm of normaliz again performs significantly better.

It is no surprise that azove outperforms all other methods, as it was specifically designed to solve this kind of problem. No other code can obtain the result in less than one hour for $n \geq 10$. For comparison, it takes 2916.652 seconds for azove to finish the task for $M(K_{19})$.

**Rule of Thumb 9** Use azove to enumerate the 0/1-points in a polytope.

Notice, however, that azove might need to be patched; see Remark 3.

### 4.6 Additional comments

The above examples show that the performance of the various algorithms does depend on properties of the polytopes that are difficult to check a priori. The following example also shows that a high initial index need not necessarily lead to problems for Latte.
Let
\[ P := \text{conv}(0, e_1, e_2, e_1 + e_2 + a \cdot e_3, e_1 + e_2 + b \cdot e_4, e_1 + e_2 + c \cdot e_5) \]

for large pairwise co-prime \( a, b, c \). This polytope is a simplex whose only lattice points are the vertices. Its index is \( abc \). Still LattE finishes fast, projection at least finishes, while bbox, 4ti2, and normaliz either do not complete the computation within reasonable time or run into numerical problems.

This example also demonstrates that the problem of counting or enumerating lattice points in polytopes has a distinctive number theoretic flavor. Small changes in the input which, from a linear programming point of view, have only little impact on the problem may behave completely different with respect to the counting problem, as small changes may greatly influence the number of integer solutions of an equation. The algorithms used in 4ti2, LattE, and normaliz all implicitly depend on such number theoretic influences, which explains some of the results we observed. In contrast, the projection method implemented in polymake uses a plain convex geometric approach. This suggests why the latter approach is usually not the best on any given input but often behaves quite well.

Rule of Thumb 10 If you want to enumerate lattice points in a general polytope and do not have any further information then try projection.

5 Summary

For our discussion on polyhedral and optimization algorithms and their implementations we have focused on two key problems. The representation conversion problem is a central task in classical polyhedral geometry, while the problem of counting or enumerating lattice points is a main task in integer or combinatorial optimization, number theory and algebra. Commonly used algorithms and implementations for these two problems are conveniently available through polymake. For each of the problems polymake itself adds additional algorithms, the beneath-and-beyond method for the conversion problem and both the projection as well as the bounding box method for the enumeration of lattice points.

We have explained with various examples that in particular our implementation for the projection method behaves surprisingly well on a wide range of problems, while many of the other algorithms show a much stronger dependence on the type of the input. For the convex hull algorithms the picture is less clear as each algorithm and each implementation has its particular class of problems where it is most successful.

In our Rule of Thumb 1 we advertised the double description method. Many of our tests seem to suggest that its implementation in ppl is superior to the one in cdd. It should be noted, however, that there are instances known where that superiority is reversed. For instance, Avis and Jordan [8] report on cases where ppl is a hundred times slower than cdd.

For lack of maintenance the software packages porta and azove are essentially outdated, but they can still sometimes be useful. However, any naive usage may result in crashes.
Our examples were chosen with the desire to look into a small number of plausible scenarios. The main message is that very often it is difficult to predict which convex hull algorithm performs best on a specific input. Nonetheless, we hope that our ten “rules of thumb” can serve as a guideline.

Acknowledgements We thank Thomas Opfer for contributing his implementation of the dual simplex method, originally written for his Master’s Thesis [53], to the polymake project, and this includes the code maintenance until today. Moreover, we are very grateful to the developers of cdd, lrs, normaliz and ppl as they gave us various kinds of valuable feedback. The comments by David Avis and Winfried Bruns were particularly detailed. What we found most rewarding is the fact that, partially in reaction to the 2014 preprint version of this paper, the teams of lrs and normaliz published new releases of their codes. Throughout these showed improvements which are sometimes very substantial, e.g., for normaliz’ handling of non-symmetric cut polytopes. The interested reader may find it worthwhile to compare the results below with that preprint version (which is still available as arXiv:1408.4653v1).

Appendix 1: Experimental setup

Everything was calculated on identical Linux machines with the memory limit set to 4 GB (via ulimit). All tests in one section were done on the same machine. Any test exceeding this bound is marked as Δ in the respective tables. All timings were measured in CPU seconds, except for tests on non-symmetric cut polytopes where we used wallclock time to show the performance of the multithreaded version of normaliz. Those entries marked with an * ran only one iteration. The hardware for all tests was:

- CPU: AMD Phenom(tm) II X6 1090T
- bogomips: 6421.34
- MemTotal: 8191520 kB

All tests were done on openSUSE 13.1 (x86_64), with Linux kernel 3.11.10-25, gcc 4.9.3 and perl 5.18.1.

All tests were run through polymake version 2.15-beta3 via the respective interfaces. This creates some overhead, for instance, due to data conversion. While bb is the only implementation which is actually part of polymake, this does not constitute a principal technical advantage over the other convex hull codes tested. The libraries cddlib 0.94h, lrslib 6.0 and libnormaliz 2.99.4 (which contains the same code as version 3.0) are shipped with polymake under the GNU General Public License (GPL). As far as ppl is concerned the polymake distribution only comes with a bare interface, i.e., without the ppl code. We used the ppl library version 1.1.

The external software packages used via a file based interfaces are 4ti2 version 1.6.6, azove version 2.0, LattE version 1.7.3 and porta version 1.4.1-20090921.

The GMP was configured to use the standard memory allocator malloc. Employing a different memory allocator, such as TCMalloc [37], can have a great impact, in particular, in a multi-threaded setting. However, the precise behavior depends on numerous factors. For instance, we observed that TCMalloc was clearly superior to malloc in an openSUSE 12.2 environment, while the difference is only marginal in our setup with openSUSE 13.1, on the same hardware.

The histogram in Fig. 6 was created with MATLAB [41].
### Table 3: Timings (in s) for convex hull computations of non-symmetric cut polytopes $\text{Cut}(G_k)$, see Sect. 3.3

| $k$ | $d$ | $n$ | $m$ | $\text{bb}$ | $\text{cdd}$ | $\text{lrs}$ | $\text{normaliz}$ | $\text{porta}$ | $\text{ppl}$ |
|-----|-----|-----|-----|-----------|-----------|-----------|---------------|-------------|-------------|
|     |     |     |     | 1 Thread | 6 Threads |           |               |             |             |
| 0   | 6   | 32  | 20  | 0.00     | 0.005     | 0.005     | 0.001         | 0.001       | 0.065       | 0.001       |
| 1   | 7   | 64  | 22  | 0.03     | 0.015     | 0.060     | 0.001         | 0.002       | 0.089       | 0.002       |
| 2   | 8   | 128 | 24  | 0.16     | 0.050     | 0.937     | 0.003         | 0.005       | 0.107       | 0.004       |
| 3   | 9   | 256 | 26  | 1.25     | 0.184     | 18.195    | 0.006         | 0.010       | 0.246       | 0.008       |
| 4   | 10  | 512 | 28  | 14.41    | 0.700     | 329.093   | 0.014         | 0.021       | 0.916       | 0.018       |
| 5   | 11  | 1024| 30  | ▲         | 2.877     | 7699.132* | 0.038         | 0.047       | 7.980       | 0.043       |
| 6   | 12  | 2048| 32  | –        | 12.086    | –         | 0.114         | 0.114       | 98.987      | 0.110       |
| 7   | 13  | 4096| 34  | –        | 50.064    | –         | 0.390         | 0.293       | 1810.960    | 0.298       |
| 8   | 14  | 8192| 36  | –        | 210.074   | –         | 1.425         | 0.841       | 36,729.820* | 0.890       |
| 9   | 15  | 16,384| 38  | –       | 974.681   | –         | 5.514         | 2.799       | –           | 3.182       |
| 10  | 16  | 32,768| 40  | –       | –        | –         | 22.310        | 10.233      | –           | 12.680      |
| 11  | 17  | 65,536| 42  | –       | –        | –         | 93.935        | 42.461      | –           | 53.547      |
| 12  | 18  | 131,072| 44  | –       | –        | –         | 399.652       | 181.790     | –           | 216.302     |
| 13  | 19  | 262,144| 46  | –       | –        | –         | 1681.473      | 952.263     | –           | 1148.847    |
| 14  | 20  | 524,288| 48  | –       | –        | –         | 7173.680*     | 4843.120*   | –           | 3641.770    |
Table 4 Running times (in seconds) for integer hull computations of Fibonacci knapsack polytopes $F_d(b)$, see Sect. 3.4.

| $d \setminus b$ | 40   | 50   | 60   | 70   | 80   | 90   | 100  |
|-----------------|------|------|------|------|------|------|------|
| bb              |      |      |      |      |      |      |      |
| 4               | 0.128| 0.339| 0.873| 1.921| 3.127| 7.507| 16.265|
| 5               | 0.358| 1.268| 4.273| 15.039|42.077|136.823|355.191|
| 6               | 0.843| 3.878|21.148|76.548|305.121|1071.274|3294.703|
| bb (random permutation of input) | | | | | | | |
| 4               | 0.069| 0.138| 0.295| 0.572| 1.035| 2.023| 3.738|
| 5               | 0.220| 0.555| 1.203| 2.901| 6.861|16.215| 35.602|
| 6               | 0.596| 1.844| 4.495|11.417|27.508| 59.907|152.199|
| bb (vertices first) | | | | | | | |
| 4               | 0.026| 0.052| 0.107| 0.239| 0.448| 0.933| 2.080|
| 5               | 0.054| 0.140| 0.363| 1.188| 3.457| 9.689| 23.175|
| 6               | 0.109| 0.253| 0.813| 3.193|10.093|29.070| 86.741|
| cdd             |      |      |      |      |      |      |      |
| 4               | 2.022| 8.992|30.877|90.281|231.238|537.414|1173.495|
| 5               | 5.667|30.601|126.978|447.216|1352.025|3594.319|8737.240*|
| 6               | 9.392|58.682|274.606|1092.438|3787.679| – | – |
| cdd (random permutation of input) | | | | | | | |
| 4               | 2.168| 9.794|34.103|100.972|266.833|717.830|1600.367|
| 5               | 6.195|33.444|139.920|555.176|1837.662|4928.060| – | |
| 6               | 10.379|64.724|313.493|1501.114|5145.730| – | – |
| cdd (vertices first) | | | | | | | |
| 4               | 2.229| 9.800|33.682|99.496|282.405|717.921|1482.070|
| 5               | 6.162|33.009|137.282|579.661|1707.650|4614.850| – | |
| 6               | 10.132|63.107|307.064|1448.336|4777.220| – | – |
| lrs             |      |      |      |      |      |      |      |
| 4               | 5.949|32.270|132.774|448.725|1262.392|3244.181|7583.130*|
| 5               | 27.849|213.319|1163.065|4957.700*| – | – | – |
| 6               | 63.492|648.386|4527.030*| – | – | – | – |
| lrs (random permutation of input) | | | | | | | |
| 4               | 0.385| 1.317| 3.829| 8.170| 20.089| 39.051| 78.252|
| 5               | 2.108| 9.002|32.371|111.586|287.381|599.524| 987.394|
| 6               | 5.343|32.730|170.878|644.030|1624.175|4069.540*| – | |
| lrs (vertices first) | | | | | | | |
| 4               | 6.765|40.089|160.902|549.346|1837.019|5302.680*| – | |
| 5               | 32.923|259.961|1516.097|7975.330*| – | – | – |
| 6               | 75.724|818.549|6345.470*| – | – | – | – |
| nmz             |      |      |      |      |      |      |      |
| 4               | 0.275| 0.805| 2.103| 5.622| 13.681| 29.505| 65.861|
| 5               | 0.565| 2.377| 8.096| 26.586| 74.743| 194.972| 421.936|
| 6               | 0.949| 3.910|17.758|68.364|184.154|438.641|1100.974|

\* Indicates that the computation was not completed within the time limit of 2 hours.
### Table 4 continued

| d\b | 40   | 50   | 60   | 70   | 80   | 90   | 100  |
|-----|------|------|------|------|------|------|------|
| porta |      |      |      |      |      |      |      |
| 4   | 0.070 | 0.180 | 0.410 | 0.965 | 1.734 | 3.495 | 7.696 |
| 5   | 0.195 | 0.685 | 1.868 | 5.363 | 12.614 | 37.579 | 96.675 |
| 6   | 0.416 | 1.471 | 5.581 | 21.792 | 64.878 | 180.298 | 521.574 |
| porta (random permutation of input) |      |      |      |      |      |      |      |
| 4   | 0.176 | 0.534 | 1.302 | 2.930 | 5.555 | 8.624 | 15.973 |
| 5   | 0.533 | 2.208 | 5.117 | 17.558 | 37.111 | 99.375 | 181.517 |
| 6   | 1.163 | 5.903 | 22.705 | 78.768 | 245.711 | 656.937 | 1634.647 |
| porta (vertices first) |      |      |      |      |      |      |      |
| 4   | 0.035 | 0.073 | 0.133 | 0.258 | 0.337 | 0.501 | 0.940 |
| 5   | 0.076 | 0.213 | 0.349 | 0.685 | 0.854 | 2.278 | 4.443 |
| 6   | 0.169 | 0.311 | 0.793 | 2.073 | 3.141 | 5.174 | 9.992 |
| ppl |      |      |      |      |      |      |      |
| 4   | 0.028 | 0.050 | 0.096 | 0.181 | 0.302 | 0.524 | 0.945 |
| 5   | 0.048 | 0.123 | 0.274 | 0.647 | 1.361 | 3.472 | 8.084 |
| 6   | 0.077 | 0.215 | 0.647 | 2.074 | 5.531 | 14.517 | 40.975 |

### Table 5 Running times (in seconds) for Voronoi diagrams of random point sets, see Sect. 3.5

| d\m | 500   | 1000  | 1500  | 2000  | 2500  | 3000  |
|-----|-------|-------|-------|-------|-------|-------|
| bb  |       |       |       |       |       |       |
| 3   | 0.261 | 0.621 | 1.040 | 1.517 | 2.010 | 2.569 |
| 4   | 1.415 | 3.102 | 4.875 | 6.705 | 8.622 | 10.547 |
| 5   | 11.144 | 25.280 | 40.011 | 55.229 | 69.848 | 85.517 |
| 6   | 100.962 | 255.429 | 425.652 | 608.085 | 800.647 | 995.853 |
| 7   | 1318.488 | 4245.600* | – | – | – | – |
| cdd |       |       |       |       |       |       |
| 3   | 6.604 | 28.420 | 67.606 | 124.657 | 200.765 | 295.799 |
| 4   | 53.436 | 246.447 | 599.624 | 1124.073 | 1817.252 | 2712.942 |
| 5   | 326.138 | 1686.490 | 4315.200* | – | – | – |
| 6   | 2679.876 | – | – | – | – | – |
| 7   | – | – | – | – | – | – |
| lrs |       |       |       |       |       |       |
| 3   | 0.496 | 2.149 | 5.034 | 9.147 | 14.741 | 21.264 |
| 4   | 1.955 | 8.677 | 20.541 | 37.901 | 60.886 | 89.300 |
| 5   | 11.744 | 50.578 | 119.551 | 220.404 | 354.748 | 526.344 |
| 6   | 63.601 | 318.346 | 798.688 | 1530.386 | 2537.572 | 3801.890* |
| 7   | 404.055 | 2063.702 | – | – | – | – |
### Table 5

| $d \setminus m$ | \hspace{1em}500\hspace{1em} | \hspace{1em}1000\hspace{1em} | \hspace{1em}1500\hspace{1em} | \hspace{1em}2000\hspace{1em} | \hspace{1em}2500\hspace{1em} | \hspace{1em}3000\hspace{1em} |
|---------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| **normaliz**  |                 |                 |                 |                 |                 |                 |
| 3             | 0.605           | 3.683           | 11.179          | 24.950          | 44.928          | 68.525          |
| 4             | 3.108           | 21.365          | 52.959          | 100.007         | 165.561         | 249.134         |
| 5             | 33.374          | 151.977         | 365.866         | 671.886         | 1085.843        | 1644.610        |
| 6             | 219.029         | 1013.544        | 2436.256        | –               | –               | –               |
| 7             | 1287.456        | –               | –               | –               | –               | –               |
| **ppl**       |                 |                 |                 |                 |                 |                 |
| 3             | 0.289           | 1.355           | 3.757           | 7.876           | 13.947          | 22.495          |
| 4             | 2.508           | 14.480          | 45.456          | 100.803         | 186.210         | 319.268         |
| 5             | 41.547          | 277.779         | 935.003         | 2255.994        | 4334.870*       | –               |
| 6             | 917.853         | 6762.370*       | –               | –               | –               | –               |
| 7             | –               | –               | –               | –               | –               | –               |

### Table 6

Timings (in sec.) for counting lattice points in $F_5(b)$; Sect. 4.3

| $b$ | $n$ | $\ell$ | 4ti2 | bbox | LatE | normaliz | proj |
|-----|-----|--------|------|------|------|----------|------|
|     |     |        |      |      |      | Default  | All primal | Primal | Dual |
| 40  | 6   | 1366   | 0.500| 0.247| 0.012| 0.028    | 0.048 | 0.007 | 0.011|
| 50  | 6   | 3137   | 3.331| 0.501| 0.014| 0.028    | 0.024 | 0.014 | 0.023|
| 60  | 6   | 6509   | 22.171| 1.193| 0.013| 0.027    | 0.212 | 0.039 | 0.039|
| 70  | 6   | 12,182 | 116.258| 2.420| 0.014| 0.027    | 0.069 | 0.080 | 0.063|
| 80  | 6   | 21,245 | 515.834| 4.928| 0.013| 0.027    | 0.121 | 0.160 | 0.100|
| 90  | 6   | 35,025 | 1657.244| 7.766| 0.014| 0.030    | 0.197 | 0.300 | 0.152|
| 100 | 6   | 55,157 | –     | 12.787| 0.013| 0.027    | 0.329 | 0.555 | 0.216|
| 110 | 6   | 83,616 | –     | 20.288| 0.012| 0.028    | 0.523 | 0.931 | 0.305|
| 120 | 6   | 122,749| –     | 33.606| 0.015| 0.028    | 0.745 | 1.499 | 0.414|
| 130 | 6   | 175,306| –     | 49.113| 0.013| 0.028    | 1.173 | 2.342 | 0.556|
| 140 | 6   | 244,473| –     | 64.496| 0.013| 0.029    | 1.689 | 3.473 | 0.727|
| 150 | 6   | 333,905| –     | 92.070| 0.013| 0.028    | 2.304 | 5.120 | 0.943|
| 160 | 6   | 447,757| –     | 132.536| 0.014| 0.029    | 3.147 | 7.150 | 1.229|
| 170 | 6   | 590,715| –     | 177.143| 0.013| 0.027    | 4.395 | 10.116| 1.549|
| 180 | 6   | 768,029| –     | 222.824| 0.014| 0.029    | 5.553 | 13.457| 1.907|
| 190 | 6   | 985,546| –     | 291.333| 0.012| 0.028    | 7.214 | 18.045| 2.352|
| 200 | 6   | 1,249,741| –   | 387.886| 0.014| 0.028    | 9.235 | 24.116| 2.880|
Table 7  Timings (in sec.) for counting lattice points in $F_d(60)$; Sect. 4.3

| $d$ | $n$ | $\ell$ | 4ti2 | bbox | LattE | normaliz | proj |
|-----|-----|-------|------|------|-------|----------|------|
|     |     |       |      |      |       | Default  | All primal | Primal | Dual |
| 4   | 5   | 4008  | 7.722| 0.236| 0.007 | 0.009    | 0.011  | 0.019 | 0.011 |
| 5   | 6   | 6509  | 22.173| 1.209| 0.015 | 0.027    | 0.193  | 0.036 | 0.040 |
| 6   | 7   | 7853  | 33.795| 3.861| 0.030 | 0.113    | 0.070  | 0.047 | 0.090 |
| 7   | 8   | 8165  | 43.212| 8.397| 0.077 | 0.504    | 0.148  | 0.054 | 0.153 |
| 8   | 9   | 8171  | 44.388| 18.088| 0.188 | 2.120    | 0.287  | 0.054 | 0.233 |
| 9   | 10  | 8171  | 43.496| 19.418| 0.507 | 11.207   | 0.365  | 0.060 | 0.319 |
| 10  | 11  | 8171  | 43.577| 20.616| 1.315 | 44.764   | 0.434  | 0.064 | 0.413 |
| 11  | 12  | 8171  | 44.137| 21.919| 3.521 | 255.896  | 0.502  | 0.066 | 0.524 |
| 12  | 13  | 8171  | 44.183| 23.121| 9.238 | 2357.660 | 0.587  | 0.069 | 0.639 |
| 13  | 14  | 8171  | 44.413| 24.264| 24.864 | 36,124,990* | 0.689  | 0.073 | 0.767 |
| 14  | 15  | 8171  | 44.275| 25.290| 66.012 | –       | 16.999 | 0.074 | 0.908 |
| 15  | 16  | 8171  | 44.953| 26.550| 177.424 | –       | 18.982 | 0.077 | 1.053 |
| 16  | 17  | 8171  | 44.967| 27.808| 467.912 | –       | 21.059 | 0.082 | 1.210 |
| 17  | 18  | 8171  | 45.149| 28.907| 1246.255 | –      | 22.979 | 0.084 | 1.383 |
| 18  | 19  | 8171  | 45.115| 30.089| 3282.651 | –      | 25.063 | 0.084 | 1.571 |
| 19  | 20  | 8171  | 45.233| 31.594| –      | –       | 27.296 | 0.090 | 1.770 |
| 20  | 21  | 8171  | 45.396| 32.797| –      | –       | 29.629 | 0.092 | 1.975 |

Table 8  Running times (in second) for counting lattice points in random polytopes; Sect. 4.4

| $d\backslash b$ | 20   | 30   | 40   | 50   | 60   | 70   |
|----------------|------|------|------|------|------|------|
| bbox           |      |      |      |      |      |      |
| 4              | 0.082| 0.130| 0.154| 0.190| 0.207| 0.213|
| 5              | 0.756| 1.439| 2.548| 3.116| 3.766| 4.845|
| 6              | 4.298| 11.853| 18.827| 34.396| 47.261| 63.581|
| 7              | 20.466| 52.392| 144.850| 236.414| 400.435| 581.584|
| 8              | 89.321| 270.244| 679.447| 1693.210| 2929.315| –    |
| 9              | 473.808| 1005.608| 3511.637| –    | –    | –    |
| LattE          |      |      |      |      |      |      |
| 4              | 1.798| 1.639| 1.419| 1.357| 1.405| 1.172|
| 5              | 438.684| 612.918| 534.548| 588.029| 507.007| 404.041|
| 6              | –    | –    | –    | –    | –    | –    |
| 7              | –    | –    | –    | –    | –    | –    |
| 8              | –    | –    | –    | –    | –    | –    |
| 9              | –    | –    | –    | –    | –    | –    |
| $d \setminus b$ | 20   | 30   | 40   | 50   | 60   | 70   |
|-----------------|------|------|------|------|------|------|
| **LattE (all primal)** |      |      |      |      |      |      |
| 4               | 0.252| 0.375| 0.490| 0.512| 0.569| 0.672|
| 5               | 3.111| 4.651| 6.690| 8.114| 9.516| 11.017|
| 6               | 49.751| 122.619| 154.804| 235.039| 281.204| 338.060|
| 7               | 801.512| 3016.324| 4459.500*| –   | –   | –   |
| 8               | 22,332.390*| –   | –   | –   | –   | –   |
| 9               | –   | –   | –   | –   | –   | –   |
| **normaliz**    |      |      |      |      |      |      |
| 4               | 0.002| 0.003| 0.004| 0.006| 0.006| 0.005|
| 5               | 0.005| 0.007| 0.013| 0.020| 0.022| 0.025|
| 6               | 0.017| 0.040| 0.085| 0.129| 0.175| 0.237|
| 7               | 0.098| 0.426| 0.782| 1.307| 1.941| 2.392|
| 8               | 1.068| 3.214| 8.565| 16.734| 24.145| 34.041|
| 9               | 5.256| 50.533| 129.305| 234.613| 474.163| 626.180|
| **projection**  |      |      |      |      |      |      |
| 4               | 0.013| 0.023| 0.029| 0.035| 0.042| 0.040|
| 5               | 0.100| 0.232| 0.462| 0.621| 0.762| 1.032|
| 6               | 0.450| 1.806| 3.945| 6.306| 10.476| 14.247|
| 7               | 1.369| 9.502| 28.062| 57.688| 102.579| 149.536|
| 8               | 3.414| 41.705| 158.855| 138.180| 148.227| 223.361|
| 9               | 7.490| 151.081| 219.040| 248.157| 769.460| 1017.670|
Table 9  Timings (in seconds) for counting lattice points in the matching polytope of $K_n$, see Sect. 4.5

| $n$ | $m$ | 4ti2 | azove | bbox | LatTE  | normaliz | proj |
|-----|-----|------|-------|------|--------|----------|------|
|     |     |      |       |      | Default| All primal| Primal| Dual |
| 4   | 10  | 0.00 | 0.00  | 0.05 | 0.028  | 0.070    | 0.000 | 0.001 | 0.002 |
| 5   | 15  | 0.00 | 0.00  | 0.051| 0.450  | 16.865   | 0.006 | 0.003 | 0.009 |
| 6   | 21  | 0.00 | 0.003 | 2.430| 13.195 | –        | 0.093 | 0.008 | 0.059 |
| 7   | 28  | 0.18 | 0.004 | 281.244| 531.014| –        | 1782.251| 0.072 | 0.507 |
| 8   | 36  | 12.05| 0.004 | 6138.776*| –        | –        | –    | 2.163 | 4.306 |
| 9   | 45  | 1557.38| 0.004| –    | –        | –        | –    | 95.743| 60.759 |
| 10  | 55  | –    | 0.009 | –    | –        | –        | –    | 558.374| –     |
| 11  | 66  | –    | 0.017 | –    | –        | –        | –    | –    | –     |
| 12  | 78  | –    | 0.062 | –    | –        | –        | –    | –    | –     |
| 13  | 91  | –    | 0.236 | –    | –        | –        | –    | –    | –     |
| 14  | 105 | –    | 1.046 | –    | –        | –        | –    | –    | –     |
| 15  | 120 | –    | 4.696 | –    | –        | –        | –    | –    | –     |
| 16  | 136 | –    | 23.224| –    | –        | –        | –    | –    | –     |
| 17  | 153 | –    | 112.785| –    | –        | –        | –    | –    | –     |
| 18  | 171 | –    | 578.191| –    | –        | –        | –    | –    | –     |
| 19  | 190 | –    | 2916.652| –    | –        | –        | –    | –    | –     |
| 20  | 210 | –    | 15,462.050*| –    | –        | –        | –    | –    | –     |
References

1. 4ti2 team, 4ti2—A software package for algebraic, geometric and combinatorial problems on linear spaces. http://www.4ti2.de (2015)
2. Achterberg, T.: SCIP: solving constraint integer programs. Math. Program. Comput. 1(1), 1–41. http://mpc.zib.de/index.php/MPC/article/view/4 (2009)
3. Avis, D.: lrslib, version 6.0. http://cgm.cs.mcgill.ca/~avis/C/lrs.html (2015)
4. Avis, D., Bremner, D. and Seidel, R.: How good are convex hull algorithms? Comput. Geom. 7(5–6), 265–301 (1997) (11th ACM Symposium on Computational Geometry (Vancouver, BC, 1995))
5. Avis, D., Fukuda, K.: A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete Comput. Geom. 8(3), 295–313 (1992) (ACM Symposium on Computational Geometry (North Conway, NH, 1991))
6. Avis, D., Fukuda, K.: Reverse search for enumeration, Discrete Appl. Math. 65(1–3), 21–46 (1996) (First International Colloquium on Graphs and Optimization (GOI), 1992 (Grimentz))
7. Avis, D., Imai, H., Ito, T.: Generating facets for the cut polytope of a graph by triangular elimination. Math. Program. Ser. A 112(2), 303–325 (2008)
8. Avis, D., Jordan, C.: Comparative computational results for some vertex and facet enumeration codes, preprint arXiv:1510.02545 (2015)
9. Bagnara, R., Hill, P. M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comp. Program. 72(1–2), 3–21 (2008)
10. Baldoni, V., Berline, N., De Loera, J., Dutra, B., Koepe, M., Vergne, M.: Coefficients of Sylvester’s denumerant, Integers 15, Paper No. A11 (2015)
11. Barahona, F.: The max-cut problem on graphs not contractible to $K_5$, Oper. Res. Lett. 2(3), 107–111 (1983)
12. Barvinok, A. I.: A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. Math. Oper. Res. 19(4), 769–779 (1994)
13. Behle, M.: On threshold BDDs and the optimal variable ordering problem, Combinatorial Optimization and Applications. In: Andreas, D., Yin, X., Binhai, Z. (eds) Lecture Notes in Computer Science, vol. 4616, pp. 45–71, Springer Berlin Heidelberg (English) (2007)
14. Bejraburnin, N.: On facet inequalities of correlation polytopes, http://math.berkeley.edu/~natth/files/STAT241A.pdf
15. Karl, H.B.: Average-case analysis of the double description method and the beneath-beyond algorithm. Discrete Comput. Geom. 37(2), 175–204 (2007)
16. Bremner, D.: Incremental convex hull algorithms are not output sensitive. Discrete Comput. Geom. 21(1), 57–68 (1999)
17. Bremner, D., Dutour Sikirić, M., Schürmann, A.: Polyhedral representation conversion up to symmetries. Polyhedral computation, CRM Proc. Lecture Notes, vol. 48, Amer. Math. Soc., Providence, RI, pp. 45–71 (2009)
18. Bruns, W., Ichim, B., Söger, C.: The power of pyramid decomposition in Normaliz, J. Symbolic Comput. 74, 513–536 (2016)
19. Bruns, W., Ichim, B., Söger, C., Römer, T.: Normaliz. Algorithms for rational cones and affine monoids, version 2.9.9, http://www.math.uos.de/normaliz (2015)
20. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension. Discrete Comput. Geom. 10(4), 377–409 (1993)
21. Chernikova, N.V.: Algorithm for finding a general formula for the non-negative solutions of system of linear equations. USSR Comput. Math. Math. Phys. 4(4), 151–158 (1964)
22. Christoph, T.: SMAPO, a library of linear descriptions of low-dimensional 0/1-polytopes connected with small instances of combinatorial optimization problems. http://www.iwr.uni-heidelberg.de/groups/comopt/software/SMAPOCutCut.html (2010)
23. Christoph, T., Loebel, A.: PORTA, version 1.4.1-20090921. http://porta.zib.de/ (2009)
24. IBM Corporation, ILOG CPLEX, http://www.ibm.com/software/commerce/optimization/cplex-optimizer/
25. De Loera, J.A.: ‘The many aspects of counting lattice points in polytopes. Math. Semesterber. 52(2), 175–195 (2005)
26. De Loera, J.A., Berline, N., Baldoni, V., Dutra, B., Köppe, M., Moreinis, S., Pinto, G., Vergne, M., Wu, J.: A user’s guide for LattE integrale v1.7.3, software package LattE is available at http://www.math.ucdavis.edu/~lattice/, (2015)
27. De Loera, J. A., Dutra, B., Köppe, M., Moreinis, S., Pinto, G., Wu, J.: Software for exact integration of polynomials over polyhedra. Comput. Geom. 46(3), 232–252 (2013)
28. Dewey, C., Woods, K.: Parametric Sequence Alignment, Algebraic Statistics for Computational Biology. Cambridge Univ. Press, New York (2005)
29. Edelsbrunner, H.: Algorithms in combinatorial geometry, EATCS Monographs on Theoretical Computer Science, vol. 10. Springer-Verlag, Berlin (1987)
30. Koch, T. et al.: MIPLIB 2010. Math. Program. Comput. 3(2) 103–163 (english), http://miplib.zib.de/miplib2010.php (2011)
31. Granlund, T. et al.: GNU multiple precision arithmetic library 5.1.2, http://gmplib.org/
32. Fukuda, K.: cddlib, version 0.94h, http://www.inf.ethz.ch/personal/fukudak/cdd_home/ (2015)
33. Fukuda, K., Prodon, A.: Double description method revisited, Combinatorics and computer science (Brest 1995), Lecture Notes in Comput. Sci., vol. 1120, pp 91–111, Springer, Berlin (1996)
34. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
35. Gawrilow, E., Joswig, M.: Polymake: a framework for analyzing convex polytopes, Polytopes–combinatorics and computation (Oberwolfach 1997), DMV Sem., vol. 29, pp. 43–73, Birkhäuser, Basel, (2000)
36. Gawrilow, E., Joswig, M.: Flexible object hierarchies in polymake. In: Andrés, I., Nobuki, T. (eds.) Proceedings of the 2nd International Congress of Mathematical Software, 1–3. September 2006, pp. 219–221, Castro Urdiales, Spanien (2006)
37. Ghemawat, S., Menage, P.: Thread-CachingMalloc part of the Google Performance Tools. http://google-perftools.sourceforge.net/doc/tcmalloc.html
38. Gurobi Optimization, Inc., Gurobi optimizer reference manual (2013)
39. Hayes, A.C., David, G., Larman, : The vertices of the knapsack polytope. Discrete Appl. Math. 6(2), 135–138 (1983)
40. Hemmecke, R.: On the computation of Hilbert bases of cones, Mathematical software (Beijing, 2002) World Sci. Publ. River Edge, NJ 2002, pp. 307–317
41. The MathWorks Inc., MATLAB version 8.2.0.701 (r2013b), http://mathworks.com, Natick, Massachusetts (2010)
42. Joswig, M.: Beneath-and-Beyond Revisited, Algebra, Geometry, and Software Systems. Springer, Berlin (2003)
43. Joswig, M., Loho, G., Lorenz, B., Schröter, B.: Linear programs and convex hulls over fields of Puiseux fractions. In: Mathematical Aspects of Computer and Information Sciences: 6th International Conference, MACIS 2015, Berlin, Germany, November 11–13, 2015. Revised Selected Papers, Lecture Notes in Comput. Sci., vol. 9582, pp 429–445, Springer, Berlin (2016)
44. Joswig, M., Theobald, T.: Polyhedral and algebraic methods in computational geometry, Universitext, Springer, London, 2013, Revised and updated translation of the 2008 German original
45. Joswig, M., Ziegler, G. M.: Convex hulls, oracles, and homology. J. Symbolic Comput. 38(4), 1247–1259 (2004)
46. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V.: Generating all vertices of a polyhedron is hard. Discrete Comput. Geom. 39(1–3), 174–190 (2008)
47. Klee, V., Minty, G.J.: How good is the simplex algorithm? Inequalities III. In: Proceedings of Third Symposium University California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin, pp. 159–175, Academic Press, New York (1972)
48. Koepe, M., Verdoolaege, S., Woods, K.: An implementation of the Barvinok-Woods integer projection algorithm. In: Proceedings of the International Conference on Information Theory and Statistical Learning, pp. 53–59 (2008)
49. Körpe, M.: A primal Barvinok algorithm based on irrational decompositions. SIAM J. Discrete Math. 21(1), 220–236 (electronic) (2007)
50. McMullen, P.: The numbers of faces of simplicial polytopes. Israel J. Math. 9, 559–570 (1971)
51. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R.M.: The double description method, Contributions to the theory of games, vol. 2, Annals of Mathematics Studies, no. 28, pp. 51–73, Princeton University Press, Princeton, (1953)
52. Murai, S., Nevo, E.: On the generalized lower bound conjecture for polytopes and spheres. Acta Math. 210(1), 185–202 (2013)
53. Opfer, T.: Entwicklung eines exakten rationalen dualen Simplex-Lösers, Master’s thesis, TU Darmstadt, (2011)
54. The polymake team, Computing convex hulls and counting integer points with polymake, http://www.polymake.org/doku.php/researchdata/polymakeilp, Experimental data and source code (2015)
55. The polymake team, polymake, version 2.14, http://www.polymake.org (2015)
56. Loïc, P.: The Euclidean algorithm in dimension $n$, ISSAC ’96. Proceedings of the 1996 international symposium on Symbolic and algebraic computation, pp. 40–42 (1996)
57. Rehn, T.: Polyhedral description conversion up to symmetries, Master’s thesis, Otto-von-Guericke-Universität Magdeburg, (2010)
58. Schrijver, A.: Combinatorial optimization. Polyhedra and efficiency. Vol. C, Algorithms and Combinatorics, vol. 24, pp. 70–83, Springer-Verlag, Berlin, Disjoint paths, hypergraphs, Chapters (2003)
59. Seidel, R.: A convex hull algorithm optimal for point sets in even dimensions. Tech. report, University of British Columbia, Department of Computer Science (1981)
60. Tantau, T.: TikZ ist kein Zeichenprogramm PGF/TikZ, http://www.ctan.org/tex-archive/graphics/pgf/
61. The CGAL Project, CGAL user and reference manual, 4.7 ed., CGAL Editorial Board, (2015)
62. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. 8(3), 410–421 (1979)
63. Verdoolaege, S.: barvinok, http://barvinok.gforge.inria.fr/
64. Walter, M.: Unimodularity-test extension for polymake, https://github.com/xammy/unimodularity-test
65. Walter, M., Truemper, K.: Implementation of a unimodularity test. Math. Program. Comput. 5(1), 57–73 (2013)
66. Ziegler, G.M.: Lectures on polytopes, Graduate Texts in Mathematics, vol. 152. Springer-Verlag, New York (1995)