Journal of Symbolic Computation 74 (2016) 513–536

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

The power of pyramid decomposition in Normaliz

Winfried Bruns [a], Bogdan Ichim [b], Christof Söger [a]

[a] Universität Osnabrück, FB Mathematik/Informatik, 49069 Osnabrück, Germany
[b] Simion Stoilow Institute of Mathematics of the Romanian Academy, Research Unit 5, C.P. 1-764, 010702 Bucharest, Romania

A R T I C L E   I N F O

Article history:
Received 27 January 2014
Accepted 26 August 2015
Available online 16 September 2015

MSC:
52B20
13F20
14M25
91B12

Keywords:
Hilbert basis
Ehrhart series
Hilbert series
Rational polytope
Volume
Triangulation
Pyramid decomposition

A B S T R A C T

We describe the use of pyramid decomposition in Normaliz, a software tool for the computation of Hilbert bases and enumerative data of rational cones and affine monoids. Pyramid decomposition in connection with efficient parallelization and streamlined evaluation of simplicial cones has enabled Normaliz to process triangulations of size $\approx 5 \cdot 10^{11}$ that arise in the computation of Ehrhart series related to the theory of social choice.

1. Introduction

Normaliz (Bruns et al., 2015) is a software tool for the computation of Hilbert bases and enumerative data of rational cones and affine monoids. In the 17 years of its existence it has found numerous applications; for example, in integer programming (Bogart et al., 2010), algebraic geometry (Craw et al., 2007), theoretical physics (Kappl et al., 2011), commutative algebra (Sturmfels and Welker, 2012) or elimination theory (Emiris et al., 2013). Normaliz is used in polymake (Joswig et al., 2009), a com-

E-mail addresses: wbruns@uos.de (W. Bruns), bogdan.ichim@imar.ro (B. Ichim), csoeger@uos.de (C. Söger).

http://dx.doi.org/10.1016/j.jsc.2015.09.003
0747-7171/

puter system for polyhedral geometry, and in Regina (Burton, 2014), a system for computations with 3-manifolds.

The mathematics of the very first version was described in Bruns and Koch (2001), and the details of version 2.2 (2009) are contained in Bruns and Ichim (2010). In this article we document the mathematical ideas and the most recent development[1] resulting from them. It has extended the scope of Normaliz by several orders of magnitude.

In algebraic geometry the spectra of algebras $K[C \cap L]$ where C is a pointed cone and L a lattice, both contained in a space $\mathbb{R}^d$, are the building blocks of toric varieties; for example, see Cox et al. (2011). In commutative algebra the algebras $K[C \cap L]$ which are exactly the normal affine monoid algebras are of interest themselves. It is clear that an algorithmic approach to toric geometry or affine monoid algebras depends crucially on an efficient computation of the unique minimal system of generators of a monoid $C \cap L$ that we call its *Hilbert basis*. Affine monoids of this type are extensively discussed by Bruns and Gubeladze (2009). The existence and uniqueness of such a minimal system of generators is essentially due to Gordan (1873) and was proven in full generality by van der Corput (1931).

The computation of Hilbert bases amounts to solving homogeneous linear diophantine systems of inequalities (defining the cone) and equations and congruences (defining the lattice). Since version 2.11 Normaliz also solves inhomogeneous linear diophantine systems; in other words, it computes lattice points in polyhedra (and not just cones).

The term "Hilbert basis" was actually coined in integer programming (with $L = \mathbb{Z}^d$) by Giles and Pulleyblank (1979) in connection with totally dual integral (TDI) systems. Also see Schrijver (1998, Sections 16.4 and 22.3). One should note that in integer programming usually an arbitrary, not necessarily minimal, system of generators of $C \cap \mathbb{Z}^d$ is called a Hilbert basis of C. From the computational viewpoint and also in bounds for such systems of generators, minimality is so important that we include it in the definition. Aardal et al. (2002) discuss Hilbert bases and their connection with Graver bases (of sublattices) and Gröbner bases (of binomial ideals). (At present, Normaliz does not include Graver or Gröbner bases; 4ti2, 4ti2 team, 2015, is a tool for their computation.) It should be noted that Normaliz, or rather a predecessor, was instrumental in finding a counterexample to the Integral Carathéodory Property (Bruns et al., 1999) that was proposed by Sebő (1990). For more recent developments in nonlinear optimization using Graver bases, and therefore Hilbert bases, see De Loera et al. (2009), Hemmecke et al. (2011, 2014).

Hilbert functions and polynomials of graded algebras and modules were introduced by Hilbert himself (Hilbert, 1890) (in contrast to Hilbert bases). These invariants, and the corresponding generating functions, the Hilbert series, are fundamental in algebraic geometry and commutative algebra. See Bruns and Gubeladze (2009, Chapter 6) for a brief introduction to this fascinating area. Ehrhart functions were defined by Ehrhart (1977) as lattice point counting functions in multiples of rational polytopes; see Beck and Robbins (2007) for a gentle introduction. Stanley (1996) interpreted Ehrhart functions as Hilbert functions, creating a powerful link between discrete convex geometry and commutative algebra. In the last decades Hilbert functions have been the objective of a large number of articles. They even come up in optimization problems; for example, see De Loera et al. (2006). Surprisingly, Ehrhart functions have an application in compiler optimization; see Clauss et al. (1998) for more information.

From the very beginning Normaliz has used lexicographic triangulations; see Bruns and Ichim (2010), Bruns and Koch (2001) for the use in Normaliz and De Loera et al. (2010) for (regular) triangulations of polytopes. (Since version 2.1 Normaliz contains a second, triangulation free Hilbert basis algorithm, originally due to Pottier, 1996 and called *dual* in the following; see Bruns and Ichim, 2010). Lexicographic triangulations are essentially characterized by being incremental in the following sense. Suppose that the cone C is generated by vectors $x_1, \dots, x_n \in \mathbb{R}^d$; set $C_0 = 0$ and $C_i = \mathbb{R}_+ x_1 + \dots + \mathbb{R}_+ x_i$, $i = 1, \dots, n$. Then the lexicographic triangulation Λ (for the ordered system $x_1, \dots, x_n$) restricts to a triangulation of C_i for $i = 0, \dots, n$. Lexicographic triangulations are easy to compute, and go very well with Fourier–Motzkin elimination that computes the support hyperplanes

[1] Version 3.0 is available from http://www.math.uos.de/normaliz.

of C by successive extension from C_i to C_{i+1}, $i = 0, \dots, n-1$. The triangulation Λ_i of C_i is extended to C_{i+1} by all simplicial cones $F + \mathbb{R}_+ x_{i+1}$ where $F \in \Lambda_i$ is visible from x_{i+1}.

As simple as the computation of the lexicographic triangulation is, the algorithm in the naive form just described has two related drawbacks: (i) one must store Λ_i and this becomes very difficult for sizes $\geq 10^8$; (ii) in order to find the facets F that are visible from x_{i+1} we must match the simplicial cones in Λ_i with the support hyperplanes of C_i that are visible from x_{i+1}. While (i) is a pure memory problem, (ii) quickly leads to impossible computation times.

Pyramid decomposition is the basic idea that has enabled Normaliz to compute dimension 24 triangulations of size $\approx 5 \cdot 10^{11}$ in acceptable time on standard multiprocessor systems such as SUN xFire 4450 or Dell PowerEdge R910. Instead of going for the lexicographic triangulation directly, we first decompose C into the pyramids generated by x_{i+1} and the facets of C_i that are visible from x_{i+1}, $i = 0, \dots, n-1$. These pyramids (of level 0) are then decomposed into pyramids of level 1, etc. While the level 0 decomposition need not be a polyhedral subdivision in the strict sense, pyramid decomposition stops after finitely many iterations at the lexicographic triangulation; see Section 3 for the details and Fig. 3 for a simple example.

Pure pyramid decomposition is very memory friendly, but its computation times are often more forbidding than those of pure lexicographic triangulation since too many Fourier–Motzkin eliminations become necessary, and almost all of them are inevitably wasted. That Normaliz can nevertheless cope with extremely large triangulations relies on a well balanced combination of both strategies that we outline in Section 4.

It is an important aspect of pyramid decomposition that it is very parallelization friendly since the pyramids can be treated independently of each other. Normaliz uses OpenMP for shared memory systems. Needless to say that triangulations of the size mentioned above can hardly be reached in serial computation.

For Hilbert basis computations pyramid decomposition has a further and sometimes tremendous advantage: one can avoid the triangulation of those pyramids for which it is a priori clear that they will not supply new candidates for the Hilbert basis. This observation, on which the contribution of the authors to Bruns et al. (2011) (jointly with Hemmecke and Köppe) is based, triggered the use of pyramid decomposition as a general principle. See Remark 4.4 for a brief discussion.

In Section 5 we describe the steps by which Normaliz evaluates the simplicial cones in the triangulation for the computation of Hilbert bases, volumes and Hilbert series. After the introduction of pyramid decomposition, evaluation almost always takes significantly more time than the triangulation. Therefore it must be streamlined as much as possible. For the Hilbert series Normaliz uses a Stanley decomposition (Stanley, 1982). That it can be found efficiently relies crucially on an idea of Köppe and Verdoolaege (2008).

We document the scope of Normaliz's computations in Section 6. The computation times are compared with those of 4ti2 (4ti2 team, 2015) (Hilbert bases) and LattE (2015) (Hilbert series). The test examples have been chosen from the literature (Beck and Hoşten, 2006; Ohsugi and Hibi, 2006; Schürmann, 2013; Sturmfels and Welker 2012), the LattE distribution and the Normaliz distribution. The desire to master the Hilbert series computations asked for in Schürmann's paper (Schürmann, 2013) was an important stimulus in the recent development of Normaliz.

2. Overview of the Normaliz algorithm

The *primal* Normaliz algorithm is triangulation based, as mentioned in the Introduction. Normaliz contains a second, *dual* algorithm for the computation of Hilbert bases that implements ideas of Pottier (1996). The dual algorithm is treated in Bruns and Ichim (2010), and has not changed much in the last years. We skip it in this article, except in Section 6 where computation times of the primal and dual algorithm will be compared.

The primal algorithm starts from a pointed rational cone $C \subset \mathbb{R}^d$ given by a system of generators $x_1, \dots, x_n$ and a sublattice $L \subset \mathbb{Z}^d$ that contains $x_1, \dots, x_n$. (Other types of input data are first transformed into this format.) The algorithm is composed as follows:

1. Initial coordinate transformation to $E = L \cap (\mathbb{R}x_1 + \cdots + \mathbb{R}x_n)$;

2. Fourier–Motzkin elimination computing the support hyperplanes of C;
3. pyramid decomposition and computation of the lexicographic triangulation Δ;
4. evaluation of the simplicial cones in the triangulation:
   (a) enumeration of the set of lattice points E_σ in the fundamental domain of a simplicial subcone σ,
   (b) reduction of E_σ to the Hilbert basis $\operatorname{Hilb}(\sigma)$,
   (c) Stanley decomposition for the Hilbert series of $\sigma \cap L$;
5. Collection of the local data:
   (a) reduction of $\bigcup_{\sigma\in\Delta} \operatorname{Hilb}(\sigma)$ to $\operatorname{Hilb}(C \cap L)$,
   (b) accumulation of the Hilbert series of the $\sigma \cap L$;
6. reverse coordinate transformation to $\mathbb{Z}^d$.

The algorithm does not strictly follow this chronological order, but interleaves steps 2–5 in an intricate way to ensure low memory usage and efficient parallelization. The steps 2 and 5 are treated in Bruns and Ichim (2010), and there is not much to add here, except that 2 is now modified by the pyramid decomposition. Step 3 is described in Sections 3 and 4, and step 4 is the subject of Section 5. In view of the initial and final coordinate transformation we can assume $E = \mathbb{Z}^d$, and suppress the reference to the lattice in the following.

Note that the computation goals of Normaliz can be restricted, for example to the volume of a rational polytope. Then the evaluation of a simplicial cone just amounts to a determinant calculation. Another typical restricted computation goal is the lattice points contained in such a polytope. Then the reduction is replaced by a selection of degree 1 points from the candidate set.

The algorithms described in this paper have been implemented in version 3.0.

3. Lexicographic triangulation and pyramid decomposition

3.1. Lexicographic triangulation

Consider vectors $x_1, \dots, x_n \in \mathbb{R}^d$. For Normaliz these must be integral vectors, but integrality is irrelevant in this section. We want to compute the support hyperplanes of the cone

$$C = \operatorname{cone}(x_1, \dots, x_n) = \mathbb{R}_+x_1 + \cdots + \mathbb{R}_+x_n$$

and a triangulation of C with rays through $x_1, \dots, x_n$. Such a triangulation is a polyhedral subdivision of C into simplicial subcones σ generated by linearly independent subsets of $\{x_1, \dots, x_n\}$.

For a triangulation Σ of a cone C and a subcone C' we set

$$\Sigma|C' = \{\sigma \cap C' : \sigma \in \Sigma\}.$$

In general $\Sigma|C'$ need not be a triangulation of C', but it is so if C' is a face of C.

The *lexicographic* (or *placing*) triangulation $\Lambda(x_1, \dots, x_n)$ of $\operatorname{cone}(x_1, \dots, x_n)$ can be defined recursively as follows: (i) the triangulation of the zero cone is the trivial one, (ii) $\Lambda(x_1, \dots, x_n)$ is given by

$$\Lambda(x_1, \dots, x_n) = \Lambda(x_1, \dots, x_{n-1}) \cup \{\operatorname{cone}(\sigma, x_n) : \sigma \in \Lambda(x_1, \dots, x_{n-1}) \text{ visible from } x_n\}$$

where σ is *visible* from x_n if $x_n \notin \operatorname{cone}(x_1, \dots, x_{n-1})$ and the line segment $[x_n, y]$ for every point y of σ intersects $\operatorname{cone}(x_1, \dots, x_{n-1})$ only in y. Note that a polyhedral complex is closed under the passage to faces, and the definition above takes care of it.

In the algorithms below, a polyhedral subdivision can always be represented by its maximal faces which for convex full dimensional polyhedra are the full dimensional members in the subdivision. For simplicial subdivisions of cones one uses of course that the face structure is completely determined by set theory: every subset E of the set of generators spans a conical face of dimension $|E|$.

We state some useful properties of lexicographic triangulations:

Proposition 1. *With the notation introduced, let* $C_i = \operatorname{cone}(x_1, \dots, x_i)$ *and* $\Lambda_i = \Lambda(x_1, \dots, x_i)$ *for* $i = 1, \dots, n$.

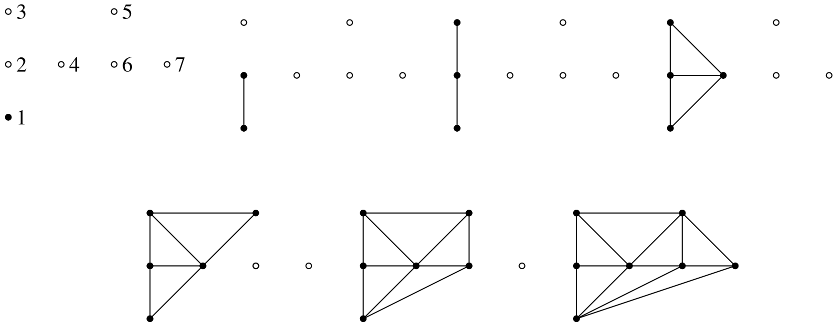

Fig. 1. Genesis of a lexicographic triangulation.

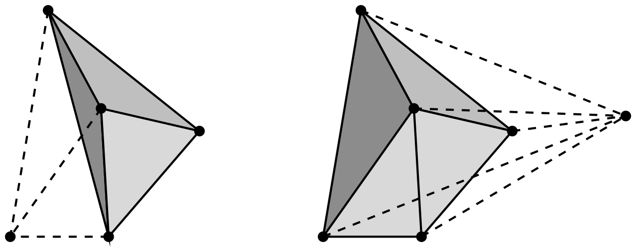

Fig. 2. A lexicographic triangulation in cone dimension 4.

1. *Λ_n is the unique triangulation of C with rays through a subset of $\{x_1, \dots, x_n\}$ that restricts to a triangulation of C_i for $i = 1, \dots, n$ and $\Lambda|C_i$ has rays through a subset of $\{x_1, \dots, x_i\}$.*
2. *For every face F of C the restriction $\Lambda|F$ is the lexicographic triangulation $\Lambda(x_{i_1}, \dots, x_{i_m})$ where $\{x_{i_1}, \dots, x_{i_m}\} = F \cap \{x_1, \dots, x_n\}$ and $i_1 < \dots < i_m$.*
3. *If $\dim C_i > \dim C_{i-1}$, then $\Lambda = \Lambda(x_1, \dots, x_{i-2}, x_i, x_{i-1}, x_{i+1}, \dots, x_n)$.*
4. *$\Lambda = \Lambda(x_{i_1}, \dots, x_{i_d}, x_{j_1}, \dots, x_{j_{n-d}})$ where $(i_1, \dots, i_d)$ is the lexicographic smallest index vector of a rank d subset of $\{x_1, \dots, x_n\}$ and $j_1 < \dots < j_{n-d}$ lists the complementary indices.*

Proof. (1) By construction it is clear that Λ_n satisfies the properties of which we claim that they determine Λ uniquely. On the other hand, the extension of Λ_{i-1} to a triangulation of C_i is uniquely determined if one does not introduce further rays: the triangulation of the part V of the boundary of C_{i-1} that is visible from x_i has to coincide with the restriction of Λ_{i-1} to V.

(2) One easily checks that $\Lambda|F$ satisfies the conditions in (1) that characterize $\Lambda(x_{i_1}, \dots, x_{i_m})$.

(3) It is enough to check the claim for $i = n$. Then the only critical point for the conditions in (1) is whether $\Lambda(x_1, \dots, x_{n-2}, x_n, x_{n-1})$ restricts to C_{n-1}. But this is the case since C_{n-1} is a facet of C if $\dim C > \dim C_{n-1}$.

(4) follows by repeated application of (3). □

For the configuration of Fig. 1, claim 4 of Proposition 1 says that we could have started with the triangle spanned by the points 1, 2, 4 and then added the other points in the given order.

In the following we will assume that C is full dimensional: $\dim C = d = \dim \mathbb{R}^d$. Part (4) helps us to keep the data structure of lexicographic triangulations simple: right from the start we need only to work with the list of dimension d simplicial cones of Λ by searching $x_{i_1}, \dots, x_{i_d}$ first, choosing $\operatorname{cone}(x_{i_1}, \dots, x_{i_d})$ as the first d-dimensional simplicial cone and subsequently extending the list as prescribed by the definition of the lexicographic triangulation. In other words, we can assume that $x_1, \dots, x_d$ are linearly independent, and henceforth we will do so.

In order to extend the triangulation we must of course know which facets of C_{i-1} are visible from x_i. Recall that a cone C of dimension d in $\mathbb{R}^d$ has a unique irredundant representation as an intersection of linear halfspaces:

$$C = \bigcap_{H \in \mathcal{H}(C)} H^+,$$

where $\mathcal{H}(C)$ is a finite set of oriented hyperplanes and the orientation of the closed half spaces H^- and H^+ is chosen in such a way that $C \subset H^+$ for $H \in \mathcal{H}(C)$. For $H \in \mathcal{H}(C_{i-1})$ the facet $H \cap C_{i-1}$ is visible from x_i if and only if x_i lies in the open halfspace $H^< = H^- \setminus H$. When we refer to support hyperplane in the following we always mean those that appear in the irredundant decomposition of C since only they are important in the algorithmic context.

Hyperplanes are represented by linear forms $\lambda \in (\mathbb{R}^d)^*$, and we always work with the basis $e_1^*, \dots, e_d^*$ that is dual to the basis $e_1, \dots, e_d$ of unit vectors. For rational hyperplanes the linear form λ can always be chosen in such a way that it has integral coprime coefficients and satisfies $\lambda(x) \geq 0$ for $x \in C$. This choice determines λ uniquely. (If one identifies $e_1^*, \dots, e_d^*$ with $e_1, \dots, e_d$ via the standard scalar product, then λ is nothing but the primitive integral inner (with respect to C) normal vector of H.) For later use we define the *(lattice) height* of $x \in \mathbb{R}^d$ over H by

$$\mathrm{ht}_H(x) = |\lambda(x)|.$$

If $F = C \cap H$ is the facet of C cut out by H, we set $\mathrm{ht}_F(x) = \mathrm{ht}_H(x)$.

We can now describe the computation of the triangulation $\Lambda(x_1, \dots, x_n)$ and the support hyperplanes in a more formal way by Algorithm 1. For simplicity we will identify a simplicial cone σ with its generating set $\subset \{x_1, \dots, x_n\}$. It should be clear from the context what is meant. For a set $\mathcal{H}$ of hyperplanes we set

$$\mathcal{H}^*(x) = \{H \in \mathcal{H}, x \in H^*\} \qquad \text{where } * \in \{<, >, +, -\}.$$

Further we introduce the notation

$$\mathcal{H}^*(C, x) = \{H \in \mathcal{H}(C), x \in H^*\} \qquad \text{where } * \in \{<, >, +, -\}.$$

The representation of hyperplanes by linear forms makes it easy to detect the visible facets: a facet is visible from y if $\lambda(y) < 0$ for the linear from λ defining the hyperplane through the facet. As pointed out above, in Algorithm 1 and at several places below we may assume that the first d elements of $x_1, \dots, x_n$ are linearly independent. This can always be achieved by rearranging the order of the elements, or by a refined bookkeeping (as done by Normaliz).

Algorithm 1 Incremental building of cone, support hyperplanes and lexicographic triangulation.

Require: A generating set $x_1, \dots, x_n$ of a rational cone C of dimension d
Ensure: The support hyperplanes $\mathcal{H}$ of C and the triangulation $\Lambda(x_1, \dots, x_n)$

```
function LexTriangulation(x_1, ..., x_n)
    Δ ← {cone(x_1, ..., x_d)}
    H ← H(cone(x_1, ..., x_d))
    for i ← d + 1 to n do
        Δ ← ExtendTri(H, Δ, x_i)
        H ← FindNewHyp(H, x_1, ..., x_i)
    return (H, Δ)
```

Require: A set of hyperplanes $\mathcal{H}$, a triangulation Δ and a point y
Ensure: The union of Δ with the set of simplicial cones spanned by y and the facets δ of the $\sigma \in \Delta$ such that $\delta \subset H$ for some $H \in \mathcal{H}$ with $y \in H^<$

```
function ExtendTri(H, Δ, y)
    parallel for H ∈ H^<(y) do
        for σ ∈ Δ do
            if |σ ∩ H| = d − 1 then
                Δ ← Δ ∪ {cone(y, σ ∩ H)}
    return Δ
```

For its main data, Normaliz uses two types of data structures:

1. Lists and matrices of integer vectors. The vectors represent generators of cones, Hilbert basis elements, etc. in $\mathbb{R}^d$, or linear forms in $(\mathbb{R}^d)^*$.
2. Lists of subsets of the set $\{x_1, \dots, x_n\}$. Each subset stands for the subcone generated by its elements.

Sometimes more complicated data structures are needed. For example, it is useful in Algorithm 1 to store the incidence relation of generators and facets.

In the following discussion we set $C_j = \operatorname{cone}(x_1, \dots, x_j)$ as above. The support hyperplanes of the first simplicial cone C_d in line 3 are computed by essentially inverting the matrix of the generators $x_1, \dots, x_d$ (see equation (1) in Section 5). The function FindNewHyp computes $\mathcal{H}(C_i)$ from $\mathcal{H}(C_{i-1})$ by Fourier–Motzkin elimination. (It does nothing if $x_i \in C_{i-1}$.) Its Normaliz implementation has been described in great detail in Bruns and Ichim (2010); therefore we skip it here, but will come back to it below when we outline its combination with pyramid decomposition. The function ExtendTri does exactly what its name says: it extends the triangulation $\Lambda(x_1, \dots, x_{i-1})$ of C_{i-1} to the triangulation $\Lambda(x_1, \dots, x_i)$ of C_i (again doing nothing if $x_i \in C_{i-1}$).

One is tempted to improve ExtendTri by better bookkeeping and using extra information on triangulations of cones. We discuss our more or less fruitless attempts in the following remark.

Remark 2. (a) If one knows the restriction of $\Lambda(x_1, \dots, x_{i-1})$ to the facets of C_{i-1}, then $\Lambda(x_1, \dots, x_i)$ can be computed very fast. However, unless $i = n$, the facet triangulation must now be extended to the facets of C_i, and this step eats up the previous gain, as experiments have shown, at least for the relatively small triangulations to which ExtendTri is really applied after the pyramid decomposition described below.

(b) The test of the condition $|\sigma \cap H| = d - 1$ is positive if and only if $d - 1$ of the generators of σ lie in H. Its verification can be accelerated if one knows which facets of the d-dimensional cones in $\Lambda(x_1, \dots, x_{i-1})$ are already shared by another simplicial cone in $\Lambda(x_1, \dots, x_{i-1})$, and are therefore not available for the formation of a new simplicial cone. But the extra bookkeeping requires more time than is gained by its use.

(c) One refinement is used in our implementation, though its influence is almost unmeasurable. Each simplicial cone in $\Lambda(x_1, \dots, x_{i-1})$ has been added with a certain generator x_j, $j < i$. (The first cone is considered to be added with each of its generators.) It is not hard to see that only those simplicial cones that have been added with a generator $x_j \in H$ can satisfy the condition $|\sigma \cap H| = d - 1$, and this information is used to reduce the number of pairs (H, σ) to be tested.

(d) If $|H \cap \{x_1, \dots, x_{i-1}\}| = d - 1$, then $H \in \mathcal{H}^<(C_{i-1}, x_i)$ produces exactly one new simplicial cone of dimension d, namely $\operatorname{cone}(x_i, H \cap \{x_1, \dots, x_{i-1}\})$, and therefore the loop over σ can be suppressed.

The product $|\mathcal{H}^<(C_{i-1}, x_i)| \cdot |\Lambda(x_1, \dots, x_{i-1})|$ determines the complexity of ExtendTri. Even though the loop over H is parallelized (as indicated by **parallel for**), the time spent in ExtendTri can be very long. (The "exterior" loops in FindNewHyp are parallelized as well.) The second limiting factor for ExtendTri is memory: it is already difficult to store triangulations of size 10^8 and impossible for size $\geq 10^9$. Therefore the direct approach to lexicographic triangulations does not work for truly large cones.

Remark 3. The computation time for the Fourier–Motzkin elimination and the lexicographic triangulation often depends significantly on the order of the generators. If only the support hyperplanes must be computed, Normaliz orders the input vectors lexicographically. If also the triangulation must be computed, the input vectors are first sorted by their L_1-norm, or by degree if a grading is defined (see Section 5), and second lexicographically. The sorting by L_1-norm or degree helps to keep the determinants of the simplicial cones small (see Section 5). On the whole, we have reached good results with this order.

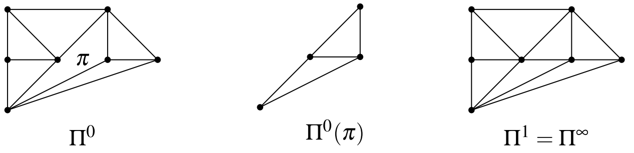

Fig. 3. Pyramid decomposition of the point configuration of Fig. 1.

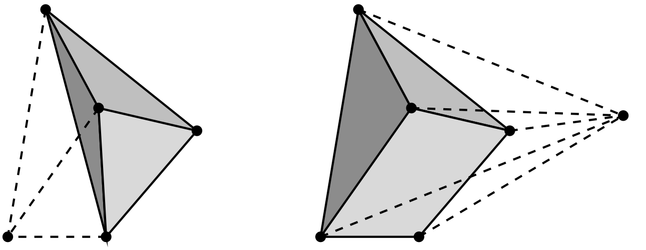

Fig. 4. Pyramid decomposition of Fig. 2.

Remark 4. Whenever possible, each parallel thread started in a Normaliz computation collects its computation results and returns them to the calling routine after its completion. In this way, the amount of synchronization between the threads is reduced to a minimum. For example, in ExtendTri, the new simplicial cones $\operatorname{cone}(y, \sigma \cap H)$ can be collected independently of each other: they are not directly added to the global list Δ in line 5, but are first stored in a list owned by the thread, and then spliced into Δ at the end of ExtendTri.

3.2. Pyramid decomposition

Now we present a radically different way to lexicographic triangulations via iterated *pyramid decompositions*. The cones that appear in this type of decomposition are called *pyramids* since their cross-section polytopes are pyramids in the usual sense, namely of type $\operatorname{conv}(F, x)$ where F is a facet and x is a vertex not contained in F.

Definition 5. The *pyramid decomposition* $\Pi(x_1, \dots, x_n)$ of $C = \operatorname{cone}(x_1, \dots, x_n)$ is recursively defined as follows: it is the trivial decomposition for $n = 0$, and

$$\Pi(x_1, \dots, x_n) = \Pi(x_1, \dots, x_{n-1}) \cup \{\operatorname{cone}(F, x_n) : \\ F \text{ a face of } \operatorname{cone}(x_1, \dots, x_{n-1}) \text{ visible from } x_n\}.$$

As already pointed out in the introduction, the pyramid decomposition is not a polyhedral subdivision in the strong sense: the intersection of two faces F and F' need not be a common face of F and F' (but is always a face of F or F'). See Figs. 3 and 4 for examples. Roughly speaking, one can say that in the pyramid decomposition forgets the potentially existing subdivision (or even triangulation) of the facets of $C(x_1, \dots, x_{n-1})$ that are visible from x_n. In order to subdivide (or even triangulate) the new pyramids it is enough to do the computations within each of them. This "localization" reduces the complexity tremendously.

In order to iterate the pyramid decomposition we set $\Pi^0(x_1, \dots, x_n) = \Pi(x_1, \dots, x_n)$, and

$$\Pi^k(x_1, \dots, x_n) = \bigcup_{P \in \Pi^{k-1}(x_1, \dots, x_n)} \{\Pi(x_i : x_i \in P)\} \qquad \text{for } k > 0.$$

We now assume that the first d vectors in the generating set of the top cone and each of its pyramids are linearly independent. Because of Proposition 1, claim 4, this assumption does not endanger the compatibility with lexicographic triangulation. Under this assumption the recursion defining Π^k cannot descend indefinitely, since the number of generators goes down with each recursion level. We denote the *total pyramid decomposition* by $\Pi^\infty(x_1,\dots,x_n)$.

Proposition 6. *One has* $\Pi^\infty(x_1,\dots,x_n) = \Pi^{n-d}(x_1,\dots,x_n) = \Lambda(x_1,\dots,x_n)$.

Proof. In the case $n = d$, the pyramid decomposition is obviously the face lattice of C, and therefore coincides with the lexicographic triangulation. For $n > d$ the first full dimensional pyramid reached is the simplicial cone $\operatorname{cone}(x_1,\dots,x_d)$. All the other pyramids have at most $n-1$ generators, and so we can use induction: For each $P \in \Pi(x_1,\dots,x_n)$ the total pyramid decomposition of P is the lexicographic triangulation $\Lambda(x_i : x_i \in P)$. According to Proposition 1(2) these triangulations match along the common boundaries of the pyramids, and therefore constitute a triangulation of C. It evidently satisfies the conditions in Proposition 1(1). $\square$

This leads to a recursive computation of $\Lambda(x_1,\dots,x_n)$ by the functions in Algorithm 2.

Algorithm 2 Incremental building of cone, support hyperplanes and lexicographic triangulation by total pyramid decomposition.

Require: A generating set $x_1,\dots,x_n$ of a rational cone C of dimension d
Ensure: The support hyperplanes $\mathcal{H}$ and of C and the triangulation $\Lambda(x_1,\dots,x_n)$
1: **function** TotalPyrDec($x_1,\dots,x_n$)
2: $\quad\Delta \leftarrow \{\operatorname{cone}(x_1,\dots,x_d)\}$
3: $\quad\mathcal{H} \leftarrow \mathcal{H}(\operatorname{cone}(x_1,\dots,x_d))$
4: $\quad$**for** $i \leftarrow d+1$ **to** n **do**
5: $\qquad(\mathcal{G},\Sigma) \leftarrow$ProcessPyrsRec($\mathcal{H}, x_1,\dots,x_i$)
6: $\qquad\mathcal{H} \leftarrow (\mathcal{H}\cup\mathcal{G}) \setminus \mathcal{H}^<(x_i)$
7: $\qquad\Delta \leftarrow \Delta\cup\Sigma$
8: $\quad$**return** $(\mathcal{H},\Delta)$

Require: A generating set $x_1,\dots,x_i$ of a rational cone C and the support hyperplanes $\mathcal{H} = \mathcal{H}(\operatorname{cone}(x_1,\dots,x_{i-1})$
Ensure: The support hyperplanes $\mathcal{H}(x_1,\dots,x_n) \setminus \mathcal{H}(x_1,\dots,x_{n-1})$ and the triangulation $\Lambda(x_1,\dots,x_n) \setminus \Lambda(x_1,\dots,x_{n-1})$
1: **function** ProcessPyrsRec($\mathcal{H}, x_1,\dots,x_n$)
2: $\quad\Delta \leftarrow \emptyset$
3: $\quad\mathcal{G} \leftarrow \emptyset$
4: $\quad$**parallel for** $H \in \mathcal{H}^<(x_n)$ **do**
5: $\qquad key \leftarrow \{x_n\} \cup (\{x_1,\dots,x_{n-1}\} \cap H)$
6: $\qquad(\mathcal{K},\Sigma) \leftarrow$TotalPyrDec($key$)
7: $\qquad\mathcal{G} \leftarrow \mathcal{G} \cup \{G \in \mathcal{K} : G \in \mathcal{H}(\operatorname{cone}(x_1,\dots,x_n))\}$
8: $\qquad\Delta \leftarrow \Delta\cup\Sigma$
9: $\quad$**return** $(\mathcal{G},\Delta)$

When called with the arguments $x_1,\dots,x_n$, the function TotalPyrDec builds $\Pi^\infty(x_1,\dots,x_n)$ (represented by its full dimensional members). As in Algorithm 1, the support hyperplanes of the simplicial cone C_d in line 3 are computed by the inversion of the generator matrix. All further support hyperplanes are given back to C_n by its "daughters" in line 6 where we also discard the support hyperplanes of C_{n-1} that have x_i in their negative half space.

The function ProcessPyrsRec manages the recursion that defines $\Pi^\infty(x_1,\dots,x_n)$. In its line 7 we must decide which support hyperplanes G of the daughter pyramid $\operatorname{cone}(key)$ are "new" support hyperplanes of the mother $C_n = \operatorname{cone}(x_1,\dots,x_n)$. We use the following criteria:

(i) $G \in \mathcal{H}(C_n) \iff x_j \in G^+$ for $j = 1,\dots,n-1$;
(ii) $G \notin \mathcal{H}(C_{n-1}) \iff x_j \in G^>$ for all $j = 1,\dots,i-1$ such that $x_j \notin key$.

One should note that pyramids effectively reduce the dimension: the complexity of $\operatorname{cone}(F, x_n)$ is completely determined by the facet F, which has dimension $d-1$.

While pyramid decomposition has primarily been developed for the computation of triangulations, it is also very useful in the computation of support hyperplanes. For Fourier–Motzkin elimination the critical complexity parameter is $|\mathcal{H}^<(C_{i-1}, x_i)| \cdot |\mathcal{H}^>(C_{i-1}, x_i)|$, and as in its use for triangulation, pyramid decomposition lets us replace a potentially very large product of the sizes of two "global" lists by a sum of small "local" products—the price to be paid is the computational waste invested for the support hyperplanes of the pyramids that are useless later on.

While being very memory efficient, total pyramid decomposition in the naïve implementation of Algorithm 2 is sometimes slower and sometimes faster than using Fourier–Motzkin elimination and building the lexicographic triangulation directly. The best solution is a hybrid algorithm that combines pyramid decomposition and lexicographic triangulation. It will be descried in the next section where we will also compare computation times and memory usage of pure lexicographic triangulation, pure pyramid decomposition and the hybrid algorithm. We compare computation times in Section 4.5.

4. The current implementation

4.1. The hybrid algorithm

Roughly speaking, the hybrid algorithm switches from Fourier–Motzkin elimination and lexicographic triangulation to pyramid decomposition for hyperplanes and triangulation when certain complexity parameters are exceeded. This strategy is realized by the function BuildCone of Algorithm 3.

Algorithm 3 Incremental building of cone, support hyperplanes and lexicographic triangulation by a hybrid algorithm.

Require: A generating set $x_1, \ldots, x_n$ of a rational cone C of dimension d. The top cone has an initially empty list Π of pyramids.
Ensure: The support hyperplanes $\mathcal{H}$ and of C and the triangulation $\Lambda(x_1, \ldots, x_n)$

1: **function** BuildCone($x_1, \ldots, x_n$)
2: $\quad\Delta \leftarrow \{\text{cone}(x_1, \ldots, x_d)\}$
3: $\quad\mathcal{H} \leftarrow \mathcal{H}(\text{cone}(x_1, \ldots, x_d))$
4: $\quad$**for** $i \leftarrow d+1$ **to** n **do**
5: $\quad\quad$**if** *MakePyramidsForHyps* **then**
6: $\quad\quad\quad(\mathcal{G}, \Sigma) \leftarrow$ProcessPyrsRec($\mathcal{H}, x_1, \ldots, x_i$)
7: $\quad\quad\quad\mathcal{H} \leftarrow (\mathcal{H} \cup \mathcal{G}) \setminus \mathcal{H}^<(x_i)$
8: $\quad\quad\quad\Delta \leftarrow \Delta \cup \Sigma$
9: $\quad\quad$**else**
10: $\quad\quad\quad$**if** *MakePyramidsForTri* **then**
11: $\quad\quad\quad\quad$**for** $H \in \mathcal{H}^<(\mathcal{H}, x_i)$ **do**
12: $\quad\quad\quad\quad\quad key \leftarrow \{x_i\} \cup (\{x_1, \ldots, x_{i-1}\} \cap H)$
13: $\quad\quad\quad\quad\quad\Pi \leftarrow \Pi \cup \{key\}$
14: $\quad\quad\quad$**else**
15: $\quad\quad\quad\quad\Delta \leftarrow$ExtendTri($\mathcal{H}, \Delta, x_i$)
16: $\quad\quad\quad\mathcal{H} \leftarrow$FindNewHyp($\mathcal{H}, x_1, \ldots, x_i$)
17: $\quad$**if** *TopCone* **then**
18: $\quad\quad$**parallel for** $P \in \Pi$ **do**
19: $\quad\quad\quad$buildCone(P)
20: $\quad\quad\quad\Pi \leftarrow \Pi \setminus \{P\}$
21: $\quad$**return** $(\mathcal{H}, \Delta)$

The boolean *MakePyramidsForHyps* (line 5) is determined by a single condition:

> it is set to *true* if the complexity parameter $|\mathcal{H}^<(C_{i-1}, x_i)| \cdot |\mathcal{H}^>(C_{i-1}, x_i)|$ exceeds a threshold, and to *false* otherwise.

As the name *MakePyramidsForHyps* indicates, the computation of support hyperplanes is transferred to the pyramids over the hyperplanes $\mathcal{H}^<(x_i)$ if the complexity parameter is exceeded. Pyramids created for the computation of support hyperplanes must be treated very carefully since the mother cone must wait for the computation of their support hyperplanes. We come back to this point below.

The *MakePyramidsForTri* (line 10) combines three conditions:

1. while set to *false* initially, it remains *true* once the switch to pyramids has been done in line 5 or line 10;
2. it is set *true* if the complexity parameter $|\mathcal{H}^{<}(C_{i-1}, x_i)| \cdot |\Delta|$ exceeds a threshold;
3. it is set *true* if the memory protection threshold is exceeded.

The last point needs to be explained. BuildCone is not only called for the processing of the top cone C, but also for the parallelized processing of pyramids. Since each of the "parallel" pyramids produces simplicial cones, the buffer in which the simplicial cones are collected for evaluation, may be severely overrun without condition (3), especially if $|\mathcal{H}^{<}(x_i)|$ is small, and therefore condition (2) is reached only for large $|\Lambda(x_1, \dots, x_{i-1})|$.

Pyramids that are created for triangulation can simply be stored since their triangulation is not needed for the continuation of the pyramid decomposition. Line 13 of BuildCone therefore adds them to the pyramid list Π which is part of the data of the top cone. The stored pyramids are evaluated after the top cone has been completely built (lines 17–20). It is a crucial aspect of pyramid decomposition that the loop in lines 18–20 is parallelized: the evaluation of a pyramid is a completely independent computation.

In the triangulation of the stored pyramids, new daughter pyramids may be created and added to the list. However, the number of pyramids is bounded by $|\Lambda(x_1, \dots, x_n)|$. At its termination, BuildCone returns the support hyperplanes of the top cone and the lexicographic triangulation $\Lambda(x_1, \dots, x_n)$.

Algorithm 3 is only a structural model of the actual implementation. Some of its technical details will be described below.

4.2. Pyramids for support hyperplanes

Pyramids that have been created because of the complexity of Fourier–Motzkin elimination are treated by the function ProcessPyrsRec. The Rec in its name indicates that the computation of the mother cone must wait for the completion of the daughter pyramid, at least for its support hyperplanes.

Algorithm 4 Processing of pyramids towards support hyperplanes and triangulation of mother cone.

Require: A generating set $x_1, \dots, x_i$ of a rational cone C and the support hyperplanes $\mathcal{H} = \mathcal{H}(\text{cone}(x_1, \dots, x_{i-1}))$
Ensure: The support hyperplanes $\mathcal{H}(x_1, \dots, x_i) \setminus \mathcal{H}(x_1, \dots, x_{i-1})$ and part of the triangulation $\Lambda(x_1, \dots, x_i) \setminus \Lambda(x_1, \dots, x_{i-1})$

function ProcessPyrsRec($\mathcal{H}, x_1, \dots, x_i$)
  $\Delta \leftarrow \emptyset$
  $\mathcal{G} \leftarrow \emptyset$
  parallel for $H \in \mathcal{H}^{<}(x_i)$ **do**
    key $\leftarrow \{x_i\} \cup (\{x_1, \dots, x_{i-1}\} \cap H)$
    if *Small* **then**
      $(\mathcal{K}, \Sigma) \leftarrow$ BuildCone(*key*)
      $\mathcal{G} \leftarrow \mathcal{G} \cup \{G \in \mathcal{K} : G \in \mathcal{H}(\text{cone}(x_1, \dots, x_i))\}$
      $\Delta \leftarrow \Delta \cup \Sigma$
    else
      $\mathcal{G} \leftarrow \mathcal{G} \cup$ MatchWitPosHyps($H, \mathcal{H}, x_1 \dots, x_i$)
      $\Pi \leftarrow \Pi \cup \{key\}$
  return $(\mathcal{G}, \Delta)$

The function is similar to the function ProcessPyrsRec in Algorithm 2, except that we now distinguish between "small" and "large" pyramids. Small pyramids are treated recursively as in the total pyramid decomposition, namely by applying BuildCone to them. The treatment of large pyramids differs in two ways:

1. the triangulation of the pyramid is deferred;

Table 1
Numerical data of test examples.

Input	edim	Rank	#ext	#supp	# triangulation
`CondPar`	24	24	234	27	1,344,671
`5x5`	25	15	1940	25	14,615,011
`lo6`	16	16	720	910	5,796,124,824
`cyclo60`	17	17	60	656,100	11,741,300
`A443`	40	30	48	4948	2,654,272
`A543`	47	36	60	29,387	102,538,890
`A553`	55	43	75	306,955	9,248,466,183

2. the Fourier–Motzkin step MatchWitPosHyps is used to find the support hyperplanes of the mother cone that originate from H.

The criterion for *small* is based on a comparison of the expected computation times for (i) building the pyramid over H and (ii) the Fourier–Motzkin step in which H is "matched" with the hyperplanes $G \in \mathcal{H}^{>}(x_i)$; see Bruns and Ichim (2010). This refinement was the last step added to the processing of pyramids. It is irrelevant in sequential computations, but large pyramids previously had the tendency to significantly delay the completion of the parallelized loop in line 4.

4.3. Interruption strategy

Normaliz keeps all data in RAM. Therefore it is necessary to control the size of the lists that contain simplicial cones and pyramids. This is achieved by a strategy that interrupts the production of pyramids and simplicial cones at suitable points as soon as the lists sizes have exceeded a preset value. The choice of the interruption points must take into consideration that Normaliz avoids nested parallelization for efficiency. (This is the default choice of OpenMP.)

As soon as BuildCone switches to pyramids, the triangulation $\Lambda(x_1, \ldots, x_{i-1})$ is no longer needed for further extension. Therefore it is shipped to the evaluation buffer. The simplicial cones are evaluated and the buffer is emptied whenever it has exceeded its preset size and program flow allows its parallelized evaluation.

The strategy for the evaluation of pyramids is similar, but it takes into account the recursive nature of the pyramid decomposition. The pyramid list is actually split into levels, and pyramids of level i produce subpyramids of level $i+1$. If the number of level $i+1$ pyramids becomes too large, the production at level i is interrupted in favor of the processing of the level $i+1$ pyramids.

4.4. Partial triangulation

The idea of pyramid decomposition was born when the authors observed that the computation of Hilbert bases in principle does not need a full triangulation of C. If a simplicial cone σ cannot contribute new candidates for the Hilbert basis of C, it need not be evaluated, and if a pyramid consists only of such simplicial cones, it need not be triangulated at all. This is the case if $\mathrm{ht}_H(x_i) = 1$.

The resulting strategy has sometimes striking results and was already described in Bruns et al. (2011).

4.5. Computation times

Section 6 contains extensive data on the performance of Normaliz. The computation times listed there include the evaluation of the simplicial cones for Hilbert bases and Hilbert series using the hybrid algorithm.

Here we want to compare lexicographic triangulation/Fourier–Motzkin elimination, pure pyramid decomposition and the hybrid algorithm in the computation of triangulations and support hyperplanes and triangulations, excluding any evaluation. (Normaliz can be restricted to these tasks.) The sources of the test input files of Table 1 are listed in Section 6 where we give computation times for a large number of examples. The times reported in this section were taken on a SUN xFire 4450 with 4 Intel Xeon X7460 (a total of 24 cores running at 2.66 GHz) and 128 GB RAM.

Table 2
Triangulation.

Input	Threads	Lex triang	Total pyr dec	Hybrid
CondPar	1	15.8 s	2:06 m	3.0 s
	20	10.5 s	1:20 m	2.8 s
A443	1	8:32 m	4:37 m	12.0 s
	20	39.7 s	1:23 m	5.4 s
A543	1	–	–	8:06 m
	20	4:53 h	–	44.0 s
A553	20	–	–	1:22 h
lo6	1	–	–	3:19 h
	20	–	–	27:11 m
5x5	1	45:39 m	11:52 m	1:25 m
	20	5:16 m	5:18 m	18.5 s
cyclo60	1	–	12:35 m	5:10 m
	20	5:45 h	3:14 m	1:21 m

Table 3
Support hyperplanes.

Input	Threads	Fourier–Motzkin	Hybrid
lo6	1	39.3 s	44.2 s
	20	4.5 s	4.1 s
cyclo60	1	–	2:52 m
	20	1:23 h	44.3 s
A553	1	2:48 h	11:47 m
	20	10:29 m	1:08 m

As Table 2 shows, the hybrid algorithm is far superior to lexicographic triangulation as soon as the triangulations are large enough to have pyramids really built. Moreover, the need of storing the whole triangulation in RAM limits the applicability of lexicographic triangulation to sizes of $\approx 10^8$: A543 needs already 21 GB of RAM, and therefore lo6 and A553 cannot be computed by it, even if one is willing to wait for a very long time. The RAM needed by the hybrid algorithm is essentially determined by the fact that Normaliz collects $2.5 \cdot 10^6$ simplicial cones for parallelized evaluation, and is typically between 500 MB and 1 GB.

When the number of support hyperplanes is very large relative to the triangulation size, as for cyclo60, total pyramid decomposition is much better than lexicographic triangulation and can compete with the hybrid algorithm. This is not surprising since the pyramids built by the hybrid algorithm are close to being simplicial. The efficiency of parallelization depends on the use of ProcessPyrsRec: the dependence of the mother on the daughters limits the gain by parallelization.

For the computation of support hyperplanes the hybrid algorithm shows its power only for cones with truly large numbers of support hyperplanes, like A553 or cyclo60. The third example lo6 in Table 3 is a borderline case in which Pure Fourier–Motzkin elimination and the hybrid algorithm behave almost identically. The computation times of total pyramid decomposition are almost identical with those for triangulation since the only difference is that the simplicial cones must be stored.

5. Evaluation of simplicial cones

The fast computation of triangulations via pyramid decomposition must be accompanied by an efficient evaluation of the simplicial cones in the triangulation Δ, which, after the introduction of the pyramid decomposition, is almost always the more time consuming step. Like the processing of pyramids, the evaluation of simplicial cones is parallelized in Normaliz.

Let σ be a simplicial cone generated by the linearly independent vectors $v_1, \dots, v_d$. The evaluation is based on the *generator matrix* G_σ whose *rows* are $v_1, \dots, v_d$. Before we outline the evaluation procedure, let us substantiate the remark made in Section 3 that finding the support hyperplanes amounts to the inversion of G_σ. Let H_i be the support hyperplane of σ opposite to v_i, given by the linear form $\lambda_i = a_{1i}e_1^* + \cdots + a_{di}e_d^*$ with coprime integer coefficients a_j. Then

$$\lambda_i(v_k) = \sum_{j=1}^{d} v_{kj}a_{ji} = \begin{cases} \mathrm{ht}_{H_i}(v_i), & k = i, \\ 0, & k \neq i. \end{cases} \tag{1}$$

Thus the matrix (a_{ij}) is G_σ^{-1} up to scaling of its columns. Usually the inverse is computed only for the first simplicial cone in every pyramid since its support hyperplanes are really needed. But matrix inversion is rather expensive, and Normaliz goes to great pains to avoid it.

Normaliz computes sets of vectors, primarily Hilbert bases, but also measures, for example the volumes of rational polytopes. A polytope P arises from a cone C by cutting C with a hyperplane, and for Normaliz such hyperplanes are defined by gradings: a *grading* is a linear form $\deg : \mathbb{Z}^d \to \mathbb{Z}$ (extended naturally to $\mathbb{R}^d$) with the following properties: (i) $\deg(x) > 0$ for all $x \in C$, $x \neq 0$, and (ii) $\deg(\mathbb{Z}^d) = \mathbb{Z}$. The first condition guarantees that the intersection $P = C \cap A_1$ for the affine hyperplane

$$A_1 = \{x \in \mathbb{R}^d : \deg(x) = 1\}$$

is compact, and therefore a rational polytope. The second condition is harmless for integral linear forms since it can be achieved by extracting the greatest common divisor of the coefficients of deg with respect to the dual basis.

The grading deg can be specified explicitly by the user or chosen implicitly by Normaliz. The implicit choice makes only sense if there is a natural grading, namely one under which the extreme integral generators of C all have the same degree. (If it exists, it is of course uniquely determined.)

At present, Normaliz evaluates the simplicial cones σ in the triangulation of C for the computation of the following data:

(HB) the Hilbert basis of C,
(LP) the lattice points in the rational polytope $P = C \cap A_1$,
(Vol) the normalized volume $\mathrm{vol}(P)$ of the rational polytope P (also called the *multiplicity* of C),
(HF) the *Hilbert* or *Ehrhart function* $H(C,k) = |kP \cap \mathbb{Z}^d|$, $k \in \mathbb{Z}_+$.

5.1. Volume computation

Task (Vol) is the easiest, and Normaliz computes $\mathrm{vol}(P)$ by summing the volumes $\mathrm{vol}(\sigma \cap A_1)$ where σ runs over the simplicial cones in the triangulation. With the notation introduced above, one has

$$\mathrm{vol}(\sigma \cap A_1) = \frac{|\det(G_\sigma)|}{\deg(v_1)\cdots\deg(v_d)}.$$

For the justification of this formula note that the simplex $\sigma \cap A_1$ is spanned by the vectors $v_i/\deg(v_i)$, $i = 1,\dots,d$, and that the vertex 0 of the d-simplex $\delta = \mathrm{conv}(0, \sigma \cap A_1)$ has (lattice) height 1 over the opposite facet $\sigma \cap A_1$ of δ so that $\mathrm{vol}(\sigma \cap A_1) = \mathrm{vol}(\delta)$.

In pure volume computations Normaliz (since version 2.9) utilizes the following proposition that often reduces the number of determinant calculations significantly.

Proposition 7. *Let σ and τ be simplicial cones sharing a facet F. Let $v_1,\dots,v_d$ span τ and let v_d be opposite of F. If $|\det(G_\sigma)| = 1$, then $|\det(G_\tau)| = \mathrm{ht}_F(v_d)$.*

Proof. The proposition is a special case of Bruns and Gubeladze (2009, Prop. 3.9), but is also easily seen directly. Suppose that w_d is the generator of σ opposite to F. Then $G_\sigma = \{v_1,\dots,v_{d-1},w_d\}$, and $|\det G_\sigma| = 1$ by hypothesis. Therefore $v_1,\dots,v_{d-1},w_d$ span $\mathbb{Z}^d$. With respect to this basis, the matrix of coordinates of $v_1,\dots,v_d$ is lower trigonal with 1 on the diagonal, except in the lower right corner where we find $-\mathrm{ht}_F(v_d)$. □

Every new simplicial cone τ found by ExtendTri is taken piggyback by an already known "partner" σ sharing a facet F with τ. Therefore Normaliz records $|\det G_\sigma|$ with σ, and if $|\det G_\sigma| = 1$

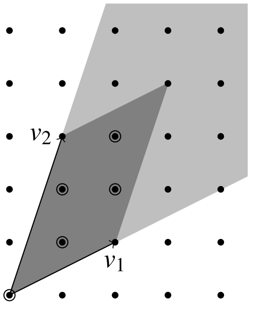

Fig. 5. Lattice points in the fundamental domain.

there is no need to compute $|\det(G_\tau)|$ since the height of the "new" generator v_d over F is known. Remark 10(b) contains some numerical data illuminating the efficiency of this strategy that we call *exploitation of unimodularity*. One should note that it is inevitable to compute $|\det(G_\sigma)|$ for the first simplicial cone in every pyramid.

5.2. Lattice points in the fundamental domain

The sublattice U_σ spanned by $v_1, \dots, v_d$ acts on $\mathbb{R}^d$ by translation. The semi-open parallelotope

$$\mathrm{par}(v_1, \dots, v_d) = \{q_1 v_1 + \cdots + q_d v_d : 0 \le q_i < 1\}$$

is a fundamental domain for this action; see Fig. 5. In particular,

$$E = E_\sigma = \mathrm{par}(v_1, \dots, v_d) \cap \mathbb{Z}^d$$

is a set of representatives of the group $\mathbb{Z}^d/U_\sigma$. The remaining tasks depend crucially on the set E.

For the efficiency of the evaluation it is important to generate E as fast as possible. One finds E in two steps:

(Rep) find a representative of every residue class of the vectors in $\mathbb{Z}^d$, and
(Mod) reduce its coefficients with respect to the $\mathbb{Q}$-basis $v_1, \dots, v_d$ modulo 1.

The first idea for (Rep) that comes to mind (and used in the first version of Normaliz) is to decompose $\mathbb{Z}^d/U_\sigma$ into a direct sum of cyclic subgroups $\mathbb{Z}\overline{u}_i$, $i = 1, \dots, d$ where $u_1, \dots, u_d$ is a $\mathbb{Z}$-basis of $\mathbb{Z}^d$ and $^-$ denotes the residue class modulo U_σ. The elementary divisor theorem guarantees the existence of such a decomposition, and finding it amounts to a diagonalization of G_σ over $\mathbb{Z}$. But diagonalization is even more expensive than matrix inversion, and therefore it is very helpful that a filtration of $\mathbb{Z}^d/U_\sigma$ with cyclic quotients is sufficient. Such a filtration can be based on trigonalization:

Proposition 8. *With the notation introduced, let $e_1, \dots, e_d$ denote the unit vectors in $\mathbb{Z}^d$ and let $X \in \mathrm{GL}(d, \mathbb{Z})$ such that XG_σ is an upper triangular matrix D with diagonal elements $a_1, \dots, a_d \ge 1$. Then the vectors*

$$b_1 e_1 + \cdots + b_d e_d, \qquad 0 \le b_i < a_i,\ i = 1, \dots, d, \tag{2}$$

represent the residue classes in $\mathbb{Z}^d/U_\sigma$.

Proof. Note that the rows of XG_σ are a $\mathbb{Z}$-basis of U_σ. Since $|\mathbb{Z}^d/U_\sigma| = |\det G_\sigma| = a_1 \cdots a_d$, it is enough to show that the elements listed represent pairwise different residue classes. Let p be the largest index such that $a_p > 1$. Note that a_p is the order of the cyclic group $\mathbb{Z}\overline{e}_p$, and that we obtain a $\mathbb{Z}$-basis of $U'_\sigma = U_\sigma + \mathbb{Z}e_p$ if we replace the p-th row of XG_σ by e_p. If two vectors $b_1 e_1 + \cdots + b_p e_p$ and $b'_1 e_1 + \cdots + b'_p e_p$ in our list represent the same residue class modulo U_σ, then they are even more so modulo U'_σ. It follows that $b_i = b'_i$ for $i = 1, \dots, p-1$, and taking the difference of the two vectors, we conclude that $b_p = b'_p$ as well. $\square$

The first linear algebra step that comes up is therefore the trigonalization

$$XG_\sigma = D. \tag{3}$$

Let G_σ^{tr} be the transpose of G_σ. For (Mod) it is essentially enough to reduce those e_i modulo 1 that appear with a coefficient > 0 in (2), and thus we must solve the simultaneous linear systems

$$G_\sigma^{\mathrm{tr}} x_i = e_i, \qquad a_i > 1, \tag{4}$$

where we consider x_i and e_i as column vectors. In a crude approach one would simply invert the matrix G_σ^{tr} (or G_σ), but in general the number of i such that $a_i > 1$ is small compared to d (especially if d is large), and it is much better to solve a linear system with the specific multiple right hand side given by (4). The linear algebra is of course done over $\mathbb{Z}$, using $a_1 \cdots a_d$ as a common denominator. Then Normaliz tries to produce the residue classes and to reduce them modulo 1 (or, over $\mathbb{Z}$, modulo $a_1 \cdots a_d$) as efficiently as possible.

For task (LP) one extracts the vectors of degree 1 from E, and the degree 1 vectors collected from all σ from the set of lattice points in $P = C \cap A_1$. For (HB) one first reduces the elements of $E \cup \{v_1, \dots, v_d\}$ to a Hilbert basis of σ, collects these and then applies “global” reduction in C. This procedure has been described in Bruns and Ichim (2010).

5.3. Hilbert series and Stanley decomposition

The mathematically most interesting task is (HF). The Hilbert series is defined by

$$H_C(t) = \sum_{x \in C \cap \mathbb{Z}^d} t^{\deg x} = \sum_{k=0}^{\infty} H(C,k) t^k, \qquad H(C,k) = |\{x \in C : \deg x = k\}|.$$

It is well-known that $H_C(t)$ is the power series expansion of a rational function in t.

For a simplicial cone σ spanned by $v_1, \dots, v_d$ as above one has

$$H_\sigma(t) = \frac{h_0 + h_1 t + \cdots + h_s t^s}{(1-t^{g_1}) \cdots (1-t^{g_d})}, \qquad g_i = \deg v_i, \qquad h_j = |\{x \in E_\sigma : \deg x = j\}|.$$

This follows immediately from the disjoint decomposition

$$\sigma \cap \mathbb{Z}^d = \bigcup_{x \in E_\sigma} x + M_\sigma \tag{5}$$

where M_σ is the (free) monoid generated by $v_1, \dots, v_d$.

However, one cannot compute $H_C(t)$ by simply summing these functions over $\sigma \in \Delta$ since points in the intersections of the simplicial cones σ would be counted several times. Fortunately, the intricate inclusion–exclusion problem can be avoided since there exist *disjoint* decompositions

$$C = \bigcup_{\sigma \in \Delta} \sigma \setminus S_\sigma \tag{6}$$

of C by semi-open simplicial cones $\sigma \setminus S_\sigma$ where S_σ is the union of some facets (and not just arbitrary faces!) of σ. Following Kleinschmidt and Smilansky (1991) we call a decomposition of type (6) a *facet cover* of Δ. (The name is motivated by the fact that each lower dimensional face of Δ is contained in exactly one of the “surviving” facets.)

Before we discuss the existence and computation of a facet cover, let us first derive a representation of the Hilbert series based on it. It generalizes the h-vector formula of McMullen-Walkup (1998, 5.1.14).

Let $\sigma \in \Delta$ and $x \in E_\sigma$, $x = \sum q_i v_i$. Then we define $\varepsilon(x)$ as the sum of all v_i such that (i) $q_i = 0$ and (ii) the facet opposite to v_i belongs to S. Since $(x + M_\sigma) \setminus S = \varepsilon(x) + x + M_\sigma$, we obtain the *Stanley decomposition*

$$C \cap \mathbb{Z}^d = \bigcup_{\sigma \in \Delta} M_\sigma \setminus S_\sigma = \bigcup_{\sigma \in \Delta} \bigcup_{x \in E_\sigma} x + \varepsilon(x) + M_\sigma, \tag{7}$$

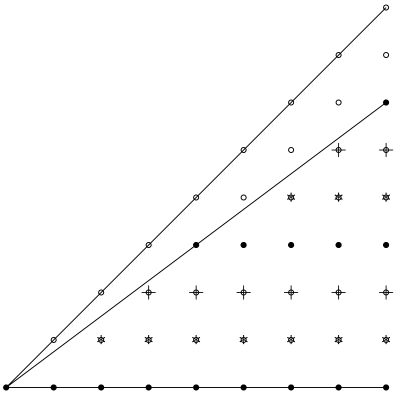

Fig. 6. A Stanley decomposition.

of $C \cap \mathbb{Z}^d$ into disjoint subsets. A Stanley decomposition into 4 components is illustrated by Fig. 6 in which lattice points in different components are marked differently.

The series $H_{\sigma\setminus S_\sigma}(t)$ is as easy to compute as $H_\sigma(t)$:

$$\begin{aligned} H_{\sigma\setminus S_\sigma}(t) &= \sum_{y\in M_\sigma\setminus S_\sigma} t^{\deg y} = \sum_{x\in E_\sigma}\sum_{z\in M_\sigma} t^{\deg x+\varepsilon(x)+z} = \sum_{x\in E_\sigma} t^{\deg x+\varepsilon(x)} H_\sigma(t) \\ &= \frac{\sum_{x\in E_\sigma} t^{\deg \varepsilon(x)+\deg x}}{(1-t^{g_1})\cdots(1-t^{g_d})}. \end{aligned} \tag{8}$$

It only remains to sum the series $H_{\sigma\setminus S_\sigma}(t)$ over the triangulation Δ.

The existence of a facet cover and (consequently) a Stanley decomposition of C was shown by Stanley (1982, Theorem 5.2) using the existence of a line shelling of C (proved by Bruggesser and Mani). Instead of finding a shelling order for the lexicographic triangulation (which is in principle possible), Normaliz 2.0–2.5 used a line shelling for the decomposition, as discussed in Bruns and Ichim (2010).

This approach works well for cones of moderate size, but has a major drawback: finding the sets S requires searching over the shelling order, and in particular the whole triangulation must be stored. We learned a much simpler principle for the disjoint decomposition (already implemented in Normaliz 2.7) from Köppe and Verdoolaege (2008). It was previously used by Kleinschmidt and Smilansky (1991) (also see Stanley, 1996, p. 85). As a consequence, each simplicial cone in the triangulation can be treated in complete independence from the others, and can therefore be discarded once it has been evaluated (unless the user insists on seeing the triangulation):

Lemma 9. *Let O_C be a vector in the interior of C such that O_C is not contained in a support hyperplane of any simplicial σ in a triangulation of C. For σ choose S_σ as the union of the support hyperplanes $\mathcal{H}^<(\sigma, O_C)$. Then the semi-open simplicial cones $\sigma \setminus S_\sigma$ form a disjoint decomposition of C.*

See Köppe and Verdoolaege (2008) for a proof. Fig. 7 shows a facet cover resulting from Lemma 9.

It is of course not possible to choose an *order vector* O_C that avoids all hyperplanes in advance, but this is not a real problem. Normaliz chooses O_C in the interior of the first simplicial cone, and works with a lexicographic infinitesimal perturbation O'_C. (This trick is known as "simulation of simplicity" in computational geometry; see Edelsbrunner, 1987.) If $O_C \in H^<$ (or $O_C \in H^>$), then $O'_C \in H^<$ (or $O'_C \in H^>$). In the critical case $O_C \in H$, we take the linear form λ representing H and look up its coordinates in the dual basis $e_1^*, \dots, e_d^*$. If the first nonzero coordinate is negative, then $O'_C \in H^<$, and else $O'_C \in H^>$.

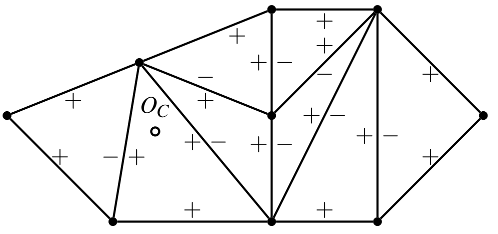

Fig. 7. Using the order vector.

At first it seems that one must compute the support hyperplanes of σ in order to apply Lemma 9. However, it is much better to solve the system

$$G_\sigma^{\mathrm{tr}} I^\sigma = O_C. \tag{9}$$

The solution I^σ is called the *indicator* of σ. One has $O_C \in H^<$ (or $O_C \in H^>$) if $I_i^\sigma < 0$ (or $I_i^\sigma > 0$) for the generator v_i opposite to H (λ vanishes on H). Let us call σ *generic* if all entries of I^σ are nonzero.

If $I_i^\sigma = 0$—this happens rarely, and very rarely for more than one index i—then we are forced to compute the linear form representing the support hyperplane opposite of v_i. In view of (1) this amounts to solving the systems

$$G_\sigma x = e_i, \qquad I_i^\sigma = 0, \tag{10}$$

simultaneously for the lexicographic decision.

If σ is unimodular, in other words, if $|\det G_\sigma| = 1$, then the only system to be solved is (9), provided that σ is generic. Normaliz tries to take advantage of this fact by guessing whether σ is unimodular, testing two necessary conditions:

(PU1) Every σ (except the first) is inserted into the triangulation with a certain generator x_i. Let H be the facet of σ opposite to x_i. If $\mathrm{ht}_H(x_i) > 1$, then σ is nonunimodular. (The number $\mathrm{ht}_H(x_i)$ has been computed in the course of the triangulation.)

(PU2) If $\gcd(\deg v_1, \dots, \deg v_d) > 1$, then σ is not unimodular.

If σ passes both tests, we call it *potentially unimodular*. (Data on the efficiency of this test will be given in Remark 10(a).)

After these preparations we can describe the order in which Normaliz treats the trigonalization (3) and the linear systems (4), (9) and (10):

(L1) If σ is potentially unimodular, then (9) is solved first. It can now be decided whether σ is indeed unimodular.

(L2) If σ is not unimodular, then the trigonalization (3) is carried out next. In the potentially unimodular, but nongeneric case, the trigonalization is part of the solution of (10) (with multiple right hand side).

(L3) In the nonunimodular case, we now solve the system (4) (with multiple right hand side).

(L4) If σ is not potentially unimodular and not generic, it remains to solve the system (10) (with multiple right hand side).

As the reader may check, it is never necessary to perform all 4 steps. In the unimodular case, (L1) must be done, and additionally (L2) if σ is nongeneric. If σ is not even potentially unimodular, (L2) and (L3) must be done, and additionally (L4) if it is nongeneric. In the potentially unimodular, but nonunimodular case, (L1), (L2) and (L3) must be carried out.

5.4. Presentation of Hilbert series

We conclude this section with a brief discussion of the computation and the representation of the Hilbert series by Normaliz. The reader can find the necessary background in Bruns and Gubeladze (2009, Chapter 6).

Summing the Hilbert series (8) is very simple if they all have the same denominator, for example in the case in which the generators of C (or at least the extreme integral generators) have degree 1. For efficiency, Normaliz first forms "denominator classes" in which the Hilbert series with the same denominator are accumulated. At the end, the class sums are added over a common denominator that is extended whenever necessary. This yields a "raw" form of the Hilbert series of type

$$H_C(t) = \frac{R(t)}{(1-t^{s_1})\cdots(1-t^{s_r})}, \qquad R(t) \in \mathbb{Z}[t], \tag{11}$$

whose denominator in general has $> d$ factors.

In order to find a presentation with d factors, Normaliz proceeds as follows. First it reduces the fraction to lowest terms by factoring the denominator of (11) into a product of cyclotomic polynomials:

$$H_C(t) = \frac{Z(t)}{\zeta_{z_1}\cdots\zeta_{z_w}}, \qquad Z(t) \in \mathbb{Z}[t], \quad \zeta_{z_j} \nmid Z(t), \tag{12}$$

which is of course the most economical way for representing $H_C(t)$ (as a single fraction). The orders and the multiplicities of the cyclotomic polynomials can easily be bounded since all denominators in (8) divide $(1-t^\ell)^d$ where ℓ is the least common multiple of the degrees $\deg x_i$. So we can find a representation

$$H_C(t) = \frac{F(t)}{(1-t^{e_1})\cdots(1-t^{e_d})}, \qquad F(t) \in \mathbb{Z}[t], \tag{13}$$

in which e_d is the least common multiple of the orders of the cyclotomic polynomials that appear in (12), e_{d-1} is the least common multiple of the orders that have multiplicity ≥ 2, etc. Normaliz produces the presentation (13) whenever the degree of the numerator remains of reasonable size.

It is well-known that the Hilbert function itself is a quasipolynomial:

$$H(C,k) = q_0(k) + q_1(k)k + \cdots + q_{d-1}(k)k^{d-1}, \qquad k \geq 0, \tag{14}$$

where the coefficients $q_j(k) \in \mathbb{Q}$ are periodic functions of k whose common period is the least common multiple of the orders of the cyclotomic polynomials in the denominator of (12). Normaliz computes the quasipolynomial, with the proviso that its period is not too large. It is not hard to see that the periods of the individual coefficients are related to the representation (13) in the following way: e_k is the common period of the coefficients $q_{d-1},\dots,q_{d-k}$. The leading coefficient q_{d-1} is actually constant (hence $e_1 = 1$), and related to the multiplicity by the equation

$$q_{d-1} = \frac{\operatorname{vol}(P)}{(d-1)!}. \tag{15}$$

Since q_{d-1} and $\operatorname{vol}(P)$ are computed completely independently from each other, equation (15) can be regarded as a test of correctness for both numbers.

The choice (13) for $H_C(t)$ is motivated by the desire to find a standardized representation whose denominator conveys useful information. The reader should note that this form is not always the expected one. For example, for $C = \mathbb{R}^2_+$ with $\deg(e_1) = 2$ and $\deg(e_2) = 3$, the three representations (11)–(13) are

$$\frac{1}{(1-t^2)(1-t^3)} = \frac{1}{\zeta_1^2\zeta_2\zeta_3} = \frac{1-t+t^2}{(1-t)(1-t^6)}.$$

Actually, it is unclear what the most natural standardized representation of the Hilbert series as a fraction of two polynomials should look like, unless the denominator is $(1-t)^d$. Perhaps the most satisfactory representation should use a denominator $(1-t^{p_1})\cdots(1-t^{p_d})$ in which the exponents p_i are the degrees of a homogeneous system of parameters (for the monoid algebra $K[\mathbb{Z}^d \cap C]$ over an infinite field K). At present Normaliz cannot find such a representation (except the one with the trivial denominator $(1-t^\ell)^d$), but future versions may contain this functionality.

6. Computational results

In this section we want to document that the algorithmic approach described in the previous sections (and Bruns and Ichim, 2010) is very efficient and masters computations that appeared inaccessible some years ago. We compare Normaliz 3.0 to 4ti2, version 1.6.6 (4ti2 team, 2015), for Hilbert basis computations and to LattE integrale, version 1.7.3 (Baldoni et al., 2015), for Hilbert series.

Almost all computations were run on a Dell PowerEdge R910 with 4 Intel Xeon E7540 (a total of 24 cores running at 2 GHz), 128 GB of RAM and a hard disk of 500 GB. The remaining computations were run on a SUN xFire 4450 with a comparable configuration. In parallelized computations we have limited the number of threads used to 20. As the large examples below show, the parallelization scales efficiently. In Tables 5 and 6 serial execution is indicated by `1x` whereas `20x` indicates parallel execution with a maximum of 20 threads. Normaliz needs relatively little memory. Almost all Normaliz computations mentioned run stably with < 1 GB of RAM.

Normaliz is distributed as open source under the GPL. In addition to the source code, the distribution contains executables for the major platforms Linux, Mac and Windows.

6.1. Overview of the examples

We have chosen the following test candidates:

1. `CondPar`, `CEffPl` and `PlVsCut` come from social choice theory. `CondPar` represents the Condorcet paradox, `CEffPl` computes the Condorcet efficiency of plurality voting, and `PlVsCut` compares plurality voting to cutoff, all for 4 candidates. See Schürmann (2013) for more details.
2. `4x4`, `5x5` and `6x6` represent monoids of "magic squares": squares of size 4×4, 5×5 and 6×6 to be filled with nonnegative integers in such a way that all rows, columns and the two diagonals sum to the same "magic constant". They belong to the standard LattE distribution (Baldoni et al., 2015).
3. `bo5` and `lo6` belong to the area of statistical ranking; see Sturmfels and Welker (2012). `bo5` represents the boolean model for the symmetric group S_5 and `lo6` represents the linear order model for S_6.
4. `small` and `big` are test examples used in the development of Normaliz without further importance. `small` has already been discussed in Bruns and Ichim (2010).
5. `cyclo36`, `cyclo38`, `cyclo42` and `cyclo60` represent the cyclotomic monoids of orders 36, 38, 42 and 60. They are additively generated by the pairs $(\zeta, 1) \in \mathbb{C}\times\mathbb{Z}_+$ where ζ runs over the roots of unity of the given order. They have been discussed by Beck and Hoşten (2006).
6. `A443` and `A553` represent monoids defined by dimension 2 marginal distributions of dimension 3 contingency tables of sizes $4\times4\times3$ and $5\times5\times3$. They had been open cases in the classification of Ohsugi and Hibi (2006) and were finished in Bruns et al. (2011).
7. `cross10`, `cross15` and `cross20` are (the monoids defined by) the cross polytopes of dimensions 10, 15 and 20 contained in the LattE distribution (Baldoni et al., 2015).

The columns of Table 4 contain the values of characteristic numerical data of the test examples M, namely: edim is the embedding dimension, i.e., the rank of the lattice in which M is embedded by its definition, whereas rank is the rank of M. #ext is the number of the extreme rays of the cone $\mathbb{R}_+M$, and #supp the number of its support hyperplanes. #Hilb is the size of the Hilbert basis of M.

The last two columns list the number of simplicial cones in the triangulation and the number of components of the Stanley decomposition. These data are not invariants of M. However, if the triangulation uses only lattice points of a lattice polytope P (all examples starting from `bo5`), then the number of components of the Stanley decomposition is exactly the normalized volume of P.

Table 4
Numerical data of test examples.

Input	edim	Rank	#ext	#supp	#Hilb	# triangulation	# Stanley dec
CondPar	24	24	234	27	242	1,344,671	1,816,323
PlVsCut	24	24	1872	28	9621	257,744,341,008	2,282,604,742,033
CEffPl	24	24	3928	30	25,192	347,225,775,338	4,111,428,313, 448
4x4	16	8	20	16	20	48	48
5x5	25	15	1940	25	4828	14,615,011	21,210,526
6x6	36	24	97,548	36	522,347	–	–
bo5	31	27	120	235	120	20,853,141,970	20,853,141,970
lo6	16	16	720	910	720	5,796,124,824	5,801,113,080
small	6	6	190	32	34,591	4580	2,276,921
big	7	7	27	56	73,551	542	18,788,796
cyclo36	13	13	36	46,656	37	44,608	46,656
cyclo38	19	19	38	923,780	39	370,710	923,780
cyclo42	13	13	42	24,360	43	153,174	183,120
cyclo60	17	17	60	656,100	61	11,741,300	13,616,100
A443	40	30	48	4,948	48	2,654,272	2,654,320
A553	55	43	75	306,955	75	9,248,466,183	9,249,511,725
cross10	11	11	20	1024	21	512	1024
cross15	16	16	30	32,678	31	16,384	32,768
cross20	21	21	40	1,048,576	41	524,288	1,048,576

The open entries for 6x6 seem to be out of reach presently. The Hilbert series of 6x6 is certainly a challenge for the future development of Normaliz. Other challenges are lo7, the linear order polytope for S_7 and the first case of the cyclotomic monoids cyclo105 that is not covered by the theorems of Beck and Hoşten (2006). Whether cyclo105 will ever become computable, is quite unclear in view of its gigantic number of support hyperplanes. However, we are rather optimistic for lo7; the normality of the linear order polytope for S_7 is an open question.

6.2. Hilbert bases

Table 5 contains the computation times for the Hilbert bases of the test candidates. When comparing 4ti2 and Normaliz one should note that 4ti2 is not made for the input of cones by generators, but for the input via support hyperplanes (CondPar – 6x6). The same applies to the Normaliz dual mode -d. While Normaliz is somewhat faster even in serial execution, the times are of similar magnitude. It is certainly an advantage that its execution has been parallelized. When one runs Normaliz with the primary algorithm on such examples it first computes the extreme rays of the cone and uses them as generators.

Despite of the fact that several examples could not be expected to be computable with 4ti2, we tried. We stopped the computations when the time had exceeded 150 h (T) or the memory usage had exceeded 100 GB (R). However, one should note that A553 (and related examples) can be computed by “LattE for tea, too” (http://www.latte-4ti2.de), albeit with a very large computation time; see Bruns et al. (2011). This approach uses symmetries to reduce the amount of computations.

In Table 5 the option -d indicates the dual algorithm, and -N indicates the primal algorithm for Hilbert bases. The number n of threads is given by nx.

The examples CEffPl, PlVsCut, 5x5 and 6x6 are clear cases for the dual algorithm. However, it is sometimes difficult to decide whether the primary, triangulation based algorithm or the dual algorithm is faster. As small clearly shows, the dual algorithm behaves badly if the final Hilbert basis is large, even if the number of support hyperplanes is small.

The computation time of bo5 which is close to zero is quite surprising at first glance, but it has a simple explanation: the lexicographic triangulation defined by the generators in the input file is unimodular so that all pyramids have height 1, and the partial triangulation is empty.

The computation time for the Hilbert basis of cyclo38 is large compared to the time for the Hilbert series in Table 6. The reason is the large number of support hyperplanes together with a large number of candidates for the Hilbert basis. Therefore the reduction needs much time.

Table 5
Computation times for Hilbert bases.

Input	4ti2	Nmz -d 1x	Nmz -d 20x	Nmz -N 1x	Nmz -N 20x
CondPar	0.024 s	0.014 s	0.026 s	2.546 s	0.600 s
PlVsCut	6.672 s	0.820 s	0.476 s	–	–
CEffPl	6:08 m	28.488 s	3.092 s	–	–
4x4	0.008 s	0.003 s	0.011 s	0.005 s	0.016 s
5x5	3.823 s	1.004 s	0.339 s	1:06 m	23.714 s
6x6	115:26:31 h	14:19:39 h	1:19:34 h	–	–
bo5	T	–	–	0.273 s	0.174 s
lo6	31:09 m	1:46 m	39.824 s	1:08 m	13:369 s
small	48:19 m	18:45 m	3:25 m	1.935 s	1.878 s
big	T	–	–	1:45 m	15.636 s
cyclo36	T	–	–	0.774 s	0.837 s
cyclo38	R	–	–	6:32:50 h	1:04:04 h
cyclo60	R	–	–	2:55 m	1:02 m
A443	T	–	–	1.015 s	0.270 s
A553	R	–	–	44:11 m	4:24 m

Table 6
Computation times for Hilbert series and Hilbert polynomials.

Input	LattE ES	LattE+M ES	LattE EP	Nmz 1x	Nmz 20x
CondPar	O	S	–	18.085 s	8.949 s
PlVsCut	O	S	–	–	145:43:03 h
CEffPl	O	S	–	–	197:45:10 h
4x4	0.329 s	4.152 s	–	0.006 s	0.018 s
5x5	O	72:39:23 h	–	3:59 m	1:12 m
bo5	T	T	T	82:40:18 h	6:41:12 h
lo6	R	R	T	13:02:44 h	1:21:52 h
small	46.266 s	30:15 m	22.849 s	0.233 s	0.095 s
big	R	R	10.246 s	1.473 s	0.148 s
cyclo36	R	R	23:03 m	1.142 s	1.106 s
cyclo38	R	R	R	26.442 s	22.789 s
cyclo42	R	R	1:44:07 h	3.942 s	1.521 s
cyclo60	R	R	T	5:57 m	1:44 m
A443	R	R	R	49.541 s	18.519 s
A553	R	R	T	88:21:18 h	6:29:05 h
cross10	T	T	9.550 s	0.016 s	0.022 s
cross15	R	R	21:48 m	0.536 s	0.533 s
cross20	R	R	R	26.678 s	26.029 s

The Hilbert basis computations in the Normaliz primary mode show the efficiency of partial triangulations (see Section 4.4). Some numerical data are contained in Bruns et al. (2011).

We have omitted the `cross` examples from the Hilbert basis computation in view of the obvious unimodular triangulation of the cross polytopes (different from the one used by Normaliz). `cross20` needs 16 s for `Nmz -N x1`.

6.3. *Hilbert series*

Now we compare the computation times for Hilbert series of Normaliz and LattE. One should note that the computations with LattE are not completely done by open source software: for the computation of Hilbert series it invokes the commercial program Maple. LattE has a variant for the computation of Hilbert polynomials that avoids Maple; however, it can only be applied to lattice polytopes (and not to rational polytopes in general).

There are three columns with computation times for LattE. The first, `LattE ES`, lists the times for LattE alone, without Maple, the second, `LattE + M ES`, the combined computation time of LattE and Maple (both for Hilbert series), and the third, `LattE EP`, the computation time of LattE for the Hilbert polynomial. In all of these three columns we have chosen the best time that we have been able to reach with various parameter settings for LattE. However, LattE has failed on many candidates, partly because it produces enormous output files. We have stopped it when the time exceeded 150 h (T), the memory usage was more than 100 GB RAM (R) or it has produced more than 400 GB of output (O). These limitations were imposed by the system available for testing. In three cases it has exceeded the system stack limit; this is marked by S.

It is easy to see that `cross`n has Hilbert series $(1+t)^n/(1-t)^{n+1}$. Therefore it is a good test candidate for the correctness of the algorithm.

Remark 10. (a) From the Hilbert series calculation of `PlVsCut` we have obtained the following statistics on the types of simplicial cones:

1. $61,845,707,957$ are unimodular,
2. $108,915,272,879$ are not unimodular, but satisfy condition (PU1), and of these
3. $62,602,898,779$ are potentially unimodular.

This shows that condition (PU2) that was added at a later stage has a satisfactory effect. (The number of potentially unimodular, but nonunimodular simplicial cones is rather high in this class.) The average value of $|\det G_\sigma|$ is ≈ 10. This can be read off Table 4 since the sum of the $|\det G_\sigma|$ is the number of components of the Stanley decomposition.

The number of nongeneric simplicial cones is 129,661,342. The total number s of linear systems that had to be solved for the computation of the Hilbert series is bounded by $516{,}245{,}872{,}838 \le s \le 516{,}375{,}534{,}180$.

The total number of pyramids was 80,510,681. It depends on the number of parallel threads that are allowed.

(b) For examples with a high proportion of unimodular cones the exploitation of unimodularity based on Proposition 7 is very efficient in volume computations. With this strategy, `lo6` requires only 102,526,351 determinant calculations instead of 5,801,113,080. For `PlVsCut` it saves about 25%.

(c) For the examples from social choice theory (`CondPar`, `CEffPl`, `PlVsCut`) Schürmann (2013) has suggested a very efficient improvement via symmetrization that replaces the Ehrhart series of a polytope by the generalized Ehrhart series of a projection. Normaliz now has an offspring, NmzIntegrate, that computes generalized Ehrhart series; see Bruns and Söger (2015).

The volumes of the pertaining polytopes had already been computed by Schürmann with LattE integrale. This information was very useful for checking the correctness of Normaliz.

(d) The short Normaliz computation times for the `cyclo` and `cross` examples are made possible by the special treatment of simplicial facets in the Fourier–Motzkin elimination; see Bruns and Ichim (2010).

Acknowledgements

The authors like to thank Mihai Cipu, Matthias Köppe, Achill Schürmann, Bernd Sturmfels, Alin Ştefan and Volkmar Welker for the test examples that were used during the recent development of Normaliz and for their useful comments. We are grateful to Elisa Fascio for her careful reading of the first version and Lukas Katthän for a reference to Stanley (1996) in connection with the order vector.

Bogdan Ichim was partially supported a grant of CNCS – UEFISCDI, project number PN-II-RU-TE-2012-3-0161 during the preparation of this work and the development of Normaliz.

The Normaliz project is supported by the DFG Schwerpunktprogramm 1489 "Algorithmische und experimentelle Methoden in Algebra, Geometrie und Zahlentheorie".

References

4ti2 team, 2015. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at http://www.4ti2.de.

Aardal, K., Weismantel, R., Wolsey, L.A., 2002. Non-standard approaches to integer programming. In: Workshop on Discrete Optimization, DO'99. Piscataway, NJ, Discrete Appl. Math. 123, 5–74.

Baldoni, V., Berline, N., De Loera, J.A., Dutra, B., Köppe, M., Moreinis, S., Pinto, G., Vergne, M., Wu, J., 2015. A User's Guide for LattE integrale v1.7.2. Software package LattE is available at http://www.math.ucdavis.edu/~latte/.
Beck, M., Hoşten, S., 2006. Cyclotomic polytopes and growth series of cyclotomic lattices. Math. Res. Lett. 13, 607–622.
Beck, M., Robins, S., 2007. Computing the continuous discretely: Integer-point enumeration in polyhedra. Springer. Electronically available at http://math.sfsu.edu/beck/ccd.html.
Bogart, T., Raymond, A., Thomas, R.R., 2010. Small Chvatal rank. Math. Program., Ser. A 124, 45–68.
Bruns, W., Gubeladze, J., 2009. Polytopes, Rings and K-Theory. Springer.
Bruns, W., Gubeladze, J., Henk, M., Martin, A., Weismantel, R., 1999. A counterexample to an integer analogue of Carathéodory's theorem. J. Reine Angew. Math. 510, 179–185.
Bruns, W., Hemmecke, R., Ichim, B., Köppe, M., Söger, C., 2011. Challenging computations of Hilbert bases of cones associated with algebraic statistics. Exp. Math. 20, 25–33.
Bruns, W., Herzog, J., 1998. Cohen–Macaulay Rings, rev. ed. Cambridge University Press.
Bruns, W., Ichim, B., 2010. Normaliz: algorithms for affine monoids and rational cones. J. Algebra 324, 1098–1113.
Bruns, W., Ichim, B., Söger, C., 2015. Normaliz. Algorithms for rational cones and affine monoids. Available from http://www.math.uos.de/normaliz.
Bruns, W., Koch, R., 2001. Computing the integral closure of an affine semigroup. Univ. Iagel. Acta Math. 39, 59–70.
Bruns, W., Söger, C., 2015. Generalized Ehrhart series and integration in Normaliz. J. Symb. Comput. 68, 75–86.
Burton, B.A., 2014. Regina: software for 3-manifold theory and normal surfaces. Available from http://regina.sourceforge.net/.
Clauss, P., Loechner, V., Wilde, D., 1998. Ehrhart polynomials for precise program analysis. Project at http://icps.u-strasbg.fr/Ehrhart/Ehrhart.html.
Cox, D.A., Little, J., Schenck, H.K., 2011. Toric Varieties. American Mathematical Society.
Craw, A., Maclagan, D., Thomas, R.R., 2007. Moduli of McKay quiver representations II: Gröbner basis techniques. J. Algebra 316, 514–535.
De Loera, J.A., Hemmecke, R., Köppe, M., Weismantel, R., 2006. Integer polynomial optimization in fixed dimension. Math. Oper. Res. 31, 147–153.
De Loera, J.A., Hemmecke, R., Onn, S., Rothblum, U.G., Weismantel, R., 2009. Convex integer maximization via Graver bases. J. Pure Appl. Algebra 213, 1569–1577.
De Loera, J.A., Rambau, J., Santos, F., 2010. Triangulations. Structures for Algorithms and Applications. Algorithms Comput. Math., vol. 25. Springer.
Edelsbrunner, H., 1987. Algorithms in Combinatorial Geometry. Springer.
Ehrhart, E., 1977. Polynômes arithméthiques et methode des polyédres en combinatoire. Birkhäuser.
Emiris, I.Z., Kalinka, T., Konaxis, C., Luu Ba, T., 2013. Implicitization of curves and (hyper)surfaces using predicted support. Theor. Comput. Sci. 479, 81–98.
Giles, F.R., Pulleyblank, W.R., 1979. Total dual integrality and integer polyhedra. Linear Algebra Appl. 25, 191–196.
Gordan, P., 1873. Über die Auflösung linearer Gleichungen mit reellen Coefficienten. Math. Ann. 6, 23–28.
Hemmecke, R., Köppe, M., Weismantel, R., 2014. Graver basis and proximity techniques for block-structured separable convex integer minimization problems. Math. Program., Ser. A 145, 1–18.
Hemmecke, R., Onn, S., Weismantel, R., 2011. A polynomial oracle-time algorithm for convex integer minimization. Math. Program., Ser. A 126, 97–117.
Hilbert, D., 1890. Über die Theorie der algebraischen Formen. Math. Ann. 36, 472–534.
Joswig, M., Müller, B., Paffenholz, A., 2009. Polymake and lattice polytopes. In: Krattenthaler, C., et al. (Eds.), DMTCS Proc. AK, Proceedings of FPSAC, pp. 491–502.
Kappl, R., Ratz, M., Staudt, C., 2011. The Hilbert basis method for D-flat directions and the superpotential. J. High Energy Phys. 10 (27), 1–10.
Kleinschmidt, P., Smilansky, Z., 1991. New results for simplicial spherical polytopes. In: Goodman, J.E., Pollack, R., Steuger, W. (Eds.), Discrete and Computational Geometry. DIMACS Ser. Discret. Math. Theor. Comput. Sci., vol. 6.
Köppe, M., Verdoolaege, S., 2008. Computing parametric rational generating functions with a Primal Barvinok algorithm. Electron. J. Comb. 15 (R16), 1–19.
Ohsugi, H., Hibi, T., 2006. Toric ideals arising from contingency tables. In: Commutative Algebra and Combinatorics. Ramanujan Math. Soc. Lect. Notes Ser., vol. 4, pp. 87–111.
Pottier, L., 1996. The Euclidean algorithm in dimension n. Research report. In: ISSAC 96. ACM Press.
Schrijver, A., 1998. Theory of Linear and Integer Programming. Wiley.
Schürmann, A., 2013. Exploiting polyhedral symmetries in social choice. Soc. Choice Welf. 40, 1097–1110.
Sebő, A., 1990. Hilbert bases, Carathéodory's theorem, and combinatorial optimization. In: Kannan, R., Pulleyblank, W. (Eds.), Integer Programming and Combinatorial Optimization. University of Waterloo Press, Waterloo, pp. 431–456.
Stanley, R.P., 1982. Linear Diophantine equations and local cohomology. Invent. Math. 68, 175–193.
Stanley, R.P., 1996. Combinatorics and Commutative Algebra, second ed. Birkhäuser.
Sturmfels, B., Welker, V., 2012. Commutative algebra of statistical ranking. J. Algebra 361, 264–286.
van der Corput, J.G., 1931. Über Systeme von linear-homogenen Gleichungen und Ungleichungen. Proc. K. Ned. Akad. Wet. 34, 368–371.