THE POWER OF PYRAMID DECOMPOSITION IN NORMALIZ

WINFRIED BRUNS, BOGDAN ICHIM, AND CHRISTOF SÖGER

ABSTRACT. 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 Hilbert series related to combinatorial voting theory.

1. INTRODUCTION

Normaliz is a software tool for the computation of Hilbert bases and enumerative data of rational cones and affine monoids. In the 14 years of its existence it has found numerous applications; for example, in integer programming (Bogart, Raymond and Thomas [5]), algebraic geometry (Craw, Maclagan and Thomas [16]), theoretical physics (Kappl, Ratz and Staudt [30]), commutative algebra (Sturmfels and Welker [39]) or elimination theory (Emiris, Kalinka, Konaxis and Ba [23]). Normaliz is used in polymake [29], a computer system for polyhedral geometry, and in Regina [13], a system for computations with 3-manifolds.

The mathematics of the very first version was described in Bruns and Koch [11], and the details of version 2.2 (2009) are contained in Bruns and Ichim [9]. 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, Little and Schenck [15]. 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 [6]. The existence and uniqueness of such a minimal system of generators is essentially due to Gordan [25] and was proven in full generality by van der Corput [40].

The term "Hilbert basis" was actually coined in integer programming (with $L = \mathbb{Z}^d$) by Giles and Pulleyblank [24] in connection with totally dual integral (TDI) systems. Also see Schrijver [34, Sections 16.4 and 22.3]. One should note that in integer programming

2010 *Mathematics Subject Classification.* 52B20, 13F20, 14M25, 91B12.

Key words and phrases. Hilbert basis, Hilbert series, rational polytope, volume, triangulation, pyramid decomposition.

[1] Version 2.10.1 has been uploaded to `http://www.math.uos.de/normaliz` on June 27, 2013.

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, Weismantel and Wolsey [2] 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 [1] 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, Gubeladze, Henk, Weismantel and Martin [7]) that was proposed by Sebő [36]. For more recent developments in nonlinear optimization using Graver bases, and therefore Hilbert bases, see J. De Loera, R. Hemmecke, S. Onn, U.G. Rothblum, R. Weismantel [18], Hemmecke, Köppe and Weismantel [26], and Hemmecke, Onn and Weismantel [27].

Hilbert functions and polynomials of graded algebras and modules were introduced by Hilbert himself [28] (in contrast to Hilbert bases). These invariants, and the corresponding generating functions, the Hilbert series, are fundamental in algebraic geometry and commutative algebra. See [6, Chapter 6] for a brief introduction to this fascinating area. Ehrhart functions were defined by Ehrhart [22] as lattice point counting functions in multiples of rational polytopes; see Beck and Robbins [4] for a gentle introduction. Stanley [38] 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, Hemmecke, Köppe and Weismantel [17]. Surprisingly, Ehrhart functions have an application in compiler optimization; see Clauss, Loechner and Wilde [14] for more information.

From the very beginning Normaliz has used lexicographic triangulations; see [9], [11] for the use in Normaliz and De Loera, Rambau and Santos [20] for (regular) triangulations of polytopes. (Since version 2.1 Normaliz contains a second, triangulation free Hilbert basis algorithm, originally due to Pottier [33] and called *dual* in the following; see [9]). Lexicographic triangulations are essentially characterized by being incremental in the following sense. Suppose that the cone C is generated by vectors $x_1, \dots, x_n \in \mathbb{R}^d$ and set $C_i = \mathbb{R}_+x_1 + \cdots + \mathbb{R}_+x_i$, $i = 0, \dots, n$. 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 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 2 for the details and Figure 1 for a simple example.

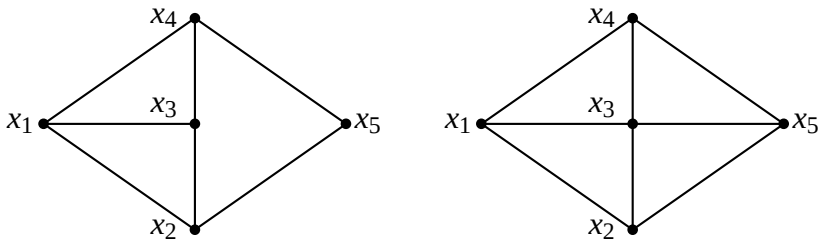

FIGURE 1. A pyramid decomposition and a lexicographic triangulation

Pure pyramid decomposition is very memory friendly, but its computation times are even 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 3.

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 [8] (jointly with Hemmecke and Köppe) is based, triggered the use of pyramid decomposition as a general principle. See Remark 8 for a brief discussion.

In Section 4 we describe the steps by which Normaliz evaluates the simplicial cones in the triangulation for the computation of Hilbert bases, volumes and Hilbert series. The 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 [37]. That it can be found efficiently relies crucially on an idea of Köppe and Verdoolaege [31].

We document the scope of Normaliz's computations in Section 5. The computation times are compared with those of 4ti2 [1] (Hilbert bases) and LattE [19] (Hilbert series). The test examples have been chosen from the literature (Beck and Hoşten [3], Ohsugi and Hibi [32], Schürmann [35], Sturmfels and Welker [39]), the LattE distribution and the Normaliz distribution. The desire to master the Hilbert series computations asked for in Schürmann's paper [35] was an important stimulus in the recent development of Normaliz.

2. Lexicographic triangulation and Pyramid decomposition

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 a triangulation of the cone

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

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

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

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 always 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 cones 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 $i = 1,\dots,n$.*

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 $i_1 < \cdots < i_m$.*
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 $j_1 < \cdots < j_{n-d}$ 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). □

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 $\mathrm{cone}(x_{i_1},\dots,x_{i_d})$ 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:

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

where $\mathscr{H}(C)$ is a finite set of hyperplanes and the orientation of the closed half spaces H^- and H^+ is chosen in such a way that $C \subset H^+$ for $H \in \mathscr{H}(C)$. For $H \in \mathscr{H}(C_{i-1})$ 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 $\Lambda(x_1,\dots,x_n)$ in a more formal way in Table 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 further use we introduce the notation

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

Table 1 formalizes the computation of the lexicographic triangulation.

The function ADDSIMPLEX adds a simplicial cone to the (initially empty) list of simplicial cones that, upon completion, contains the lexicographic triangulation of C. The function FINDNEWHYP computes $\mathscr{H}(C_i)$ from $\mathscr{H}(C_{i-1})$ by Fourier-Motzkin elimination. (It does nothing if $x_i \in C_{i-1}$.) Its Normaliz implementation has been described in great detail in [9]; therefore we skip it here. The function EXTENDTRI does exactly

```
LexTriangulation(x_1,...,x_n)
1  AddSimplex(x_1,...,x_d)
2  for i ← d+1 to n
3  do
4     ExtendTri(i)
5     FindNewHyp(i)
```

```
ExtendTri(i)
1  parallel for H ∈ H^<(C_{i-1},x_i)
2  do
3     for σ ∈ Λ(x_1,...,x_{i-1})
4     do
5        if |σ ∩ H| = d-1
6           then AddSimplex(x_i ∪ (σ ∩ H))
```

TABLE 1. Incremental building of lexicographic triangulation

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 $x_i \in C_{i-1}$).

Note that the set of hyperplanes over which the loop in ExtendTri runs is given by $\mathscr{H}^<(C_{i-1},x_i)$.

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 $H \in \mathscr{H}^<(C_{i-1},x_i)$ 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 $|\mathscr{H}^<(C_{i-1},x_i)| \cdot |\Sigma|$ 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.

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 3. 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

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

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 Figure 1 for an example.

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

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

Note that this recursion cannot descend indefinitely, since the number of generators goes down in each recursion level. We denote the *total pyramid decomposition* by $\Pi^\infty(x_1,\dots,x_n)$. More precisely:

Proposition 4. *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). □

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

```
TOTALPYRDEC(x_1,...,x_n)
1  ADDSIMPLEX(x_1,...,x_d)
2  for i ← d+1 to n
3  do
4     PROCESSPYRSREC(i)
```

```
PROCESSPYRSREC(i)
1  for H ∈ H^<(C_{i-1},x_i)
2  do key ← {x_i} ∪ ({x_1,...,x_{i-1}} ∩ H)
3     TOTALPYRDEC(key)
```

TABLE 2. Total pyramid decomposition

first realizes the building of $\Pi(x_1,\dots,x_n)$ (represented by its full dimensional members) and the second takes care of the recursion that defines $\Pi^\infty(x_1,\dots,x_n)$.

Pyramid decomposition has the virtue of requiring very little memory since the triangulation need not be stored for its future extension. However, there is a severe drawback:

as above, one must compute the support hyperplanes in $\mathscr{H}(P)$ for all pyramids encountered. In a "pure" approach, one computes the support hyperplanes of the simplicial cones at the bottom of the pyramid decomposition; this is essentially the inversion of the matrix of its generators (see equation (4.1)). Then one passes them back from a pyramid to its "mother", discarding those that fail to have all generators of the "mother" in its positive halfspace or have been found previously. These two conditions are easily tested. Suppose P is the pyramid to which TOTALPYRDEC is applied in PROCESSPYRSREC and $G \in \mathscr{H}(P)$. Then $G \in \mathscr{H}(x_1 \ldots, x_i) \setminus \mathscr{H}(x_1, \ldots, x_{i-1})$ if and only if the following two conditions are satisfied:

(i) $x_j \in G^+$ for $j = 1, \ldots, i-1$;
(ii) $x_j \in G^>$ for all $j = 1, \ldots, i-1$ such that $x_j \notin P$.

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 being very memory efficient, total pyramid decomposition is in a naïve implementation much slower than building the lexicographic triangulation directly. For one of our standard test examples ($4 \times 4 \times 3$ contingency tables, dimension 30 with 48 extreme rays; see [8]) the lexicographic triangulation with respect to the order of generators in the input file has $2,654,272$ full dimensional cones. In serial computation on an Intel i7 2600 PC, LEXTRIANGULATION computes it in approximately 2 minutes, whereas TOTALPYRDEC needs about 11 minutes. The current implementation, described in the next section, reduces the serial computation time to 13 seconds.

Remark 5. Pyramid decomposition is not only useful for the computation of triangulations, but also helps in finding support hyperplanes. For them the critical complexity parameter is $|\mathscr{H}^<(C_{i-1}, x_i)| \cdot |\mathscr{H}^>(C_{i-1}, x_i)|$, and as in its use for triangulation, pyramid decomposition lets us replace a 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. Nevertheless pyramid decomposition leads to a substantial reduction in computing time also for support hyperplanes, and Normaliz uses this effect. We illustrate this by computation times for the $5 \times 5 \times 3$ contingency tables (dimension 55 with 75 extreme rays; see [8]). The cone has $306,955$ support hyperplanes. On a Sun xFire 4450 we measured a serial computation time of $16,822$ seconds if only FINDNEWHYP is used. The current implementation reduces this to $4,334$ seconds.

3. THE CURRENT IMPLEMENTATION

Since version 2.7 (and partly since 2.5) Normaliz has combined lexicographic triangulation with pyramid decomposition. The support hyperplanes and the triangulation are extended from one generator to the next only until certain bounds are exceeded. From that point on, the algorithm BUILDCONE described in Table 3 switches to pyramid decomposition, and the same mixed strategy is then applied to the pyramids.

We now use two types of passage to pyramids, a recursive one via PROCESSPYRSREC and a nonrecursive one via PROCESSPYRS. The main reason for this split approach is that on the one hand recursion limits the effect of parallelization (as it does in Normaliz 2.7),

```
BUILDCONE(x_1,...,x_n; recursion)
 1  ADDSIMPLEX(x_1,...,x_d)
 2  for i ← d+1 to n
 3  do
 4     if MakePyramidsSupp & recursion
 5       then PROCESSPYRSREC(i)
 6       else  if MakePyramidsTri
 7                then PROCESSPYRS(i, level)
 8                else  EXTENDTRI(i)
 9             FINDNEWHYP(i)
10  if TopCone
11    then EVALUATEPYRS(0)
```

```
PROCESSPYRSREC(i)
1  for H ∈ ℋ^<(C_{i-1}, x_i)
2  do key ← {x_i} ∪ ({x_1,...,x_{i-1}} ∩ H)
3     BUILDCONE(key, true)
```

```
PROCESSPYRS(i, level)
1  for H ∈ ℋ^<(C_{i-1}, x_i)
2  do key ← {x_i} ∪ ({x_1,...,x_{i-1}} ∩ H)
3     STOREPYR(key, level+1)
```

TABLE 3. Combining lexicographic triangulation and pyramid decomposition

and, on the other hand, the recursive approach nevertheless saves time in the computation of support hyperplanes for the top cone.

The boolean *recursion* indicates whether the recursive passage to pyramids is allowed. For the top cone BUILDCONE is called with *recursion* = *true*. The boolean *MakePyramidsSupp* combines two conditions:

(1) while set to *false* initially, it remains *true* once the branch PROCESSPYRSREC has been taken the first time;
(2) it is set *true* if the complexity parameter $|\mathscr{H}^<(C_{i-1},x_i)|\cdot|\mathscr{H}^>(C_{i-1},x_i)|$ exceeds a threshold.

In the nonrecursive passage to pyramids we cut the umbilical cord between a pyramid and its mother and just store the pyramid for later evaluation. The nonrecursive call is controlled by the boolean *MakePyramidsTri* that combines three conditions:

(1) while set to *false* initially, it remains *true* once the branch PROCESSPYRS has been taken the first time;
(2) it is set *true* if the complexity parameter $|\mathscr{H}^<(C_{i-1},x_i)|\cdot|\Sigma|$ 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 the stored 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 $|\mathscr{H}^<(C_{i-1},x_i)|$ is small, and therefore condition (2) is reached only for large $|\Lambda(x_1,\dots,x_{i-1})|$. The variable *level* indicates the generation of the pyramid; for the top cone it has value -1, and increases by 1 with each new generation.

At the end of BUILDCONE for the top cone C we start the evaluation of the stored pyramids as described in Table 4.

```
EVALUATEPYRS(level)
1  if PyramidList[level] = ∅
2    then return
3  parallel for P ∈ PyramidList[level]
4  do
5     BUILDCONE(P, false)
6  EVALUATEPYRS(level + 1)
```

TABLE 4. Evaluation of pyramids

Remark 6. (a) For efficiency Normaliz completely avoids nested parallelization. Therefore the parallelization in FINDNEWHYP and EXTENDTRI is switched off when the parallelization in EVALUATEPYRS is active. On the other hand, these are active when the top cone or recursively built pyramids are being processed.

(b) Despite of considerable efforts we have not found a completely satisfactory solution in which pyramids could always be processed recursively and simultaneously in parallel. Because of (a) we can only parallelize the pyramids directly produced from the top cone in the recursive approach, and then parallelization may be limited by an insufficient number of hyperplanes in $\mathscr{H}^<(C_{i-1},x_i)$ or, more often, by enormous differences in the sizes of the pyramids, so that one of them may be running solo for a long time—recognizing the size in advance has turned out difficult. Parallelization in FINDNEWHYP and EXTENDTRI is then the better solution.

Moreover, a large pyramid together with its children may produce a huge number of simplicial cones and overrun the evaluation buffer. Serial loops can be interrupted at any time, and therefore the memory problem cannot arise.

(c) As soon as BUILDCONE switches to pyramids, the triangulation $\Lambda(x_1,\dots,x_{i-1})$ is no longer needed for further extension. Therefore it is shipped to the evaluation buffer. The buffer is emptied whenever it has exceeded its preset size and program flow allows its parallelized evaluation. (Because of (a) this is not always possible.)

(d) The strategy for the evaluation of pyramids is similar. If the buffer for $level+1$ is exceeded, evaluation on that level will be started as soon as possible. Usually this results in a tree of evaluations over several levels.

We add a few minor details of the implementation.

Remark 7. (a) For nonrecursive pyramids the support hyperplanes arising from the last generator need not be computed since they are irrelevant for triangulation and pyramid decomposition.

(b) Simplicial facets of C_{i-1} produce exactly one simplicial pyramid in C_i. They are treated directly by ADDSIMPLEX.

(c) If the extreme rays of C have been singled out from the given generators $x_1,\dots,x_n$ before BUILDCONE is called, then only the extreme rays are used in the pyramid decomposition and the lexicographic triangulation.

(d) If a grading is defined explicitly (see Section 4), then Normaliz orders the generators by degree and those of the same degree by input order before building the cone C. This is

an attempt to cover as much ground as possible by using generators of small degree. On the whole, we have reached good results with this choice.

Remark 8. *(Partial triangulation)* The idea of pyramid decomposition was born when the authors observed that the computation of Hilbert bases usually 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 $\mathrm{ht}_H(x_i) = 1$ in PROCESSPYRS.

The resulting strategy has sometimes striking results and was already described in [8]. It is mentioned here only for completeness. If a full triangulation is not required, then PROCESSPYRS discards all pyramids of height 1 from further processing. (However, their support hyperplanes must be computed if processed recursively.) If followed strictly, the recursion will not stop before the simplicial cones at the bottom of the pyramid decomposition. As for full triangulations, this is usually not optimal. Normaliz therefore switches to EXTENDTRI for pyramids of height ≥ 2 from a certain level on.

4. 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, which is almost always the more time consuming step.

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 2 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

$$\lambda_i(v_k) = \sum_{j=1}^{d} v_{kj}a_{ji} = \begin{cases} \mathrm{ht}_{H_i}(v_i), & k = i, \\ 0, & k \neq i. \end{cases} \tag{4.1}$$

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),
(Ehr) the *Hilbert* or *Ehrhart function* $H(C,k) = |kP \cap \mathbb{Z}^d|$, $k \in \mathbb{Z}_+$.

4.1. **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 9. *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 [6, 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 $|\det G_\sigma| = 1$ there is no need to compute $|\det(G_\tau)|$ since the height of the "new" generator v_d over F is known. Remark 13(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.

4.2. **Lattice points in semi-open parallelotopes.** The remaining tasks depend on the set E of lattice points in the semi-open parallelotope

$$\mathrm{par}(v_1,\dots,v_d) = \{q_1v_1 + \cdots + q_dv_d : 0 \le q_i < 1\}.$$

For the efficiency of the evaluation it is important to generate $E = \mathbb{Z}^d \cap \mathrm{par}(v_1,\dots,v_d)$ as fast as possible. The basic observation is that E is a set of representatives of the group $\mathbb{Z}^d/U_\sigma$ where the subgroup U_σ is spanned by $v_1,\dots,v_d$. Thus one finds E in two steps:

(Rep) find a representative of every residue class, 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 $\overline{\ }$ 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 10. *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 $a_1,\dots,a_d\geq 1$. Then the vectors*

$$b_1e_1+\cdots+b_de_d,\qquad 0\leq b_i<a_i,\ i=1,\dots,d, \tag{4.2}$$

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 $b_1e_1+\cdots+b_pe_p$ and $b'_1e_1+\cdots+b'_pe_p$ 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. □

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

$$XG_\sigma = D. \tag{4.3}$$

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 (4.2), and thus we must solve the simultaneous linear systems

$$G_\sigma^{\mathrm{tr}}x_i = e_i,\qquad a_i>1, \tag{4.4}$$

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 (4.4). 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 [9], and nothing essential has been added meanwhile.

4.3. **Hilbert series and Stanley decomposition.** The most difficult and mathematically most interesting task is (Ehr). For its solution one uses the well-known fact that the *Hilbert* or *Ehrhart series*, the generating function

$$H_C(t) = \sum_{k=0}^{\infty} H(C,k)t^k,$$

is a rational function of t. For σ one has

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

This follows immediately from the disjoint decomposition

$$\mathbb{Z}^d \cap \sigma = \bigcup_{x\in E} x + M_\sigma \tag{4.5}$$

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

However, one cannot compute $H_C(t)$ by simply adding these functions 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 of C by semi-open simplicial cones $\sigma \setminus S$ where S is a union of facets (and not just arbitrary faces!) of σ. The series $H_{\sigma\setminus S}(t)$ is as easy to compute as $H_\sigma(t)$ itself. Let $x \in E$, $x = \sum q_i v_i$. 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. Then

$$H_{\sigma\setminus S}(t) = \frac{\sum_{x\in E} t^{\deg \varepsilon(x) + \deg x}}{(1-t^{g_1})\cdots(1-t^{g_d})}. \tag{4.6}$$

This follows from the fact that $(x+M_\sigma)\setminus S = \varepsilon(x) + x + M_\sigma$, and so we just sum over the disjoint decomposition of $\mathbb{Z}^d \cap (\sigma \setminus S)$ induced by (4.5). (Also see [9, Lemma 11].)

The existence of a disjoint decomposition of C into sets of type $\sigma \setminus S$ was shown by Stanley [37] 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 [9]. 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. Köppe and Verdoolaege [31] proved a much simpler principle for the disjoint decomposition (already implemented in Normaliz 2.7). 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 11. *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 $\mathscr{H}^<(\sigma, O_C)$. Then the semi-open simplicial cones $\sigma \setminus S_\sigma$ form a disjoint decomposition of C.*

See [31] for a proof. 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 [21]). 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 $O'_C \in H^>$.

At first it seems that one must compute the support hyperplanes of σ in order to apply Lemma 11. However, it is much better to solve the system

$$G_\sigma^{\mathrm{tr}} I^\sigma = O_C. \tag{4.7}$$

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 (4.1) this amounts to solving the systems

$$G_\sigma x = e_i, \qquad I_i^\sigma = 0, \tag{4.8}$$

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 (4.7), 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 13(a)).

After these preparations we can describe the order in which Normaliz treats the trigonalization (4.3) and the linear systems (4.4), (4.7) and (4.8):

(L1) If σ is potentially unimodular, then (4.7) is solved first. It can now be decided whether σ is indeed unimodular.

(L2) If σ is not unimodular, then the trigonalization (4.3) is carried out next. In the potentially unimodular, but nongeneric case, the trigonalization is part of the solution of (4.8) (with multiple right hand side).

(L3) In the nonunimodular case, we now solve the system (4.4) (with multiple right hand side).

(L4) If σ is not potentially unimodular and not generic, it remains to solve the system (4.8) (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.

Remark 12. (a) If one stores the transformation matrix X of (4.3) and its inverse (for example as a sequence of row exchanges and elementary transformations), then one can solve the remaining systems without further trigonalization. However, in general the bookkeeping needs more time than it saves as tests have shown.

(b) The simplicial cones stored in the evaluation buffer are processed in parallel, and parallelization works very well for them.

(c) Simplicial cones of height 1 need not be evaluated for (HB) and (LP); see Remark 8.

4.4. **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 [6, Chapter 6].

Summing the Hilbert series (4.6) 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

$$H_C(t) = \frac{R(t)}{(1-t^{s_1})\cdots(1-t^{s_r})}, \qquad R(t) \in \mathbb{Z}[t], \tag{4.9}$$

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 (4.9) into a product of cyclotomic polynomials:

$$H_C(t) = \frac{Z(t)}{\zeta_{z_1}\cdots\zeta_{z_w}}, \qquad Z(t) \in \mathbb{Z}[t], \quad \zeta_{z_j} \nmid Z(t), \tag{4.10}$$

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 (4.6) divide $(1-t^\ell)^d$ where ℓ is the least common multiple of the degrees $\deg x_i$. So we can find a representation

$$H_C(t) = \frac{F(t)}{(1-t^{e_1})\cdots(1-t^{e_d})}, \qquad F(t) \in \mathbb{Z}[t], \tag{4.11}$$

in which e_d is the least common multiple of the orders of the cyclotomic polynomials that appear in (4.10), e_{d-1} is the least common multiple of the orders that have multiplicity ≥ 2 etc. Normaliz produces the presentation (4.11) whenever the degree of the numerator remains of reasonable size.

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

$$H(C,k) = q_0(k) + q_1(k)k + \cdots + q_{d-1}(k)k^{d-1}, \qquad k \geq 0, \tag{4.12}$$

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 (4.10). 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 (4.11) 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

$$q_{d-1} = \frac{\mathrm{vol}(P)}{(d-1)!}. \tag{4.13}$$

Since q_{d-1} and $\mathrm{vol}(P)$ are computed completely independently from each other, equation (4.13) can be regarded as a test of correctness for both numbers.

The choice (4.11) 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 (4.9)–(4.11) 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.

5. Computational results

In this section we want to document that the algorithmic approach described in the previous sections (and [9]) is very efficient and masters computations that appeared inaccessible some years ago. We compare Normaliz 2.8 to 4ti2, version 1.5, for Hilbert basis computations and to LattE, version 1.5, for Ehrhart series (the versions we have used are both contained in the package LattE integrale 1.5.3 [19]).

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 6 and 7 `-x=1` indicates serial execution whereas `-x=20` 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.

5.1. **Overview of the examples.** We have chosen the following test candidates:

(1) `CondPar`, `CEffPl` and `PlVsCut` come from combinatorial voting 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 [35] 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 [19].
(3) `bo5` and `lo6` belong to the area of statistical ranking; see Sturmfels and Welker [39]. `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 [9].
(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 [3].
(6) `A443` and `A553` represent monoids defined by dimension 2 marginal distributions of dimension 3 contingency tables of sizes $4 \times 4 \times 3$ and $5 \times 5 \times 3$. They had been open cases in the classification of Ohsugi and Hibi [32] and were finished in [8].
(7) `cross10`, `cross15` and `cross20` are (the monoids defined by) the cross polytopes of dimensions 10, 15 and 20 contained in the LattE distribution [19].

The columns of Table 5 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.

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 [3]. 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.

5.2. **Hilbert bases.** Table 6 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.

Input	edim	rank	#ext	#supp	#Hilb	# triangulation	# Stanley dec
CondPar	24	24	234	27	242	1.344.671	1.816.323
PlVsCut	24	24	1.872	28	9.621	271.164.705.162	2.282.604.742.033
CEffPl	24	24	3.928	30	25.192	347.225.775.338	4.111.428.313.448
4x4	16	8	20	16	20	46	48
5x5	25	15	1.940	25	4.828	12.112.488	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	5.745.903.354	5.801.113.080
small	6	6	190	32	34.591	1.593	2.276.921
big	7	7	27	56	73.551	337	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	161.249	183.120
cyclo60	17	17	60	656.100	61	12.475.500	13.616.100
A443	40	30	48	4.948	48	2.654.272	2.654.320
A553	55	43	75	306.955	75	9.248.527.905	9.249.511.725
cross10	11	11	20	1.024	21	512	1.024
cross15	16	16	30	32.678	31	16.384	32.768
cross20	21	21	40	1.048.576	41	524.288	1.048.576

TABLE 5. Numerical data of test examples

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 [8]. This approach uses symmetries to reduce the amount of computations.

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. An example illustrating this dilemma is given by lo6. 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 series of cyclo38 is large compared to the time for the Hilbert series in Table 7. The reason is the large number of support hyperplanes

Input	4ti2	Nmz -d -x=1	Nmz -d -x=20	Nmz -N -x=1	Nmz -N -x=20
CondPar	0.025 s	0.018 s	0.031 s	5.628 s	1.775 s
PlVsCut	6.697 s	1.967 s	0.567 s	–	–
CEffPl	6:09 m	2:15 m	15.75 s	–	–
4x4	0.008 s	0.003 s	0.011 s	0.004 s	0.010 s
5x5	3.835 s	1.684 s	0.382 s	4:27 m	1:40 m
6x6	123:18:24 h	20:07:14 h	1:21:11 h	–	–
bo5	T	–	–	0.588 s	0.377 s
lo6	31:32 m	16:18 m	1:19 m	60:16 m	11:54 m
small	48:48 m	38:02 m	3:33 m	3.098 s	3.861 s
big	T	–	–	3:05 m	43.758 s
cyclo36	T	–	–	1.477 s	1.092 s
cyclo38	R	–	–	36:55:53 h	2:02:32 h
cyclo60	R	–	–	7:06 m	2:17 m
A443	T	–	–	1.080 s	0.438 s
A553	R	–	–	3:15:15 h	15:59 m

TABLE 6. Computation times for Hilbert bases

together with a large number of candidates for the Hilbert basis. Therefore the reduction needs much time.

The Hilbert basis computations in the Normaliz primary mode show the efficiency of partial triangulations (see Remark 8). Some numerical data are contained in [8].

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 19.904 s with Nmz -x=20.

5.3. **Ehrhart series.** Now we compare the computation times for Ehrhart 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 Ehrhart series it invokes the commercial program Maple. LattE has a variant for the computation of Ehrhart 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 Ehrhart series), and the third, LattE EP, the computation time of LattE for the Ehrhart 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 hours (T), the memory usage was more than 100 GB RAM (R) or it has produced more than 400 GB of output

(O). These limitation were imposed by the system available for testing. In two cases it has exceeded the system stack limit; this is marked by S.

Input	LattE ES	LattE+M ES	LattE EP	Nmz -x=1	Nmz -x=20
CondPar	O	S	-	28.352 s	8.388 s
PlVsCut	O	O	-	–	175:11:26 h
CEffPl	O	S	-	–	218:13:55 h
4x4	0.579 s	3.465 s	-	0.004 s	0.010 s
5x5	O	72:39:23 h	-	5:20 m	2:47 m
bo5	T	T	T	90:05:43 h	7:20:50 h
lo6	R	R	T	14:17:09 h	2:18:14 h
small	57.890 s	39:36 m	41.337 s	0.370 s	0.274 s
big	R	R	9:15 m	1.513 s	0.334 s
cyclo36	R	R	28:55 m	1.591 s	1.653 s
cyclo38	R	R	R	40.099 s	33.220 s
cyclo42	R	R	2:14:06 h	4.784 s	4.547 s
cyclo60	R	R	R	7:47 m	5:36 m
A443	R	R	R	54.758 s	16.515 s
A553	R	R	T	112:59:33 h	7:37:04 h
cross10	T	T	11.712 s	0.016 s	0.021 s
cross15	R	R	49:06 m	0.675 s	0.487 s
cross20	R	R	R	32.820 s	23.207 s

TABLE 7. Computation times for Ehrhart series and Ehrhart polynomials

Remark 13. (a) From the Ehrhart 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 5 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 Ehrhart series is bounded by $516,245,872,838 \leq s \leq 516,375,534,180$.

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 9 is very efficient in volume computations. For example,

`lo5` requires only $102,526,351$ determinant calculations instead of $5,801,113,080$. For `PlVsCut` it saves about 25%.

(c) For the examples from combinatorial voting theory (`CondPar`, `CEffPl`, `PlVsCut`) Schürmann [35] 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 [12].

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 [9].

6. Acknowledgement

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.

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.

References

[1] 4ti2 team. `4ti2`–*A software package for algebraic, geometric and combinatorial problems on linear spaces*. Available at `http://www.4ti2.de`.

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

[3] M. Beck and S. Hoşten. *Cyclotomic polytopes and growth series of cyclotomic lattices.* Math. Res. Lett. **13** (2006), 607–622.

[4] M. Beck and S. Robins. *Computing the continuous discretely: Integer-point enumeration in polyhedra*, Springer 2007, Electronically available at `http://math.sfsu.edu/beck/ccd.html`.

[5] T. Bogart, A. Raymond and R.R. Thomas, *Small Chvatal rank*. Math. Program. Ser. A **124** (2010), 45–68.

[6] W. Bruns, J. Gubeladze, *Polytopes, rings and K-theory*, Springer, 2009.

[7] W. Bruns, J. Gubeladze, M. Henk, A. Martin, and R. Weismantel, *A counterexample to an integer analogue of Carathéodory's theorem.* J. Reine Angew. Math. **510** (1999), 179–185.

[8] W. Bruns, R. Hemmecke, B. Ichim, M. Köppe, and C. Söger, *Challenging computations of Hilbert bases of cones associated with algebraic statistics*. Exp. Math. **20** (2011), 25–33.

[9] W. Bruns and B. Ichim, *Normaliz: Algorithms for affine monoids and rational cones.* J. Algebra **324** (2010), 1098–1113.

[10] W. Bruns, B. Ichim and C. Söger, *Normaliz. Algorithms for rational cones and affine monoids.* Available from `http://www.math.uos.de/normaliz`.

[11] W. Bruns and R. Koch, *Computing the integral closure of an affine semigroup.* Univ. Iagel. Acta Math. **39** (2001), 59–70.

[12] W. Bruns and C. Söger, *Generalized Ehrhart series and Integration in Normaliz.* `arXiv:1211.5178`

[13] B. A. Burton, *Regina: software for 3-manifold theory and normal surfaces.* Available from `http://regina.sourceforge.net/`

[14] P. Clauss, V. Loechner and D. Wilde, *Ehrhart polynomials for precise program analysis.* Project at `http://icps.u-strasbg.fr/Ehrhart/Ehrhart.html`
[15] D. A. Cox, J. Little and H. K. Schenck, *Toric varieties.* American Mathematical Society, 2011.
[16] A. Craw, D. Maclagan and R.R. Thomas, *Moduli of McKay quiver representations II: Gröbner basis techniques*. J. Algebra **316** (2007), 514–535.
[17] J. A. De Loera, R. Hemmecke, M. Köppe and R. Weismantel, *Integer polynomial optimization in fixed dimension.* Math. Oper. Res. **31** (2006), 147–153.
[18] J. A. De Loera, R. Hemmecke, S. Onn, U. G. Rothblum and R. Weismantel, *Convex integer maximization via Graver bases.* J. Pure Appl. Algebra **213** (2009), 1569–1577.
[19] J.A. De Loera, M. Köppe et al., *LattE integrale*. Available at `http://www.math.ucdavis.edu/~latte/`
[20] J. A. De Loera, J. Rambau and F. Santos. *Triangulations. Structures for algorithms and applications.* Algorithms and Computation in Mathematics **25**. Springer, 2010.
[21] H. Edelsbrunner, *Algorithms in combinatorial geometry*. Springer 1987.
[22] E. Ehrhart *Polynômes arithméthiques et methode des polyédres en combinatoire.* Birkhäuser, 1977.
[23] I. Z. Emiris, T. Kalinka, C. Konaxis and Thang Luu Ba *Implicitization of curves and (hyper)surfaces using predicted support.* Theoret. Comput. Sci. **479** (2013), 81–98.
[24] F.R. Giles and W.R. Pulleyblank, *Total dual integrality and integer polyhedra.* Linear Algebra Appl. **25** (1979) 191–196.
[25] P. Gordan, *Über die Auflösung linearer Gleichungen mit reellen Coefficienten*. Math. Ann. **6** (1873), 23–28.
[26] R. Hemmecke, M. Köppe and R. Weismantel, *Graver basis and proximity techniques for block-structured separable convex integer minimization problems.* Math. Program. Ser. A (February 2013), DOI 10.1007/s10107-013-0638-z.
[27] R. Hemmecke, S. Onn and R.Weismantel, *A polynomial oracle-time algorithm for convex integer minimization.* Math. Program. **126** (2011), Ser. A, 97–117.
[28] D.Hilbert. *Über die Theorie der algebraischen Formen. Math. Ann.* **36** (1890), 472–534.
[29] M. Joswig, B. Müller and A. Paffenholz, *Polymake and lattice polytopes.* In *DMTCS proc. AK,* C. Krattenthaler (ed.) et al., Proceedings of FPSAC 2009, pp. 491–502.
[30] R. Kappl, M. Ratz und C. Staudt, *The Hilbert basis method for D-flat directions and the superpotential*. Journal of High Energy Physics **10** (2011), 27, 1–11.
[31] M. Köppe and S. Verdoolaege, *Computing parametric rational generating functions with a Primal Barvinok algorithm.* Electr. J. Comb. **15** (2008), R16, 1–19.
[32] H. Ohsugi and T. Hibi, *Toric ideals arising from contingency tables.* In: Commutative Algebra and Combinatorics. Ramanujan Mathematical Society Lecture Note Series **4** (2006), 87–111.
[33] L. Pottier, *The Euclide algorithm in dimension n*. Research report, ISSAC 96, ACM Press 1996.
[34] A. Schrijver, *Theory of linear and integer programming.* Wley, 1998.
[35] A. Schürmann, *Exploiting polyhedral symmetries in social choice.* Social Choice and Welfare **40** (2013), 1097–1110.
[36] A. Sebő, *Hilbert bases, Carathéodory's theorem, and combinatorial optimization*, in 'Integer Programming and Combinatorial Optimization' (R. Kannan, W. Pulleyblank, eds.), University of Waterloo Press, Waterloo 1990, 431–456.
[37] R. P. Stanley, *Linear Diophantine equations and local cohomology.* Invent. math. **68** (1982), 175–193.
[38] R. P. Stanley, *Combinatorics and commutative algebra*. second ed. Birkhäuser, 1996.
[39] B. Sturmfels and V. Welker, *Commutative algebra of statistical ranking.* J. Algebra **361** (2012), 264–286.
[40] J. G. van der Corput, *Über Systeme von linear-homogenen Gleichungen und Ungleichungen*. Proc. Kon. Nederl. Akad. Wetensch. **34** (1931), 368–371.

WINFRIED BRUNS, UNIVERSITÄT OSNABRÜCK, FB MATHEMATIK/INFORMATIK, 49069 OSNABRÜCK, GERMANY
E-mail address: wbruns@uos.de

BOGDAN ICHIM, INSTITUTE OF MATHEMATICS "SIMION STOILOW" OF THE ROMANIAN ACADEMY, C.P. 1-764, 010702 BUCHAREST, ROMANIA
E-mail address: bogdan.ichim@imar.ro

CHRISTOF SÖGER, UNIVERSITÄT OSNABRÜCK, FB MATHEMATIK/INFORMATIK, 49069 OSNABRÜCK, GERMANY
E-mail address: csoeger@uos.de