UNIVERSITÄT OSNABRÜCK

Parallel Algorithms for Rational Cones and Affine Monoids

Christof Söger

Dissertation
zur Erlangung des Doktorgrades (Dr. rer. nat.)

FB Mathematik/Informatik
Universität Osnabrück

Januar 2014

Acknowledgements

First of all I would like to thank my advisor Winfried Bruns for his inspiring guidance and support. Our discussions have been very helpful, and I am always enjoying to work with him. I also wish to thank all my colleagues at the Institute for Mathematics. They are providing a very pleasant working atmosphere.

Many thanks also to Merle for listening to my incomprehensible discourses and her standby in various ways. Moreover, I thank all of my friends for offering me advice and distraction, especially Jochen for thought-provoking discussions on algorithms and parallelization.

Special thanks to my family. They provided excellent frame conditions for my whole undergraduate and doctoral studies.

Contents

Introduction

In this thesis we will present parallel algorithms for rational cones and affine monoids which pursue two main computational goals:

- finding the Hilbert basis, a minimal generating system of the monoid of lattice points of a cone, and
- counting elements degree-wise in a generating function, the Hilbert series.

The software Normaliz [**7**] implements these algorithms. The first version of Normaliz was developed by Winfried Bruns and Robert Koch in 1997/98. In 2008 it was rewritten in C++ by Bogdan Ichim for version 2.0, and since version 2.1 it also contains the computation of Hilbert basis via a dual approach based on Pottier's algorithm [**32**]. Since then several improvements have been integrated into Normaliz, including partial triangulations, the Hilbert series computation for arbitrary $\mathbb{Z}$-gradings, and various performance improvements.

Hilbert bases and Hilbert series have found applications in various areas of mathematics, like algebraic geometry, commutative algebra and combinatorial optimization. We will give examples in the course of the thesis. There are also some unsuspected applications. For instance, the software package Regina [**12, 13, 14**] uses Normaliz. Regina is for computational 3-manifold topology. Normaliz is even used by theoretical physicists [**26**].

The goal of this dissertation is to document the improvement of existing algorithms, the development of new and faster algorithms, and also the use of parallelization for further speed up.

Parallel algorithms already play an important role in computing, and this trend will increase further. In the 1990s advantages in computing performance were driven by raising clock speed. But this strategy also increases the power consumption and with it the heat generation, which in turn limits the performance gain. Therefore, the development shifted to put multiple processing cores into a single chip and to use multiple chips in a system. Nowadays basically all computers, from smartphones over standard laptops and PCs to large servers, support the execution of multiple threads at the same time. To profit from this development, it is vital to implement efficient parallel algorithms.

Therefore one main theme is the parallelization of all relevant algorithms. In some cases this was easy to achieve, in others it forced a restructuring of the algorithms or the development of completely new ideas. Anyway, careful thoughts on the algorithm and the implementation are needed to achieve a good parallel performance. We use the OpenMP programming interface [**31**] for the implementation of the parallel algorithms in C++.

Besides the time requirements, another critical point is the memory usage. We have designed the software in such a way that intermediate data should not allocate a too large part of the memory. All examples that are mentioned in this thesis require less than 2 GB of RAM. Of course, the desired output can be almost arbitrarily large, and in this case also more memory may be needed.

Consequently we are now able to compute significantly larger examples. One might ask: "Is it really useful to compute even bigger examples?" We think it certainly is. Sometimes it is possible to handle a mathematical problem for all but some special cases, or it is shown that a problem can be reduced to few critical cases. Then it is extremely meaningful to compute these cases explicitly. But maybe even more importantly, examples are a basic tool for understanding the structure of a problem and help to derive new conjectures and results. In this way the possibility to compute examples faster and an increased range of examples support mathematicians in their research.

The geometrical and algebraic objects of this thesis are defined in Chapter 1. There we also introduce additional notations and conventions.

In Chapter 2 we describe the fundamental tools, first of all an algorithm for dualizing the cone, i.e. computing the support hyperplanes from the extreme rays and vice versa. Then we discuss the computation of a triangulation of the cone which will be used in all following algorithms. A breakthrough for complicated examples was the construction of triangulations, and also support hyperplanes, via pyramids, see Section 2.4. The pyramid decomposition not only enables us to compute such examples, but also makes high levels of parallelization possible. The last section addresses how to create a disjoint decomposition of the cone out of the triangulation.

In the next two chapters these algorithms are utilized. The Hilbert basis computation is the topic of Chapter 3. It is done in two steps: first a generating set is created, then it is reduced to the Hilbert Basis. We describe the existing algorithm and discuss its parallelization. There the triangulation is used to reduce the problem to the simplicial case. Thus, we profit here already from the advances in computing the triangulation. For some examples we can get a formidable improvement from the use of a partial triangulation. As the name suggests, the partial triangulation only covers parts of the cone and leaves out parts which do not contribute new candidates for the Hilbert basis. Afterwards we particularize the use of the disjoint decomposition in this case, especially in the interaction with the partial triangulation.

Our second main goal, the computation of Hilbert series, is addressed in Chapter 4. Again the disjoint decomposition of the cone that we gain from the triangulation is used. We discuss the semi-open simplicial case, which comes up in the disjoint decomposition of a non-simplicial cone, and the fascinating connection between Hilbert series and Hilbert quasipolynomials. The chapter ends with the special case of computing the leading coefficient of the Hilbert quasipolynomial, which is related to the volume of a polytope in the cross-section of the cone.

In Chapter 5 we give more background on three examples, together with results of our computations and some data on computation times. The chapter closes with Section 5.4, where we compare timings of Normaliz version 2.2 and the current development version, and also serial and parallel execution.

Finally we discuss bounds for the various computed values in Chapter 6. They give an a priori estimate on the necessary precision of the arithmetic. In this way we can ensure that a computation is doable on the computer with machine integers of a fixed size. The machine integers give a significant performance advantage over types that offer arbitrary precision by composing a flexible number of machine integers. While these arbitrary precision types are great when they are really needed, in calculations with them we observed a speed penalty factor of 10 and more for small integers that would also fit in the fixed size of a machine integer. Therefore, it is very useful to have these bounds and decide whether it is safe to use machine integers or not.

In this thesis we will give proofs that are strongly connected to the algorithms, since they will lay the foundation for them, even if there might be mathematically more elegant proofs.

Before we start, a final note on the illustrating examples in the course of the thesis. A general problem with geometric objects is that our imagination is limited to three dimensions. This may lead us to the false impression that some computations are easy. For instance, a lot depends on the face structure of the cone and this is quite easy for 3-dimensional cones, since already the facets are always simplicial. But do not underestimate the complexity in higher dimensions! Consider the examples and illustrations from that point of view. Sometimes they seem trivial or do not make sense since it could be done in an easier way. Rather see them as projections of higher dimensional cases to a printable dimension.

CHAPTER 1

Preliminaries

This chapter introduces the basic objects, notations and conventions that will be used in this thesis. We will not prove any of the facts in this chapter, a reference for this is [**4**].

By $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ we denote the sets of the integer, rational and real numbers, respectively. The nonnegative parts of these sets are $\mathbb{Z}_+$, $\mathbb{Q}_+$ and $\mathbb{R}_+$.

1.1. Rational cones

The central geometric objects in this thesis are rational cones. For a clearer understanding we introduce in this section the notation and recall some facts that are used later on.

Definition 1.1. A *(rational) cone* C in $\mathbb{R}^d$ is the set of all $\mathbb{R}_+$-combinations of finitely many elements $x_1, \dots, x_n \in \mathbb{Z}^d$:

$$C = \operatorname{cone}(x_1, \dots, x_n) = \mathbb{R}_+x_1 + \cdots + \mathbb{R}_+x_n = \{a_1x_1 + \cdots + a_nx_n : a_i \in \mathbb{R}_+\}.$$

The *dimension* of a cone C is the dimension of the linear subspace created by C and is denoted by $\dim C$. We call C *pointed* if $x, -x \in C$ implies $x = 0$. A cone that is generated by linear independent vectors is called *simplicial.*

A linear form $\lambda : \mathbb{R}^d \to \mathbb{R}$ specifies a *(linear) hyperplane*

$$H_\lambda = \{x \in \mathbb{R}^d : \lambda(x) = 0\}.$$

Additionally we define the *closed halfspaces*

$$H_\lambda^+ = \{x \in \mathbb{R}^d : \lambda(x) \geq 0\}, \qquad H_\lambda^- = \{x \in \mathbb{R}^d : \lambda(x) \leq 0\}$$

and the *open halfspaces*

$$H_\lambda^> = H_\lambda^+ \setminus H_\lambda, \qquad H_\lambda^< = H_\lambda^- \setminus H_\lambda.$$

The only hyperplanes of interest to us are rational hyperplanes, their representing linear forms can be chosen to have integral coprime coefficients, what we will always do. This choice is unique up to sign. If a hyperplane occurs in the context of a cone C, we will select λ such that $C \subset H_\lambda^+$.

With this choice of the linear form λ for a hyperplane H we say the *height* of $x \in \mathbb{R}^d$ over H is $|\lambda(x)|$.

Equivalently to the definition, a rational cone can be described as the intersection of finitely many closed rational halfspaces

$$C = H_1^+ \cap \cdots \cap H_s^+.$$

Hyperplanes H that satisfy $C \subset H^+$ are called *support hyperplanes* and in this case the intersection $C \cap H$ a *face* of C. Faces of cones are again cones. The faces of dimension one less than the dimension of C are called *facets* and faces of dimension 1 *extreme rays.* Each extreme ray has a unique integral vector with coprime coefficients. These vectors are the *extreme integral generators* of the cone.

In the following by support hyperplanes we only mean those that intersect C in a facet and denote the set of these support hyperplanes by $\mathcal{H}(C)$. They are the only relevant hyperplanes for the description of C and in the algorithms. For later use we also introduce

$$\mathcal{H}^*(C,x) = \{H \in \mathcal{H}(C) \ : \ x \in H^*\} \quad \text{for } * \in \{+,-,=,<,>\}$$

with $H^= = H$. We will call $\mathcal{H}^>$, $\mathcal{H}^=$ and $\mathcal{H}^<$ the *positive, neutral and negative hyperplanes*, respectively.

A subset D of a cone $C \subset \mathbb{R}^d$ is *visible* from a point $x \in \mathbb{R}^d$ if

$$D \subset \bigcup_{H \in \mathcal{H}^<(C,x)} H.$$

This definition ensures that the line segment $[y,x]$ intersects C exactly in y for every $y \in D$ and hence there is a "free view" from x to y.

To study a *rational polytope* generated by $x_1,\dots,x_n \in \mathbb{Q}^n$, i.e.

$$P = \operatorname{conv}(x_1,\dots,x_n) = \{a_1x_1 + \cdots + a_nx_n : \ a_i \in \mathbb{R}_+, a_1 + \cdots + a_n = 1\} \subset \mathbb{R}^d,$$

we will often consider the *cone over the polytope.* This is the cone

$$C = \operatorname{cone}((x_1,1),\dots,(x_n,1)) \subset \mathbb{R}^{d+1}.$$

The intersection of C with the affine hyperplane $A_1 = \{(y_1,\dots,y_{d+1}) \in \mathbb{R}^{d+1} : y_{d+1} = 1\}$ is again P (embedded in $\mathbb{R}^{d+1}$).

1.2. Affine monoids and normalization

As discrete counterparts of cones we will consider monoids.

Definition 1.2. An *affine monoid* M is a submonoid of the free abelian group $\mathbb{Z}^d$ which is generated by finitely many elements $x_1,\dots,x_n \in \mathbb{Z}^d$:

$$M = \mathbb{Z}_+x_1 + \cdots + \mathbb{Z}_+x_n = \{a_1x_1 + \cdots + a_nx_n : a_i \in \mathbb{Z}_+\}.$$

We call M *positive* if $x, -x \in M$ implies $x = 0$.

Evidently, we can generate a rational cone from an affine monoid M by $C = \mathbb{R}_+M$. For the converse direction we use *lattices*, that are subgroups of $\mathbb{Z}^d$.

Theorem 1.3 Gordon's lemma. *Let $C \subset \mathbb{R}^d$ be a rational cone and $L \subset \mathbb{Z}^d$ a lattice. Then $C \cap L$ is an affine monoid.*

Affine monoids that arise in this way have a special property: they are integrally closed with respect to L.

Definition 1.4. Let M be an affine monoid and L a lattice. The affine monoid

$$\widehat{M_L} = \{x \in L : mx \in M \text{ for some } m \in \mathbb{Z}, m > 0\}$$

is the *integral closure of* M *in* L and if $M = \widehat{M_L}$, i.e. M coincides with its integral closure, we call M *integrally closed in* L. In the case of $L = \mathbb{Z}M$, the lattice generated by M, the integral closure is called *normalization* and integrally closed monoids *normal*. For $L = \mathbb{Z}^d$ we may omit "in $\mathbb{Z}^d$".

Computing the integral closure is one of the main topics in this thesis, another is computing the number of elements in certain parts of the monoid. For this purpose we define a $\mathbb{Z}$-*grading* on the lattice L to be a surjective linear form $\deg : L \mapsto \mathbb{Z}$ with $\deg(x) \neq 0$ for all $x \in L$, $x \neq 0$. This ensures that there is an element x in L with $\deg(x) = 1$. In connection with a positive monoid $M \subset L$ we will only use gradings that are positive on M, i.e. $\deg(M) \subset \mathbb{Z}_+$. Such a grading exists if and only if M is positive.

The connection to commutative ring theory is given by the following algebraic object:

Definition 1.5. Let M be a commutative monoid. Its *monoid algebra* over the commutative ring R is the free R-module with basis X^m, $m \in M$,

$$R[M] = \left\{ \sum_{i=1}^{k} r_i X^{m_i} : r_i \in R, m_i \in M \right\},$$

with the multiplication $X^m \cdot X^{m'} = X^{m+m'}$.

In this perspective we will use algebra terminology of $R[M]$ also for the monoid M, for example in considering M as a N-module for a submonoid $N \subset M$. The monoid of lattice points of a cone can be interpreted as the integral closure of a monoid, or as the set of exponents of the monomials in the integral closure of a monoid algebra.

1.3. Linear transformation

The input for the algorithms will be a cone $C \subset \mathbb{R}^d$ and a lattice $L \subset \mathbb{Z}^d$. The main object of interest will be the affine monoid $M = C \cap L$. First of all we may assume that C and L generate the same linear space, otherwise we restrict both to the intersection vector space $\operatorname{span}(C) \cap \operatorname{span}(L)$. Now a very useful preliminary step is to apply a linear transformation $\varphi : \mathbb{R}^d \mapsto \mathbb{R}^r$, such that

(i) $r = \dim C$ and φ is a bijection between C and $C' = \varphi(C)$;
(ii) $\varphi(L) = \mathbb{Z}^r$.

Such a transformation can be found with the help of the elementary divisor algorithm, see [**10**].

In the following, we will always assume that C has maximal dimension and the lattice is $L = \mathbb{Z}^d$. This is not relevant for the theory, but simplifies notations, the algorithms and not least also the computations.

CHAPTER 2

Building the cone

This chapter describes the fundamental algorithms that build the cone generator by generator. The first two sections describe the computation of the support hyperplanes. In this way a generator based description of the cone is turned into a description by elements in the dual space. The same algorithm can be used in the other direction by interpreting the linear forms of the support hyperplanes as generators in the dual space. Therefore this process is also called *dualizing.*

In Section 2.3 we decompose the cone into simplicial cones via a triangulation. For larger examples we introduce in Section 2.4 the decomposition into pyramids. The last section presents how to get a disjoint union out of the triangulation. These methods lay the foundation for the algorithms in the following chapters.

We will always first apply a linear transformation as described in Section 1.3 and therefore assume the cone is full dimensional. For the algorithms in this chapter the rationality of the cone is in principle not relevant. The only exceptions are the modifications that allow us to use integer arithmetic.

2.1. Support hyperplanes of a simplicial cone

The computation of the support hyperplanes for a simplicial cone is easy, we just have to invert the matrix of generators. Let the simplicial cone $C \subset \mathbb{R}^d$ be generated by linearly independent vectors $x_1, \dots, x_d \in \mathbb{Z}^d$ and let G be the matrix with rows $x_1, \dots, x_d$. Then the columns of $G^{-1} = (l_{i,j})$ give linear forms $\lambda_j(v_1, \dots, v_d) = l_{1,j}v_1 + \cdots + l_{d,j}v_d$ which evaluate to 1 for the x_j and to 0 for the $d-1$ other generators. Therefore they exactly describe the support hyperplanes $\mathcal{H}(C)$. The entires $l_{i,j}$ of G^{-1} are in $\mathbb{Q}$. After multiplying by a common denominator, e.g. $\det(G)$, we get coefficients in $\mathbb{Z}$.

The computations implemented in Normaliz avoid calculations with fractions. Extending fractions, checking for greatest common divisors, and similar operations are costly. For that reason we stay in $\mathbb{Z}$ and use the information on the problem that we want to solve in order to find a suitable common denominator. This has a strong impact on the computation speed. Usually we can manage to multiply only in one step of the computation by such a suitable factor and may divide at the very end. We will demonstrate it in the case of inverting an integer matrix.

Suppose we want to invert the matrix G. An upper triangular matrix T with positive diagonal elements and the accompanying matrix X such that $T = XG$ can be computed from G and the d-dimensional identity matrix E_d by using elementary row operations over $\mathbb{Z}$ which do not change the absolute value of the determinant:

(i) exchange two rows,
(ii) multiply a row by -1, and
(iii) add an integral multiple of a row to another row.

Starting with $(p, q) = (1, 1)$ we use the following strategy to produce a triangular matrix below and left from that position. If for $p \le k \le n$ all entires $G_{k,q}$ are zero, increase q by one and repeat (this cannot happen when G has full rank). Otherwise, choose k such that $G_{k,q}$ has the smallest absolute value but is not zero, and exchange rows p and k. For i from p to n subtract a_i-times row p from row i, where a_i is the next integer to $G_{i,q}/G_{p,q}$ which is smaller in absolute value. This ensures that for $p < i \le n$ all $G_{i,q}$ are in absolute value smaller than $G_{p,q}$. As long one of the $G_{i,q}$ is non-zero repeat the determination of k and the row subtractions. When we have obtained zeros below $G_{k,q}$, increase p and q by one and start the complete process again until the matrix is in an upper triangular form. Finally, we can multiply rows by -1 to get nonnegative diagonal entries.

Let $(t_1, \ldots, t_d)$ be the diagonal of T and $\delta = t_1 \ldots t_d$. In the classical Gaussian elimination we now would have to divide by t_i. To stay in $\mathbb{Z}$ we multiply T and X with δ and start to clean out the elements above the diagonal in the following way. For row i from d to 1 we divide row i by t_i, now the entry (i, i) is δ and all entires above are divisible by δ, so we can add an integral multiple of row i to the rows above to create zeros there. In this way we get the matrices $T' = \delta \cdot E_d$ and X' such that $T' = X'G$ or phrased differently $X' = \delta \cdot G^{-1}$.

Example 2.1. For the simplicial cone with generators $x_1 = (4, 1, 5)$, $x_2 = (6, 2, 3)$, $x_3 = (2, 1, 3)$ we compute the support hyperplanes.

$$\begin{array}{rrr|rrr}
4 & 1 & 5 & 1 & 0 & 0 \\
6 & 2 & 3 & 0 & 1 & 0 \\
2 & 1 & 3 & 0 & 0 & 1 \\
\hline
2 & 1 & 3 & 0 & 0 & 1 \\
0 & -1 & -1 & 1 & 0 & -2 \\
0 & -1 & -6 & 0 & 1 & -3 \\
\hline
2 & 1 & 3 & 0 & 0 & 1 \\
0 & -1 & -1 & 1 & 0 & -2 \\
0 & 0 & -5 & -1 & 1 & -1 \\
\hline
2 & 1 & 3 & 0 & 0 & 1 \\
0 & 1 & 1 & -1 & 0 & 2 \\
0 & 0 & 5 & 1 & -1 & 1
\end{array}
\qquad
T = \begin{pmatrix} 2 & 1 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 5 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 2 \\ 1 & -1 & 1 \end{pmatrix}$$

Now we multiply by $\delta = 10$ and continue.

$$\begin{array}{rrr|rrr}
20 & 10 & 30 & 0 & 0 & 10 \\
0 & 10 & 10 & -10 & 0 & 20 \\
0 & 0 & 50 & 10 & -10 & 10 \\
\hline
20 & 10 & 0 & -6 & 6 & 4 \\
0 & 10 & 0 & -12 & 2 & 18 \\
0 & 0 & 10 & 2 & -2 & 2 \\
\hline
20 & 0 & 0 & 6 & 4 & -14 \\
0 & 10 & 0 & -12 & 2 & 18 \\
0 & 0 & 10 & 2 & -2 & 2 \\
\hline
10 & 0 & 0 & 3 & 2 & -7 \\
0 & 10 & 0 & -12 & 2 & 18 \\
0 & 0 & 10 & 2 & -2 & 2
\end{array}
\qquad
X' = 10 \cdot G^{-1} = \begin{pmatrix} 3 & 2 & -7 \\ -12 & 2 & 18 \\ 2 & -2 & 2 \end{pmatrix}$$

The support hyperplanes are given by $(3, -12, 2)$, $(1, 1, -1)$ and $(-7, 18, 2) \in (\mathbb{R}^3)^*$.

2.2. Fourier-Motzkin elimination

Now we come to non-simplicial cones. Here there are different methods to compute the support hyperplanes. Our approach bases on the *Fourier-Motzkin elimination*, see for example [**4**, pp. 11,12].

Theorem 2.2. *Let $C = \operatorname{cone}(x_1, \dots, x_n)$ and $C' = \operatorname{cone}(x_1, \dots, x_{n-1})$. Furthermore let $\lambda_1, \dots, \lambda_s$ be linear forms that cut out C', i.e. $C' = H^+_{\lambda_1} \cap \dots \cap H^+_{\lambda_s}$. With $P = \{\lambda_i : \lambda_i(x_n) > 0\}$, $Z = \{\lambda_i : \lambda_i(x_n) = 0\}$, $N = \{\lambda_i : \lambda_i(x_n) < 0\}$ the linear forms in*

$$P \cup Z \cup \{\mu_{i,j} = \lambda_i(x_n)\lambda_j - \lambda_j(x_n)\lambda_i \;:\; \lambda_i \in P, \lambda_j \in N\}$$

cut out C.

Obviously $P \cup Z$ is exactly the set of old hyperplanes that remain valid for the new cone. The theorem ensures that together with the $\mu_{i,j}$ they cut out the new cone C, but it does not have to be the minimal set of support hyperplanes $\mathcal{H}(C)$. In the algorithm we want to maintain only this minimal set, so now assume that we start with such a set $\mathcal{H}(C')$ for C' and want to find $\mathcal{H}(C)$.

For that purpose let us analyze which generators lie in a hyperplane $\mu_{i,j}$. The hyperplane is constructed in such a way that $\mu_{i,j}(x_n) = 0$, and for $k = 1, \dots, n-1$ is $\mu_{i,j}(x_k) = 0$ if and only if $\lambda_i(x_k) = 0$ and $\lambda_j(x_k) = 0$. In other words, the face of C defined by $\mu_{i,j}$ is generated by x_n and $\{x_k : x_k \in H_{\lambda_i} \cap H_{\lambda_j}, k = 1, \dots, n-1\}$. This face is a facet of C if and only if $C \cap H_{\lambda_i} \cap H_{\lambda_j}$ is a subfacet, i.e. has dimension $d-2$. A necessary condition for this is that there are at least $d-2$ generators in $H_{\lambda_i} \cap H_{\lambda_j}$. If one of the facets contains exactly $d-1$ generators this condition is also sufficient. So to get only the relevant support hyperplanes we need to find the subfacets which belong to a positive and to a negative hyperplane. The area of C where positive and negative hyperplanes intersect can geometrically be interpreted as the boundary of the part of C that is visible from x_n.

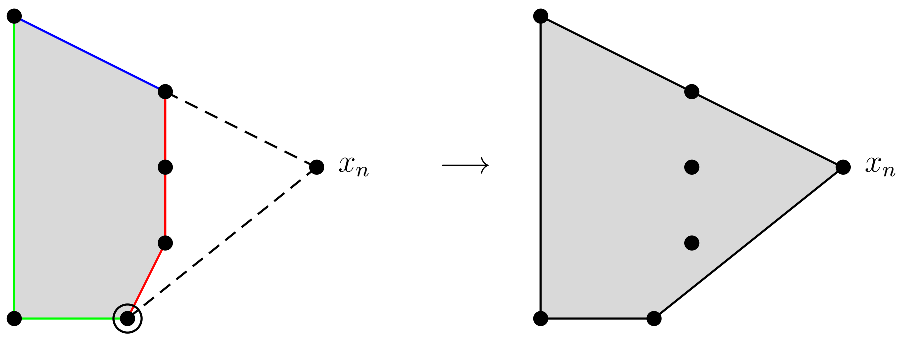


FIGURE 1. One step in the Fourier-Motzkin elimination in the cross section of a 3-dimensional cone.

Example 2.3. In Figure 1 a Fourier-Motzkin step in the cross section of a 3-dimensional cone is visualized. The gray shaded area is the previous cone C' and now x_n is added as cone generator. The positive facets from $\mathcal{H}^{>}(C', x_n)$ are green, the neutral facets from $\mathcal{H}^{=}(C', x_n)$ are blue and the negative facets from $\mathcal{H}^{<}(C', x_n)$ are red. In this case there is only one pair of negative and positive facets which intersect and give rise to a new facet hyperplane.

Here we of course have a very simple example, but in this dimension it will not get much more complicated. In 3-dimensional cones two facets intersect either in $\{0\}$ or in a subfacet, since subfacets are already of dimension 1. And there are at most two pairs of negative and positive hyperplanes where the intersection defines a subfacet. But in higher dimensions this changes dramatically.

The function FINDNEWFACETS in Algorithm 1 computes one Fourier-Motzkin step. The negative and positive hyperplanes are paired and it is checked whether their intersections are subfacets. This is determined by INTERSECTINSUBFACET where two different criteria are used. For a large number of hyperplanes it is done by a rank computation for the matrix whose rows are the generators of C' in the intersection. If the number of hyperplanes is small, it is faster to see if there is a third hyperplane η which contains the intersection of λ and ν. This is the case if and only if the intersection has a dimension lower than $d-2$.

In Normaliz simplicial facets are handled separately. This was especially important in earlier versions, where a shelling was used to compute the Hilbert series. In this case most of the facets are simplicial, see [**9**].

As recognizable from the algorithm, the information which generators lie in a hyperplane is used multiple times. This can be computed by taking the scalar product of the generator with the hyperplane linear form. For an efficient implementation it is important to keep this information and not recompute it on every use. In Normaliz every support hyperplane λ is accompanied with the bitvector of n bits whose entry on position i is 1 exactly if $\lambda(x_i) = 0$.

Algorithm 1 Support hyperplane computation

```
Require: $\mathcal{H}(C')$ for a cone $C' = \operatorname{cone}(x_1, \dots, x_{n-1}) \subset \mathbb{R}^d$, $x_n \in \mathbb{R}^d$, $X = \{x_1, \dots, x_{n-1}\}$
Return: $\mathcal{H}(C)$ for the cone $C = \operatorname{cone}(x_1, \dots, x_n)$
 1: function findNewFacets($\mathcal{H}(C')$, $X$, $x$)
 2:     $\mathcal{H}(C) \leftarrow \mathcal{H}^{>}(C', x) \cup \mathcal{H}^{=}(C', x)$
 3:     for $H_\nu \in \mathcal{H}^{<}(C', x)$ do
 4:         for $H_\lambda \in \mathcal{H}^{>}(C', x)$ do
 5:             if intersectInSubfacet($\lambda$, $\nu$, $X$, $\mathcal{H}(C')$) then
 6:                 $\mu \leftarrow$ makeCoprime($\lambda(x)\nu - \nu(x)\lambda$)
 7:                 $\mathcal{H}(C) \leftarrow \mathcal{H}(C) \cup \{H_\mu\}$
 8:     return $\mathcal{H}(C)$

Require: linear forms $\lambda, \nu \in (\mathbb{R}^d)^*$, $X$, $\mathcal{H}(C')$ as above
Return: true if $\dim(C \cap H_\lambda \cap H_\nu) = d - 2$, false otherwise
 9: function intersectInSubfacet($\lambda$, $\nu$, $X$, $\mathcal{H}(C')$)
10:     $I \leftarrow X \cap H_\lambda \cap H_\nu$                          ▷ generators in $H_\lambda \cap H_\nu$
11:     if $|I| < d - 2$ then
12:         return false
13:     if $|X \cap H_\lambda| = d - 1$ or $|X \cap H_\nu| = d - 1$ then    ▷ one facet is simplicial
14:         return true
15:     if $3 \cdot |\mathcal{H}(C')| > d^2 \cdot |I|$ then
16:         return rank(I) $= d - 2$
17:     else
18:         for $H_\eta \in \mathcal{H}(C') \setminus \{H_\lambda, H_\nu\}$ do
19:             if $I \subset H_\eta$ then
20:                 return false
21:         return true
```

We will see in the next section that this support hyperplane algorithm works very well together with our triangulation algorithm.

2.3. Triangulation

Many problems are easier to solve in the simplicial case, or it is not even clear how to solve them directly for non-simplicial cones. The idea is to split a non-simplicial cone into many smaller simplicial cones, solve the problem for every simplicial cone and put the results together to a result for the complete cone. We will now make this idea more precise.

Definition 2.4. Let C be a rational cone in $\mathbb{R}^d$. A *subdivision* Γ of C is a finite set of cones in $\mathbb{R}^d$ such that

(i) $\sigma' \in \Gamma$ for all faces σ' of $\sigma \in \Gamma$,

(ii) $\sigma_1 \cap \sigma_2$ is a face of σ_1 and σ_2 for all $\sigma_1, \sigma_2 \in \Gamma$, and
(iii) $C = \bigcup_{\sigma \in \Gamma} \sigma$.

A *triangulation* is a subdivision where all cones are simplicial.

The *lexicographical* (or *placing*) triangulation $\Lambda(x_1, \dots, x_n)$ of $\operatorname{cone}(x_1, \dots, x_n)$ is build recursively by the following rules:

(i) The zero cone has the trivial triangulation $\Lambda()$ consisting of the zero cone itself.
(ii) When $\Lambda(x_1, \dots, x_{i-1})$ is the lexicographical triangulation of the cone $C = \operatorname{cone}(x_1, \dots, x_{i-1})$ then $\Lambda(x_1, \dots, x_i)$ is defined by

$$\Lambda(x_1, \dots, x_{i-1}) \cup \{\operatorname{cone}(\sigma, x_i) \ : \ \sigma \in \Lambda(x_1, \dots, x_{i-1}),\ \sigma \text{ is visible from } x_i\}.$$

We will now present an algorithm to compute this lexicographical triangulation together with the support hyperplanes as described in Section 2.2. In computations subdivisions are represented by a list of the maximal cones. They contain all the relevant information as you get the complete subdivision set by adding all faces of the maximal cones. These maximal cones will always have the same dimension as the cone C.

Our algorithm to compute a triangulation will start with d linearly independent generators and then follow the definition of the lexicographical triangulation. For a simpler description of the algorithm we will always assume that the first d generators $x_1, \dots, x_d$ are linearly independent. If this is not the case we will take the lexicographical first set of d linearly independent vectors to the front. This reordering will be hidden here to keep the presentation clear.

So we start with the simplicial cone $C_d = \operatorname{cone}(x_1, \dots, x_d)$. Then we add one generator after another and for each $C_i = \operatorname{cone}(x_1, \dots, x_i)$ we extend the triangulation to Γ_i and compute the support hyperplanes.

To extend Γ_{i-1} to the new triangulation Γ_i we use the restriction of Γ_{i-1} to the part of C_{i-1} that is visible from x_i. We get it by going over all visible support hyperplanes of C_i. If the hyperplane contains only $d-1$ generators of C_i this hyperplane defines a simplicial facet and we add the cone generated by this facet and x_i to the triangulation. If not we select those $\sigma \in \Gamma_{i-1}$ that have $d-1$ generators in the hyperplane and add the cones over this facets to the triangulation.

Example 2.5. In Figure 2 the process of extending the triangulation of a 4-dimensional cone is visualized. Imagine that x_1, x_3, x_4, x_6 lie in one plane and only x_2 and x_5 are elevated in such a way that x_2, x_3, x_4, x_5 lie in another plane. Then the only facets of $C' = \operatorname{cone}(x_1, \dots, x_5)$ which are visible from x_6 are $F_1 = \operatorname{cone}(x_1, x_2, x_3)$ and $F_2 = (x_2, x_3, x_4, x_5)$. So far the lexicographical triangulation is

$$\Gamma' = \{\sigma_1 = \operatorname{cone}(x_1, x_2, x_3, x_4), \sigma_2 = \operatorname{cone}(x_1, x_2, x_4, x_5)\}.$$

The extension procedure forms directly $\sigma_3 = \operatorname{cone}(F_1, x_6)$ from the simplicial facet F_1 and adds it to the triangulation. But for F_2 a triangulation of the facet has to be found first by matching F_2 with the existing lexicographical triangulation. The result of the matchings are the two additional simplicial cones $\sigma_4 = \mathrm{cone}(F_2 \cap \sigma_1, x_6) = \mathrm{cone}(x_2, x_3, x_4, x_6)$ and $\sigma_5 = \mathrm{cone}(F_2 \cap \sigma_2, x_6) = \mathrm{cone}(x_2, x_4, x_5, x_6)$.

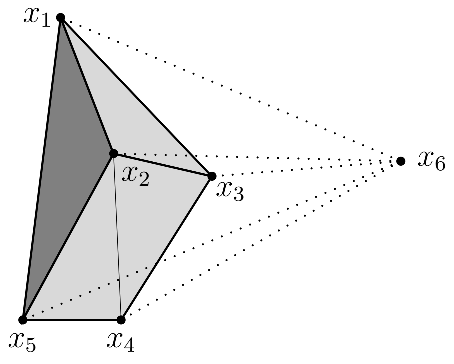


FIGURE 2. Extension of the triangulation in the cross-section of a 4-dimensional cone.

Algorithm 2 Compute a triangulation

```
Require: $\{x_1, \dots, x_n\} \subset \mathbb{Z}^d$, we assume that $x_1, \dots, x_d$ are linearly independent
Return: the support hyperplanes $\mathcal{H}$ and the lexicographical triangulation $\Gamma$
 1: function BUILDCONE($\{x_1, \dots, x_n\}$)
 2:     $\mathcal{H} \leftarrow$ invert $(x_1, \dots, x_d)$ over $\mathbb{Z}$
 3:     $\Gamma \leftarrow \{\mathrm{cone}(x_1, \dots, x_d)\}$
 4:     for $i \leftarrow d + 1$ to $n$ do
 5:         $\Gamma \leftarrow$ EXTENDTRIANGULATION($\Gamma$, $\{x_1, \dots, x_{i-1}\}$, $\mathcal{H}$, $x_i$)
 6:         $\mathcal{H} \leftarrow$ FINDNEWFACETS($\mathcal{H}$, $\{x_1, \dots, x_{i-1}\}$, $x_i$)      ▷ see Alg. 1
 7:     return $\mathcal{H}, \Gamma$

Require: a cone generated by $X$ with support hyperplanes $\mathcal{H}$ and triangulation $\Gamma$
Return: triangulation of cone$(X, x)$
 8: function EXTENDTRIANGULATION($\Gamma$, $\mathcal{H}$, $X$, $x$)
 9:     for $H_\lambda \in \mathcal{H}$ do
10:         if $\lambda(x) < 0$ then                                  ▷ visibility check
11:             $F \leftarrow \{y \in X \ : \ \lambda(y) = 0\}$
12:             if $|F| = d - 1$ then                                 ▷ simplicial case
13:                 $\Gamma \leftarrow \Gamma \cup \{\mathrm{cone}(F, x)\}$
14:             else
15:                 for $\sigma \in \Gamma$ do
16:                     if $|\sigma \cap F| = d - 1$ then
17:                         $\Gamma \leftarrow \Gamma \cup \{\mathrm{cone}(\sigma \cap F, x)\}$
18:     return $\Gamma$
```

Algorithm 2 describes this procedure in the way it was used in Normaliz version 2.2. The most time-consuming part of this algorithm is the matching of the non-simplicial facets with the triangulation. Here we added an improvement to exclude parts of the triangulation that cannot be relevant for a given facet.

For this purpose we divide the triangulation into sections $\Gamma(i) = \Gamma_i \setminus \Gamma_{i-1}$, so every section $\Gamma(i)$ contains the simplicial cones which were added in the EXTEND-TRIANGULATION phase of generator x_i. Now we examine which support hyperplanes of C_k can contain a facet of $\sigma \in \Gamma(k)$. The facet of σ which does not include x_k cannot be on the boundary. (Except when σ is the start simplicial cone, what we will discuss later.) Every other facet F has x_k as a generator, and there are two possibilities for the support hyperplane H containing F:

(i) H is a new hyperplane of C_k, i.e. $H \in \mathcal{H}(C_k) \setminus \mathcal{H}(C_{k-1})$, or
(ii) H was a neutral hyperplane in step k, i.e. $H \in \mathcal{H}^{=}(C_{k-1}, x_k)$

In both cases x_k and at least $d-2$ generators out of $x_1, \dots, x_{k-1}$ have to lie in H.

Every hyperplane H' which is created in a later step i of the algorithm cannot contain a $(d-1)$-dimensional facet of σ, since the intersection $H' \cap C_{i-1}$ is a $(d-2)$-dimensional subfacet.

Now let $H \in \mathcal{H}^{<}(C_{i-1})$ be a negative hyperplane in the triangulation extension step i, and let $x_{j_1}, \dots, x_{j_m}$ be the generators in H. Then the loop on line 15 in Algorithm 2 only needs to run over the simplicial cones in $\Gamma(j_{d-1}), \dots, \Gamma(j_m)$.

For the start simplicial cone σ_s the argumentation does not hold completely. Every facet of σ_s is on the boundary of C_d. To include σ_s as matching partner of a hyperplane containing $x_1, \dots, x_{d-1}$, we define that the start simplicial cone belongs to $\Gamma(d)$ and to $\Gamma(d-1)$.

2.4. Pyramid decomposition

The presented algorithms for the computation of support hyperplanes and the lexicographical triangulating work well for examples up to a certain complexity. With the continuous improvements of the algorithms and also of the hardware computing power, it was possible to attack increasingly large examples. The complexity of a step in those algorithms depends in a non-linear way on the size of the data. For the Fourier-Motzkin elimination it is basically given by the number of negative hyperplanes times the number of positive hyperplanes. An extension step of the triangulation depends on the number of negative (non-simplicial) hyperplanes times the size of the triangulation.

We have to match a negative hyperplane with positive hyperplanes or simplicial cones in the triangulation. With growing numbers of matching partners, it became more and more important to take a preselection of the partners. One attempt in this direction is the sectioning of the triangulation as described in the previous section. A more consequent approach to "localize" data are divide and conquer algorithms. Here a big problem is divided into multiple smaller problems of the same kind which are easier to solve. A classical divide and conquer approach for cones is not easy to implement because dividing a rational cone into pieces in a good way is difficult. One could cut the cone by some hyperplane into two parts. But then we have to compute the intersection of the hyperplane with the cone to get the additional cone generators and it is not even clear if the separate cones are easier to handle. If one takes cones generated by subsets of the generators, they have to be chosen in such a way that their union is the original cone. Additionally they should not intersect in parts that are too large because this would mean to process these parts multiple times.

We will use a decomposition into a special kind of subcones which arise naturally in course of our triangulation algorithm.

Definition 2.6. Let $C' = \operatorname{cone}(x_1, \dots, x_{i-1})$ be a cone in $\mathbb{R}^d$ and $x_i \in \mathbb{Z}^d$. Consider a negative hyperplane $H \in \mathcal{H}^<(C', x_i)$ and the corresponding facet $F = H \cap C_{i-1}$. Then we call $P = \operatorname{cone}(F, x_i)$ the *pyramid* over the *base facet* F with *apex* x_i.

So far the triangulation for a pyramid P was obtained by first restricting the existing triangulation to the base facet F of the pyramid. This gives a triangulation $\Gamma_F = \{\sigma_1, \dots, \sigma_m\}$ of F. Note that the σ_j are simplicial cones of dimension $d - 1$. Then the simplicial cones $\operatorname{cone}(\sigma_j, x_i)$ for $j = 1, \dots, m$ form a triangulation of the pyramid P.

The general advantage and also disadvantage of the pyramid decomposition is that we handle the pyramid independent of the rest of the cone. In this way we get a more local data set of support hyperplanes and triangulation which makes the algorithms faster. On the other hand, computations are redone or are only needed to process a pyramid. For example, a support hyperplane of the pyramid may also belong to another pyramid and therefore gets computed twice. But when the amount of data in the original cone becomes large it is faster to compute information from scratch in the local setting of the pyramid, instead of trying to find it in the large data set.

Besides the advantage in the computation times, the pyramid decomposition has additional benefits. One of them is the following: once we switch to pyramid decomposition to build the triangulation, the existing triangulation is not needed for this purpose anymore. Therefore it can be evaluated with the methods from the Chapters 3 and 4 and can be deleted afterwards. This allows to limit the memory used by the triangulation to a fixed value, since it is not necessary to store the complete triangulation until the end of the computation.

A second important positive effect is that the pyramids are completely independent from the cone which created them and also independent from each other. Hence they can also be stored and BUILDCONE can be called at any time when it is convenient.

In the hyperplane computation a pyramid decomposition can be useful as well. In contrast to the use of pyramids to extend the triangulation we need the hyperplanes in the next step, so their evaluation cannot be delayed but instead has to be done directly in a recursive manner. Therefore we call them *recursive pyramids.*

Algorithm 3 Compute support hyperplanes and triangulation with pyramid decomposition

```
Require: $\{x_1,\dots,x_n\} \subset \mathbb{Z}^d$, we assume that $x_1,\dots,x_d$ are linearly independent;
    Booleans doHyp and doTri which indicate whether the support hyperplanes and
    the triangulation of $C = \operatorname{cone}(x_1,\dots,x_n)$ should be computed
Return: the support hyperplanes $\mathcal{H}$ and the lexicographical triangulation $\Gamma$ of $C$
 1: function BUILDCONE($\{x_1,\dots,x_n\}$, doHyp, doTri)
 2:     $\mathcal{H} \leftarrow$ invert $(x_1,\dots,x_d)$ over $\mathbb{Z}$
 3:     if doTri then
 4:         $\Gamma \leftarrow \{\operatorname{cone}(x_1,\dots,x_d)\}$
 5:     for $i \leftarrow d+1$ to $n$ do
 6:         doPyrHyp $\leftarrow$ (doHyp or $i \neq n$) and $|\mathcal{H}^>| >$ BoundPyrHyp
 7:         doPyr $\leftarrow$ doPyrHyp or (doTri and $|\Gamma| >$ BoundPyrTri)
 8:         if doPyr then
 9:             for $H_\lambda \in \mathcal{H}^<$ do
10:                 $X \leftarrow (\{x_1,\dots,x_{i-1}\} \cap H_\lambda) \cup \{x_i\}$        ▷ pyramid generators
11:                 isLarge $\leftarrow$ doPyrHyp and $|X| > 2 \cdot d$
12:                 if isLarge then
13:                     $\mathcal{H}'', \Gamma' \leftarrow$ BUILDCONE($X$, false, doTri)
14:                     $\mathcal{H}' \leftarrow \mathcal{H}' \cup$ FINDNEWFACETS($\mathcal{H}^>(x_i), H_\lambda, \{x_1,\dots,x_{i-1}\}, x_i$)
15:                 else
16:                     $\mathcal{H}(P), \Gamma' \leftarrow$ BUILDCONE($X$, doPyrHyp, doTri)
17:                     $\mathcal{H}' \leftarrow \mathcal{H}' \cup \{H_\mu \in \mathcal{H}(P) : \mu(x_i) = 0$ and $\mu(y) > 0\ \forall y \in X \setminus \{x_i\}\}$
18:                 $\Gamma \leftarrow \Gamma \cup \Gamma'$
19:             if doPyrHyp then
20:                 $\mathcal{H} \leftarrow \mathcal{H}^>(x_i) \cup \mathcal{H}^=(x_i) \cup \mathcal{H}'$
21:         else if doTri then
22:             $\Gamma \leftarrow \Gamma \cup$ EXTENDTRIANGULATION($\Gamma$, $\mathcal{H}$, $\{x_1,\dots,x_{i-1}\}$, $x_i$)
23:         if (doHyp or $i \neq n$) and not doPyrHyp then
24:             $\mathcal{H} \leftarrow$ FINDNEWFACETS($\mathcal{H}$, $\{x_1,\dots,x_{i-1}\}$, $x_i$)
25:     if not doHyp then
26:         $\mathcal{H} \leftarrow \emptyset$
27:     return $\mathcal{H}, \Gamma$
```

We let the decision whether to use recursive pyramids depend on the number of positive support hyperplanes. If this number $|\mathcal{H}^>|$ exceeds the bound BoundPyrHyp, for every negative hyperplane H it is determined in which way to compute the new support hyperplanes that originate from H. The possibilities are:

(i) match that negative hyperplane H with the positive hyperplanes in a Fourier-Motzkin step, or
(ii) make a recursive pyramid.

For simplicial facets it is faster to compute all hyperplanes of its pyramid, which is a simplicial cone, directly by inverting, see Section 2.1. Similarly, for nearly simplicial facets, i.e. those with not many more than d generators, it is best to use a recursive call of BUILDCONE.

For larger pyramids, i.e. when the base facet has many generators, the recursive call gets expensive and would probably include further recursive calls down to a high level of recursion. This happens if there are many generators in a face of the cone of a relatively small dimension. In this case the hyperplane structure that comes from this small dimensional face is recomputed a lot of times. To avoid this, the large pyramids are not handled recursively, but by method (i) above.

In recursive pyramids the triangulation is computed together with the support hyperplanes if necessary. In this way recursive pyramids do not have to be considered again for the triangulation, and their support hyperplanes do not have to be computed twice. If in this process pyramids are created which are only needed for the triangulation, they are stored and processed later.

In line 17 the support hyperplanes $\mathcal{H}(P)$ of the pyramid P are inspected and only those are taken which are additional support hyperplanes of the new cone $C_i = \operatorname{cone}(x_1, \dots, x_i)$. Among $\mathcal{H}(P)$ the only hyperplane which does not contain x_i is $H_{-\lambda}$ and it cannot be a support hyperplane of C_i. Now let μ be the linear form of one of the remaining hyperplanes, and let y a generator of C_{i-1} but not of P. Obviously μ cannot be a hyperplane of C_i if $\mu(y) < 0$. In the case $\mu(y) = 0$, a subfacet of $P \cap C_{i-1}$ and also y lie in H_μ. We can conclude that H_μ is already in $\mathcal{H}^{=}(C_{i-1}, x_i)$. Hence we have to check if the linear form gives a strictly positive value on all these generators y.

The call of FINDNEWFACETS on line 24 is to the function of Algorithm 1, whereas on line 14 a slightly modified version is called, which does the Fourier-Motzkin elimination step for just a single negative hyperplane.

As values for the bounds we have experimentally determined BoundPyrTri = 20,000 and BoundPyrHyp = $500 \cdot d$ to give good a performance on a wide range of examples.

As already mentioned, in Normaliz the non-recursive pyramids are not directly evaluated, but stored in a buffer. The buffer has multiple levels. The pyramids from the cone we start with are stored on level 0. In course of the evaluation a pyramid of level i may create pyramids itself. Those pyramids are stored on level $i + 1$. To keep the memory usage of the algorithms small, the number of pyramids in one level of the buffer is limited to 200.000. When the limit is reached, we interrupt the current computation, evaluate the pyramids on the next level, and then return to the previous level. The main advantage of this "store and evaluate later" principle is that the evaluation of the pyramids is parallelizable. Their independence leads to a highly efficient parallelization.

Table 2.1 compares computation times with and without the use of pyramids and demonstrate the superiority for large examples. The examples A443, A543

	max. number	time support hyperplanes	
	of hyperplanes	with pyramids	without pyramids
A553	$317,387$	118 sec	672 sec
lo7 face of cd 4	$121,996$	172 sec	264 sec
lo7 face of cd 3	$803,839$	24 min	324 min

	size of	time triangulation	
	triangulation	with pyramids	without pyramids
A443	$2,654,272$	6 sec	49 sec
A543	$102,538,980$	1 min	354 min
lo6 facet	$163,674,912$	1 min	162 min
lo6	$5,745,903,354$	39 min	—

Table 2.1. Pyramid decomposition timing comparisons (on x4450 with 20 threads)

and A553 are monoids of marginal distributions of contingency tables, see Section 5.1. The other examples are related to the linear order polytopes connected to the inversion model of statistical ranking for $n = 6$ and $n = 7$, see [**38**], we use faces of codimension 4 and 3 of lo7, a facet of lo6 and lo6 itself. For even larger examples the gap between the computation times increases further. The complete triangulation without pyramids for lo6 was not even possible, because it exceeded the available memory.

Note that the triangulation produced by Algorithm 3 is the lexicographical triangulation, although we allow reordering of the generators to achieve linear independence of $x_1, \dots, x_d$ and use pyramids to extend the triangulation. See [**8**] for a detailed discussion.

2.5. Disjoint decomposition

The triangulation is a very important tool in the algorithms to decompose a problem into simplicial cases. This decomposition is not disjoint. Although this is not always a problem, sometimes it is useful (see Sections 3.5 and 3.6), for the Hilbert series computation it is even necessary (see Chapter 4), to get a disjoint decomposition.

The maximal simplicial cones in the triangulation intersect in facets and lower dimensional faces. Facets lie in exactly two simplicial cones or they are external facets of the cone and therefore in just one simplicial cone. But faces of lower dimension may belong to many simplicial cones. If we want a disjoint decomposition, we have to make sure we do not count these parts multiple times.

In the history of Normaliz different algorithms have been used to compute the Hilbert series where the disjoint decomposition is of essential importance. All of them base on a triangulation of C. One helpful fact is that C is the disjoint union of the relative interiors of all (not only the maximal) simplicial cones in the triangulation. The first versions of Normaliz used this fact and decided for each maximal simplicial cone for which faces of it the interior should be considered, see [**10**]. In this way a disjoint decomposition is achieved. Later the complexity could be considerably reduced by the use of a shelling. A shelling is an order $\sigma_1, \dots, \sigma_m$ of the simplicial cones where $\sigma_i \cap (\sigma_1 \cup \dots \cup \sigma_{i-1})$ is a union of facets of σ_i. So the problem is reduced to excluding facets (and not arbitrary faces) of the previous simplicial cones. The computation of a shelling was done by lifting the cone in a proper way into $\mathbb{R}^{d+1}$, triangulate it and then find a line shelling, see [**9**] for more details.

Both algorithms have a central weak point for big examples: To exclude faces they need to compare simplicial cones in the triangulation. This gives rise to two problems. The comparison gets very time consuming since the complexity is quadratic in the size of the triangulation. Secondly, for a big triangulation it is even impossible to store it in the memory.

Köppe and Verdoolaege showed that this inclusion-exclusion problem could be solved in a simple way without comparisons of the simplicial cones [**27**, Thm. 3]. We state it here in an adapted form from [**8**, Lemma 10].

Lemma 2.7. *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.*

Definition 2.8. We will call a vector O_C as in Lemma 2.7 *order vector*.

The idea of this criterion can be described as follows. For every simplicial cone we exclude the region that is visible from O_C. Since O_C does not lie on any of the hyperplanes, this region is a union of facets. In this way a point $p \in C$ is considered in the unique simplicial cone σ of the triangulation for which an $\varepsilon > 0$ exists such that the part $[p, O_C]_\varepsilon = \{p + \alpha(O_C - p) : \ 0 \le \alpha \le \varepsilon\}$ of the line segment from p to O_C is a subset of σ.

In our algorithm we will choose the order vector O_C to be some integral point in the interior of the first simplicial cone in the triangulation. Of course it may be contained in the support hyperplane of another simplicial cone. The *simulation of simplicity* principle from computational geometry allows us to work with an infinitesimal perturbation O'_C of O_C without even computing the perturbation explicitly, see [**21**, section 9.4].

We choose a perturbation O'_C that behaves like O_C for all hyperplanes which do not contain O_C, i.e. $O'_C \in H^*$ if $O_C \in H^*$ for $* \in \{<, >\}$. For $O_C \in H$ check the first non-zero coordinate of the linear form of H. If it is positive, we have $O'_C \in H^>$ and otherwise $O'_C \in H^<$.

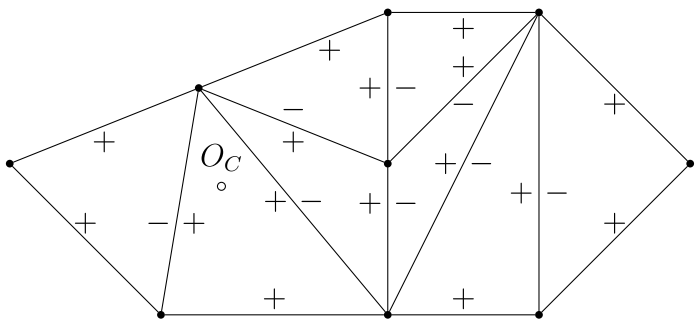


FIGURE 3. Using the order vector

Figure 3 depicts a triangulation with an order vector O_C. For every simplicial cone its facets are marked with "−" if the facet is excluded and with "+" if it is included in that simplicial cone.

For our later use, it is important to observe the following.

Remark 2.9. For a pointed cone C and O_C in the interior of C we never exclude all facets of a simplicial cone σ in the triangulation of C. Suppose O_C is in the negative halfspace for all support hyperplanes of σ. Then $-O_C$ is in the positive halfspace for all of them and $-O_C$ lies in $\sigma \subset C$. This is a contradiction to our premise that C is pointed.

CHAPTER 3

Hilbert basis

This chapter is devoted to the computation of a minimal generating system of the integral closure of a monoid, the Hilbert basis. For this purpose we will first consider simplicial cones and then use triangulations for non-simplicial cones.

Afterwards Section 3.4 introduces the new concept of a partial triangulation, together with examples where it has an enormous effect. The last two sections discuss the for large examples indispensable extension of effectively avoiding duplicates.

3.1. Hilbert basis

From now on we will always assume that the rational cone $C \subset \mathbb{R}^d$ is pointed and full-dimensional, i.e. $\dim C = d$. Furthermore, we will work with the lattice $L = \mathbb{Z}^d$. This situation can always be achieved by a linear transformation from Section 1.3.

Definition 3.1. Let M be an positive affine monoid. An element $x \in M$, $x \neq 0$, is *irreducible* if $x = y + z$ with $y, z \in M$ implies $y = 0$ or $z = 0$. If there exist such a sum with $y \neq 0$ and $z \neq 0$ we call x *reducible* and y, z *reducers* of x.

Theorem 3.2. *Let M be a positive affine monoid. Then there are only finitely many irreducible elements, and they form the unique minimal system of generators of M, the* Hilbert basis $\operatorname{Hilb}(M)$.

For a proof see [**4**, Prop. 2.14]. Hilbert bases are used in various areas. They can be interpreted as the set of minimal solutions of a homogeneous system of diophantine equations, inequalities and congruences. The common case of homogeneous system of diophantine equations over the nonnegative integers relates them to combinatorial optimization, see [**23**]

In this chapter we will describe how to compute $\operatorname{Hilb}(C) := \operatorname{Hilb}(M)$ for the integrally closed monoid $M = C \cap L$, and in this way we will also prove Theorem 3.2 in this case. Note that the irreducible elements must be contained in every generating system. For a cone C we first find a triangulation Γ as described in Chapter 2 and then compute the Hilbert basis for each simplicial cone. The simplicial case will be described in Section 3.3. Then we take the union of the Hilbert bases of the simplicial cones to obtain the set of *candidates*

$$\operatorname{Cand}(C) = \bigcup_{\sigma \in \Gamma} \operatorname{Hilb}(\sigma).$$

Of course, $\operatorname{Cand}(C)$ depends on the used triangulation. In Normaliz we always use the lexicographical triangulation (or parts of it, which lead to the same set

Cand(C) as will discuss in Sections 3.4 and 3.6). The candidates generate the monoid $M = C \cap L$.

Lemma 3.3. *Let C be a cone with a triangulation Γ. Then the set* Cand(C) *is a finite superset of* Hilb(C).

Proof. An element $x \in \operatorname{Hilb}(C) \subset C$ lies in some $\sigma \in \Gamma$. Since there is no reducer of x in C, there is certainly no reducer in σ. Cand(C) is finite since the triangulation and the Hilbert basis of the simplicial cones are finite, see 3.5 and 3.6, and therefore also the union. □

3.2. Global reduction

From the set of candidates we have to exclude the reducible candidates to get the Hilbert basis of C. This is the task of the global reduction. In order to do it efficiently we need the support hyperplanes of C and a positive grading deg, which is a linear form such that $\deg(x) > 0$ for all $x \in M, x \neq 0$. We can use the sum over all support hyperplane linear forms as such a grading.

Lemma 3.4. *Let $C \subset \mathbb{R}^d$ be a rational pointed cone with support hyperplane linear forms $\lambda_1, \dots, \lambda_s$ and $M = C \cap \mathbb{Z}^d$. An element $y \in M$, $y \neq 0$ is reducible if and only if there exists a Hilbert basis element x with*

(i) $\deg(x) \leq \deg(y)/2$, *and*
(ii) $\lambda_i(x) \leq \lambda_i(y)$ *for all* $i = 1, \dots, s$.

Proof. Condition (ii) is equivalent to $y - x \in M$, which in turn is equivalent to x being a reducer of y. Now it is sufficient to see that every element in M can be written as a finite sum of Hilbert basis elements and not more than one can have degree bigger than $\deg(y)/2$. □

Condition (i) is very useful to decide which elements must be tested as possible reducers and (ii) gives an efficient way to check reducibility.

To make effective use of condition (ii) we sort the candidates ascending by degree. In a straightforward implementation with parallelization each thread would now take the first available candidate and apply the Lemma 3.4. This would require a lot of synchronization and bookkeeping because a newly found Hilbert basis element might be a reducer of a candidate that is handled at the same time by another thread. Algorithm 4 avoids this problem by sectioning the reducers into ranges of degrees. At the same time it also does not make reduction tests multiple times. The degree ranges are determined during the algorithm according to the degrees appearing.

The most work is done in the for loop that starts at line 12, and this loop can easily be parallelized. Line 16 reflects a heuristic trick. Is a reducer found, it often also reduces other candidates. We move it to the begin so that further candidates will first be tested with it. To avoid the need for synchronization, each thread keeps its own order of R.

Algorithm 4 Reduce a candidate set to the Hilbert basis

Require: A generating set G of M, a positive grading deg, the support hyperplanes $\lambda_1, \dots, \lambda_s$ of C

Return: The Hilbert basis Hilb of M

```
 1: function globalReduction($G$, deg, $\lambda = (\lambda_1, \dots, \lambda_s)$)
 2:     for $y \in G$ do
 3:         compute and store $\deg(y)$
 4:     sort $G$ ascending by degree
 5:     Hilb $\leftarrow \emptyset$                                   ▷ collects the Hilbert basis
 6:     while $G \neq \emptyset$ do
 7:         $k \leftarrow 2\deg(y)$ where $y$ is the first element of $G$      ▷ degree bound
 8:         $R \leftarrow \emptyset$
 9:         for $x \in G : \deg(x) < k$ do                        ▷ irreducible elements
10:             move $x$ from $G$ to Hilb
11:             add $v = (\lambda_1(x), \dots, \lambda_s(x))$ to $R$
12:         for $y \in G$ do                        ▷ Now the elements with $\deg(y) \geq k$
13:             for $v = (v_1, \dots, v_s) \in R$ do
14:                 if $\lambda_i(y) \geq v_i$ for all $i = 1, \dots, s$ then      ▷ (ii) in Lemma 3.4
15:                     remove $y$ from $G$
16:                     move $v$ to the begin of $R$
17:     return Hilb
```

3.3. Hilbert basis of a simplicial cone

For this section let σ be a simplicial cone of dimension d with extreme integral generators $x_1, \dots, x_d$ and $M = \sigma \cap \mathbb{Z}^d$. In order to find the Hilbert basis of σ we first look for a finite generating set, also see [**4**, Prop. 2.43].

Theorem 3.5. *Let σ be a simplicial cone of dimension d with extreme integral generators $x_1, \dots, x_d$ and $M = \sigma \cap \mathbb{Z}^d$. Furthermore let U be the sublattice generated by $x_1, \dots, x_d$ and N the monoid generated by $x_1, \dots, x_d$. Then the set*

$$E = \{q_1 x_1 + \cdots + q_d x_d \ : \ 0 \leq q_i < 1, i = 1, \dots, d\} \cap \mathbb{Z}^d$$

is a system of representatives of $\mathbb{Z}^d/U$ and also a basis of M as free N-module. In particular we have that $E \cup \{x_1, \dots, x_d\}$ generates M as a monoid.

Proof. Since $x_1, \dots, x_d$ are linearly independent, every $y \in \mathbb{Z}^d$ can uniquely be written as a linear combination $y = a_1 x_1 + \cdots + a_d x_d$, $a_i \in \mathbb{R}$. The coefficients can uniquely be decomposed into $a_i = \lfloor a_i \rfloor + b_i$ such that $\lfloor a_i \rfloor \in \mathbb{Z}$ and $0 \leq b_i < 1$ and lead to $y = z + m$ with $z = b_1 x_1 + \cdots + b_d x_d \in E$ and $m = \lfloor a_1 \rfloor x_1 + \cdots + \lfloor a_d \rfloor x_d \in U$. A different decomposition $y = z' + m'$ with $z' \in E$, $m' \in U$ cannot exist since it would give y as a different linear combination of $x_1, \dots, x_d$. This shows the statement for $\mathbb{Z}^d/U$ and it follows analogously that E is a basis of M as N-module. $\square$

Lemma 3.6. *With the notation of Theorem 3.5 the Hilbert basis of σ is given by*

$$\mathrm{Hilb}(\sigma) = \{x_1, \dots, x_d\} \cup \{x \in E \setminus \{0\} : \text{ there exist no reducer } y \in E\}.$$

Proof. The Hilbert basis of σ is a subset of every generating set of M, in particular of $E \cup \{x_1, \dots, x_d\}$. Now the reducible elements have to be removed from this set to get the Hilbert basis. First observe that $x_1, \dots, x_d$ are irreducible because they are the linear independent extreme integral generators of σ and $x_i = y + z$ implies that $y, z \in M$ are linear combinations of $x_1, \dots, x_d$ with nonnegative coefficients, but their sum can uniquely be written as the linear combination $1 \cdot x_i$. On the other hand no x_i can be a reducer of any $y \in E$, since the coefficient of x_i in the linear combination of $y - x_i$ in $x_1, \dots, x_d$ would be negative. □

Example 3.7. Consider the cone $C = \mathrm{cone}(x_1, x_2)$ with $x_1 = (3, 1)$, $x_2 = (1, 2)$.

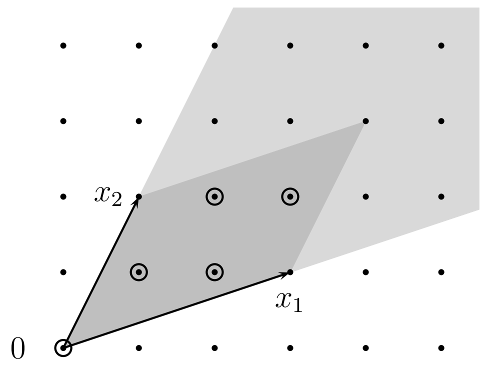


FIGURE 1. The cone C with semi-open parallelogram (dark grey) and set E (circled points).

We see from Figure 1 the set integer points in the semi-open parallelogram is

$$E = \{(0, 0), (1, 1), (2, 1), (2, 2), (2, 3)\}$$

and can check that $(1, 1)$ is a reducer of $(2, 2)$ and $(2, 3)$, whereas $(1, 1)$ and $(2, 1)$ are irreducible. So we get

$$\mathrm{Hilb}(C) = \{x_1 = (3, 1), x_2 = (1, 2), (1, 1), (2, 1)\}.$$

Now we will present an algorithm to generate the points in E and test whether they are reducible (in σ) in a very efficient way.

Proposition 3.8. *Let G and X be $d \times d$-matrices with integer entries, such that $|\det X| = 1$ and $T = XG$ is an upper triangular matrix with diagonal elements $t_1, \dots, t_d > 0$. Then the vectors in*

$$S = \{(s_1, \dots, s_d) : \ 0 \le s_i < t_i\}$$

form a system of representatives of $\mathbb{Z}^d/U$ where U is the sublattice generated by the rows of G. Especially we have $[\mathbb{Z}^d : U] = t_1 \cdots t_d = |\det G|$.

Proof. Since X is invertible over $\mathbb{Z}$, the rows of $T = XG$ also form a $\mathbb{Z}$-basis of U. Every vector $y \in \mathbb{Z}^d$ reduces to a $y' \in S$ after subtraction of suitable multiples of rows T_i of the triangular matrix T via the recursion

$$y^0 = y, \quad y^i = y^{i-1} - \lfloor y_i^{i-1}/t_i \rfloor \cdot T_i \text{ for } i = 1, \ldots, d, \quad y' = y^d.$$

Hence S generates $\mathbb{Z}^d/U$.

Let $v, w \in S$ belong to the same residue class in $\mathbb{Z}^d/U$, i.e. $v - w \in U$. Then there exist $a_1, \ldots, a_d \in \mathbb{Z}$ such that $v - w = a_1 T_1 + \cdots + a_d T_d$. Assume that $v \neq w$, then there is a $a_i \neq 0$. Let k be the smallest $k \in \{1, \ldots, d\}$ with $a_k \neq 0$. The triangular structure of T then implies $v_i - w_i = 0$ for $i = 1, \ldots, k-1$ and $v_k - w_k = a_k t_k$. This is a contradiction to $|v_k - w_k| < t_k$. □

Definition 3.9. A simplicial cone whose extreme integral generators generate $\mathbb{Z}^d$ is called *unimodular.*

The unimodular cones are exactly those with $|\det G| = [\mathbb{Z}^d : U] = 1$. They have $E = \{0\}$ and the Hilbert basis consists only of the extremal integral generators.

Proposition 3.8 gives an efficient way to computationally construct a system of representatives of $\mathbb{Z}^d/U$. According to Theorem 3.5 each class has exactly one representative in E. We now have to change from the representatives in S to the ones in E. For $s \in S$ this can easily be done when we know how to write s as a linear combination of $x_1, \ldots, x_d$. For $s = q'_1 x_1 + \cdots + q'_d x_d$ we have to add integer values to the coefficients q'_i such that for the resulting coefficients q_i we have $0 \leq q_i < 1$, in other words we have to reduce the coefficients mod 1.

Writing s as a linear combination of $x_1, \ldots, x_d$ can be viewed as a basis change from the standard basis to the basis $x_1, \ldots, x_d$. In this perspective the computations

$$q' = s \cdot G^{-1}, \qquad q \equiv q' \mod 1, \qquad v = q \cdot G$$

give us the $v \in E$.

This step is done very often, so it is time critical and crucial to make it as fast as possible. We will now present optimizations of this computations. In Section 2.1 we already discussed how to compute T and G^{-1} using only integer arithmetic. To continue the use of integers we multiply q and q' by $\delta = |\det G|$ and also reduce modulo δ, so the computations are

$$b' = s \cdot (\delta \cdot G^{-1}), \qquad b \equiv b' \mod \delta, \qquad v = (b \cdot G)/\delta. \tag{1}$$

In fact, after the computation of T one sees that not all rows of G^{-1} are used, but only those rows i, that have $t_i > 1$. In most cases the number of entries in the diagonal of T that are greater than 1 is very small. For example a cone in dimension 7 generated by 14 vectors with random entries between 0 and 9, in all 4134 simplicial cones of the triangulation only 7135 diagonal elements greater than 1 appear. So in average less than two diagonal entries different from 1.

Example 3.10. We show the computations for Example 3.7. First we trigonalize G to compute T and see that only the last row $(G^{-1})_2$ of the inverse is needed. Then we solve the (transposed) system for only $(G^{-1})_2$.

$$\begin{array}{rr} 3 & 1 \\ 1 & 2 \\ \hline 1 & 2 \\ 3 & 1 \\ \hline 1 & 2 \\ 0 & -5 \\ \hline 1 & 2 \\ 0 & 5 \end{array} \qquad T = \begin{pmatrix} 1 & 2 \\ 0 & 5 \end{pmatrix} \qquad \begin{array}{rr|r} 3 & 1 & 0 \\ 1 & 2 & 1 \\ \hline 1 & 2 & 1 \\ 3 & 1 & 0 \\ \hline 1 & 2 & 1 \\ 0 & -5 & -3 \\ \hline 1 & 2 & 1 \\ 0 & 5 & 3 \\ \hline 5 & 10 & 5 \\ 0 & 5 & 3 \\ \hline 5 & 0 & -1 \\ 0 & 5 & 3 \end{array} \qquad (G^{-1})_2 = \frac{1}{5}\begin{pmatrix} -1 & 3 \end{pmatrix}$$

For each s in the system of representatives of $\mathbb{Z}^d/U$ according to Proposition 3.8, s itself and the further steps in the computation are listed in the rows of the table below.

s	$(0,0)$	$(0,1)$	$(0,2)$	$(0,3)$	$(0,4)$
$b' = s \cdot (5 \cdot G^{-1})$	$(0,0)$	$(-1,3)$	$(-2,6)$	$(-3,9)$	$(-4,12)$
$b \equiv b' \mod 5$	$(0,0)$	$(4,3)$	$(3,1)$	$(2,4)$	$(1,2)$
$v = (b \cdot G)/5$	$(0,0)$	$(3,2)$	$(2,1)$	$(2,2)$	$(1,1)$

Of course in this small example it would have been no disadvantage to completely invert the generator matrix.

But in general the presented method can be beneficial; especially in examples were the number of diagonal elements greater than 1 is very small. Unimodular simplicial cones have even no such diagonal entry and simplicial cones with low determinant can maximally have so many as factors in the prime decomposition of the determinant. The simplicial cones of the contingency tables considered in Section 5.1 are either unimodular or have only exactly one diagonal entry that is different from 1 while the dimension goes up to 43. Here a remarkably amount of time can be saved by not inverting the matrix completely.

Table 3.1 shows the timings for 100.000 iterations of the trigonalize and solve steps necessary and also for inverting the matrix as implemented in Normaliz. Note that the matrix multiplication with the inverse to find a solution is not included in the time for inverting. The first test examples are matrices in dimension 7, 12 and 20 with random entries for 0 to 9. The last two examples are the first non-unimodular simplicial cones in the lexicographical triangulations of the monoids related to $4 \times 4 \times 3$ and $5 \times 5 \times 3$ contingency tables. For the computation either the 64 bit C++ integer type `long long` or the arbitrary precision type `mpz_class` from the GMP library [**40**] are used.

example	dim.	right sides	integer type	trigonalize	solve	invert
random7	7	2	64 bit	0.17 sec	0.23 sec	0.32 sec
random7	7	2	gmp mpz	3.13 sec	3.83 sec	7.45 sec
random12	12	1	64 bit	0.70 sec	0.77 sec	1.20 sec
random12	12	1	gmp mpz	15.75 sec	16.79 sec	42.47 sec
random20	20	2	gmp mpz	68.81 sec	76.87 sec	206.59 sec
A443-simp	30	1	64 bit	0.78 sec	0.90 sec	3.38 sec
A443-simp	30	1	gmp mpz	5.26 sec	7.54 sec	88.49 sec
A553-simp	43	1	64 bit	1.45 sec	1.68 sec	8.03 sec

TABLE 3.1. Comparison trigonalize and solve versus invert

One remark on the computation of b' and b. The multiplication by δG^{-1} can be replaced by just adding a row of δG^{-1} to a previously computed value, since every vector in S is a vector that has been created before increased in one position by 1 (starting with $(0, \dots, 0)$). The reduction modulo δ can be simplified by reducing δG^{-1} modulo δ first, then the reduction of b' can be done by subtracting δ only one time if necessary.

Now we compute the Hilbert basis of σ using Lemma 3.6. To make this *local reduction* of the candidate set $E \setminus \{0\}$ we use a modified version of the GLOBALREDUCTION method from Algorithm 4. The relevant information in the reduction are the heights $\lambda_i(v)$, these can be replaced by the values b_i. They differ from $\lambda_i(v)$ by a constant factor that only depends on i, and not on v, namely the greatest common divisor of the i-th row of $\delta \cdot G^{-1}$. Therefore they can be used to check if a difference lies within the cone and the linear forms of the support hyperplanes are not explicitly needed. As the degree function we use the sum of the coefficients of b. Thus the reduction can completely be done with the b vectors and the multiplication by G will only be done for Hilbert basis elements of σ.

In principle this local reduction to a Hilbert basis is not necessary. One could also take the union of all the sets E as input for the global reduction. But the presented local reduction is very efficient since no extra scalar products have to be computed. Also checking if an element reduces another only need dimensional many comparisons, since σ has that number of support hyperplanes, whereas C can have many more. Usually the local reduction removes a lot of reducible elements. For the example small from the Normaliz distribution it removes 88.8% of the vectors in the sets $E \setminus \{0\}$. In conclusion the local reduction is a very useful preliminary step to the global reduction.

3.4. Partial triangulation

In practice one big problem for the computation of Hilbert bases with the presented algorithm is that the triangulation can get huge, especially in large dimensions. One example for this are some monoids which arise in algebraic statistics. In [**30**] Hibi and Ohsugi examined the normality of the monoids of marginal distributions of contingency tables by taking line sums. They were able to decide it in all but the three cases of $4 \times 4 \times 3$, $5 \times 4 \times 3$ and $5 \times 5 \times 3$ contingency tables. The generators of these monoids are the marginals of the contingency tables which contain exactly one entry 1 and 0 otherwise. Therefore the generators are 0-1-vectors with a fixed number of entries 1. More on these monoids in Section 5.1.

To decide the normality we tried to compute the Hilbert basis and compare it to the given generators. For $5 \times 5 \times 3$ contingency tables, the biggest example, we get a 43-dimensional cone with 75 extreme rays. In 2009 it was not possible to compute its Hilbert basis by available software like Normaliz v2.2 or `4ti2` v1.3.2 [**1, 24**] due to the huge time and memory requirements of the used algorithms.

The main problem here is the huge triangulation. The lexicographic triangulation of that example has more than $9 \cdot 10^9$ simplicial cones but most of them are unimodular. That means their Hilbert basis consists of the integral extreme generators and therefore they do not need to be considered in the computation of the global Hilbert basis. To understand the structure of these cones we looked at smaller contingency tables. In [**30**] it was also shown that for $3 \times 3 \times 3$ tables and all smaller examples the cones are compressed, what implies the existence of an unimodular triangulation and therefore normality. A cone is *compressed* if every reverse lexicographic triangulation (or pulling triangulation) is unimodular. Another characterization is that all extreme integral generators have height 1 over all support hyperplanes which do not contain them, see [**39**]. The term compressed is appropriate since the cone generators are "squeezed" between each support hyperplane and the parallel affine hyperplane which is shifted by 1. A Normaliz computation shows that also for $4 \times 3 \times 3$ we get a compressed cone. For the three open cases this is not true anymore. However, they are not too far away from it. The heights of the generators over the support hyperplanes are at most 2.

One important ingredient to master this example is the use of pyramids as described in Section 2.4, which allows us to not store the complete triangulation. But the breakthrough was the *partial triangulation*: We exploited the idea of compressed in a weaker form to omit pyramids whose non-extreme Hilbert basis elements are contained in previously computed simplicial cones. The following lemma makes more precise which pyramids can be excluded.

Lemma 3.11. *Let C be a rational cone with extreme integral generators $x_1, \dots, x_n$. Suppose $F = \operatorname{cone}(x_1, \dots, x_{n-1})$ is a facet of C, λ the linear form of F and $\lambda(x_n) = 1$. Then the Hilbert basis of C is*

$$\operatorname{Hilb}(F) \cup \{x_n\}.$$

Proof. We can write $y \in C \cap \mathbb{Z}^d$ as $y = \sum_{i=1}^n q_i x_i$ with $q_1 \dots, q_n \in \mathbb{R}$. Since y is an integral vector,

$$\lambda(y) = \lambda\left(\sum_{i=1}^n q_i x_i\right) = \sum_{i=1}^n q_i \lambda(x_i) = q_n$$

is integral. In the case of $q_n = 0$ we have $y \in F$. Otherwise $q_n \geq 1$ and x_n is a reducer of y. This completes the proof. □

The partial triangulation now computes a superset of the Hilbert basis with the pyramid decomposition techniques from Chapter 2 and uses the lemma above to exclude parts of the triangulation.

Corollary 3.12. *Let $x_1, \dots, x_n \in \mathbb{Z}^d$ such that $C' = \operatorname{cone}(x_1, \dots, x_{n-1})$ has dimension d, and $C = \operatorname{cone}(x_1, \dots, x_n)$. Let $F_1, \dots, F_s$ be the facets of C', let $\lambda_1, \dots, \lambda_s$ be the normal linear forms of these facets, and for $i = 1 \dots, s$ let C_i be the cone generated by F_i and x_n. Then*

$$\operatorname{Hilb}(C') \cup \{x_n\} \cup \bigcup \{\operatorname{Hilb}(C_i) : -\lambda_i(x_n) \geq 2,\ i = 1, \dots, s\}$$

generates the monoid $C \cap \mathbb{Z}^d$.

Proof. If the condition in the union over the Hilbert bases of the pyramids $C_1, \dots, C_s$ is changed to $-\lambda_i(x_n) \geq 1$ we get the regular pyramid decomposition from Chapter 2, since the other facets are not visible from x_n. The cones with $-\lambda_i(x_n) = 1$ can be excluded by Lemma 3.11. Note that here the orientation of λ_i is such that $C' \subset H^+_{\lambda_i}$. □

Remarks 3.13. (a) The partial triangulation can exclude complete pyramids, not only simplicial cones. These pyramids do not have to have a unimodular triangulation. It is only important that the interior Hilbert basis lies in the base facet of the pyramid.

(b) Nevertheless, the partial triangulation still produces exactly the same set of candidates as the full triangulation.

Obviously the partial triangulation is of especially great use if the height of a generator over the facets of the so far constructed cone is seldom greater than 1. A large group of examples where this is often the case are monoids that encode some combinatorial information. Such monoid mostly have generators whose entries are in $\{0, 1\}$ or in $\{-1, 0, 1\}$, like the monoids arising from contingency tables. Another such example is the cone associated to the semi-graphoid for $|N| = 5$, see [**36**]. It was already shown in [**25**] that the corresponding monoid is not normal by explicit construction of a single Hilbert basis element that is not one of the given generators via a different method. The computation of the full Hilbert basis was not possible at that time.

With the partial triangulation and parallelization of the algorithms we were able to compute the Hilbert bases of these monoids. Data on the size of the examples and the computation times of Normaliz 2.9 with 20 threads on our Sun Fire X4450 are given in Table 3.2. The size of the full triangulation was counted by Normaliz in a separated run without storing or evaluating the triangulation. The computation shows that the monoid of the $5 \times 5 \times 3$ contingency table is normal, which completes the normality classification in [**30**]. The $6 \times 4 \times 3$ contingency table is the next larger monoid that is not normal, and we have computed the full Hilbert basis.

	Contingency tables		Semigraphoid
	$5 \times 5 \times 3$	$6 \times 4 \times 3$	$N = 5$
dimension	43	42	26
# extreme rays	75	72	80
# Hilbert basis	75	4,392	1,300
# support hyperplanes	306,955	153,858	117,978
# full triangulation	9,248,527,905	3,100,617,276	1,045,346,320
# partial triangulation	364,862	206,064	3,109,495
# candidates	41,518	10,872	168,014
real time	21:46 min	3:46 min	9:09 min

TABLE 3.2. Data of challenging Hilbert basis computations

A different approach was independently given by R. Hemmecke and M. Köppe. They exploit symmetries to drop parts of the cone which were already considered "up to symmetry". Both approaches together with successful applications to the examples in Table 3.2 and to cut monoids of graphs (see Section 5.2) were presented in [**6**].

3.5. Avoiding duplicates

Candidates that are in the boundary of a simplicial cone can also appear in a different simplicial cone. If a point is in a lower dimensional face, then there can be even multiple simplicial cone sharing this face and therefore the point will be created multiple times. When a point is created not for the first time, we will call it *duplicate*. This terminology considers the point together with the simplicial cone in which it was created. The duplicates can be omitted for the Hilbert basis computation.

Of course it highly depends on the example how many duplicates we will get. Combinatorial examples typically have small simplicial cone with a high percentage of the points in the boundary of the simplicial cone and hence have a lot of duplicates. For some examples Table 3.3 shows the number of unique candidates, the total number of candidates created (including duplicates) and the percentage of duplicates. The numbers are given for both the full triangulation and the partial triangulation from the previous section. In the partial triangulation many simplicial cone that would only give duplicates are excluded, as a result it has a lower amount of duplicates.

The example random5 has 12 generators with random entries from 0 to 9 in dimension 5; small is the cone over a polytope of dimension 5 with 190 vertices; medium is generated by 40 0-1-vectors of dimension 17; lo6 is the linear order polytope connected to the inversion model of statistical ranking for $n = 6$, see [**38**]. The next example is the monoid of a base ring associated to a transversal polymatroid provided by A. Ştefan [**35**]. In this case the partial triangulation has no effect and the numbers in Table 3.3 are just the Hilbert basis elements of degree 1. The total number of Hilbert basis elements can be expected to be several times larger but we did not compute them to avoid long computation times. The computation of the degree 1 elements was practically not possible in Normaliz 2.7 since the duplicates used too much memory. The cutgraph is the cut monoid of some graph with 8 vertices, see Section 5.2.

		full triangulation		partial triangulation	
	unique	created	duplicates	created	duplicates
random5	158,007	158,562	0.35%	158,562	0.35%
small	221,333	253,182	12.58%	249,349	11.24%
medium	3,450	41,252	91.64%	19,493	82.30%
lo6	1,068,003	55,198,831	98.07%	8,444,340	87.35%
Stefan	1,322,271	353,838,470	99.62%	353,838,470	99.62%
cutgraph	8,483,905	180,589,842	95.30%	45,080,950	81.18%

Table 3.3. Candidates and duplicates

One technique to remove the duplicates is to sort the list of all created points and delete the duplicates which are now next to each other, or keep the candidates as an ordered set all the time. For a huge number of candidates this is time expensive. In addition this approach is unfavorable for parallelization. Either one tries to keep all candidates created unique which requires synchronization between the threads and hurts the parallelization, or one handles the candidates from different threads independently (at least for some time) which means increased memory usage.

But there is a more elegant and efficient way. If we compute a full triangulation we can use the exclusion technique for facets from Lemma 2.7 to decide which points are duplicates. This gives us a direct decision whether a created Hilbert basis element of a simplicial cone should be considered for the set of all candidates or should be threated as a duplicate.

Definition 3.14. Let C be rational cone with extreme integral generators $x_1, \dots, x_n$ and O_C be an order vector of C as in Lemma 2.7. Furthermore let D be a subcone of C whose extreme integral generators are a subset of $x_1, \dots, x_n$, and $\lambda_1, \dots, \lambda_l$ the linear forms of the support hyperplanes of D. We define

$$\widetilde{\mathrm{Hilb}}(D) = \{x \in \mathrm{Hilb}(D) : \lambda_j(x) > 0 \text{ for all } j \text{ such that } \lambda_j(O_C) < 0\}$$

as the set of Hilbert basis elements that are not excluded by the order vector criterion.

Lemma 3.15. *Let $\sigma_1, \dots, \sigma_m$ be the maximal simplicial cones in a triangulation of a pointed cone $C \subset \mathbb{R}^d$ and let O_C be an order vector of C as in Lemma 2.7. Then*

$$\bigcup_{i=1}^{m} \widetilde{\mathrm{Hilb}}(\sigma_i)$$

is a disjoint union and generates the monoid $C \cap \mathbb{Z}^d$.

We use the criterion after the local reduction in the simplicial cones. A duplicate point can still be a reducer of other candidates. It would not be necessary to try to reduce the point but if it is reducible we do not have to check if it is a reducer of another point.

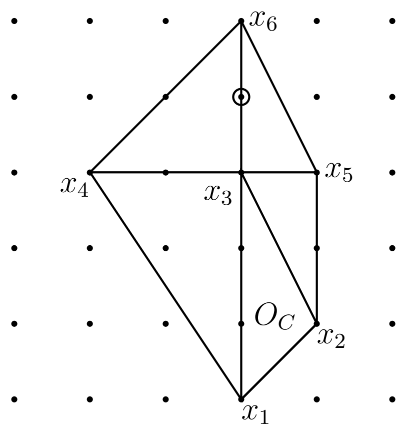
Remark 3.16. The presented duplicate avoidance criterion marks every candidate exactly once as “no duplicate”. In this way we get the same set of candidates as before and do not have to make the list of gathered candidates unique.

How useful this technique is can already be seen in Table 3.3. We will now extend it to partial triangulations.

3.6. Avoiding duplicates in partial triangulations

Unfortunately the duplicate avoidance criterion is not directly applicable to partial triangulations as the following example shows.

Example 3.17. Consider the cone over the polytope generated by $x_1 = (0,0)$, $x_2 = (1,1)$, $x_3 = (0,3)$, $x_4 = (-2,3)$, $x_5 = (1,3)$, and $x_6 = (0,5)$. We now build the partial lexicographical triangulation where the pyramids are again triangulated with the partial triangulation.



When generator x_6 is added, the pyramid over x_4, x_3, x_5 with x_6 is formed. In this pyramid the start simplicial cone has the vertices x_3, x_4, x_6. Now the height of x_5 over x_3, x_6 is 1 and therefore the pyramid x_3, x_6, x_5 is disregarded in the partial triangulation. If now the duplicate avoidance approach is used with an order vector O_C inside x_1, x_2, x_3 the point $(0, 4)$ is marked as a duplicate in x_3, x_4, x_6 and the simplical cone x_3, x_5, x_6 in which it should have been considered is not in the partial triangulation.

The problem here is that the partial triangulation and the order vector criterion use two different approaches to exclude duplicates. These approaches do not harmonize with each other. To be more precise we use the notation of Corollary 3.12: The problem arises when a pyramid C_i is excluded by the partial triangulation because its interior Hilbert basis is contained in the base facet F_i but the base facet is excluded in the cone C' by the order vector criterion. One solution to fix this is to add all these cones to the partial triangulation.

Lemma 3.18. *With the notation of Corollary 3.12 and an order vector O_C as in Lemma 2.7 we have*

$$\widetilde{\mathrm{Hilb}}(C') \cup \{x_n\} \cup \bigcup \left\{ \widetilde{\mathrm{Hilb}}(C_i) : \begin{matrix} -\lambda_i(x_n) \geq 2 \textit{ or} \\ (-\lambda_i(x_n) = 1 \textit{ and } \lambda_i(O_C) < 0) \end{matrix}, \ i = 1, \dots, s \right\}$$

is a disjoint union and generates the monoid $C \cap \mathbb{Z}^d$.

This leads to the following recursive algorithm that returns a partial triangulation which can be used together with the duplicate avoidance criterion.

The algorithm for the partial triangulation in the form of Lemma 3.12 can be easily obtained from this algorithm by removing “or $\lambda(O_C) < 0$” from the condition on line 12.

Remarks 3.19. (a) At the top level the condition $\lambda_i(O_C) < 0$ is always fulfilled since we choose O_C in the start simplicial cone.
(b) The algorithm produces a partial triangulation that is a subset of the one that would be produced using the partial triangulation criterion only on the top level cone and making a full triangulation for every pyramid.
(c) On the other hand it is a superset of the strict partial triangulation as in Corollary 3.12.
(d) In practice we do not use the partial triangulation recursively to the end, but switch back to the full triangulation at a certain recursion level because this method is faster for small pyramids.
(e) Which parts appear in this partial triangulation now depends on the order vector.
(f) Regardless of all the variations, the set of unique candidates is always the same!

Table 3.4 shows that the number of simplicial cones to evaluate is not much higher than in the standard partial triangulation. In some cases there are almost twice as many simplicial cones, but still very few in comparison to the full triangulation. And also the computation times are almost the same for the first examples in the

Algorithm 5 Partial triangulation with duplicate avoidance

```
Require: $G = \{x_1, \dots, x_n\} \subset \mathbb{Z}^d$, we assume that $x_1, \dots, x_d$ are linearly independent
Return: The partial triangulation $\Gamma$
 1: function buildPartialTriangulation($G$)
 2:     $\mathcal{H} \leftarrow$ invert $(x_1, \dots, x_d)$ over $\mathbb{Z}$
 3:     $\Gamma \leftarrow \{\text{cone}(x_1, \dots, x_d)\}$
 4:     $O \leftarrow$
 5:     for $i \leftarrow d + 1$ to $n$ do
 6:         $\Gamma \leftarrow \Gamma \cup$ partialTriangulate($\mathcal{H}$, $x_i$)
 7:         $\mathcal{H} \leftarrow$ findNewFacets($\mathcal{H}$, $x_i$)                ▷ see Algorithm 1
 8:     return $\Gamma$

Require: A cone generated by $E$ with support hyperplanes $\mathcal{H}$ and order vector $O$
Return: The partial triangulation of cone$(E, x)$
 9: function partialTriangulate($\mathcal{H}$, $E$, $x$)
10:     for $\lambda \in \mathcal{H}$ do
11:         if $\lambda(x) < 0$ then                                     ▷ visibility check
12:             if $\lambda(x) \neq -1$ or $\lambda(O_C) < 0$ then       ▷ partial triangulation check
13:                 $F \leftarrow \{y \in E \ : \ \lambda(y) = 0\}$
14:                 if $|F| = d - 1$ then                                ▷ simplicial case
15:                     $\Gamma \leftarrow \Gamma \cup \{\text{cone}(F, x)\}$
16:                 else
17:                     $\Gamma \leftarrow \Gamma \cup$ buildPartialTriangulation($F \cup \{x\}$)
18:     return $\Gamma$
```

table. For the last two the duplicate avoidance starts do be important and therefore leads to much better computation times. The cutgraph example has more than 8 million unique candidates and 45 million created points. Here the computation time reduces from 26 minutes to 11 minutes.

The example 7nog corresponds to the claw tree with 7 leaves based on the group $\mathbb{Z}_2 \times \mathbb{Z}_2$ (3-Kimura model) from [**20**]. 4096 vectors generate a 22-dimensional cone with 220 support hyperplanes. We stopped the counting of the full triangulation after 24 hours on our compute server. Until then it had reached $109 \cdot 10^9$ simplicial cones and we expect the complete triangulation to be 5 times as big. With the standard partial triangulation we stopped it after 2 days because the removal of the duplicates took to much time. The Hilbert basis computation with the duplicate avoidance in the partial triangulation completed in about 4 hours. In the global reduction 371,665,139 unique candidates had to be reduced, to find that 7nog is normal. This result verifies the conjecture [**20**, Conj. 3.13] for any tree with vertices of degree (at most) 7.

	number of simplicial cones in		
	full triangulation	partial triangulation	with duplicate avoidance
medium	60,928	12,646	21,046
A543	102,538,980	3,225	3,430
A553	9,248,527,905	364,862	481,466
lo6	5,745,903,354	3,350,516	5,390,175
semi5	1,045,346,320	1,164,210	2,150,654
cutgraph	758,457,886	36,101,158	58,734,891
7nog	$> 100 \cdot 10^9$	$2.98 \cdot 10^9$	$4.11 \cdot 10^9$

TABLE 3.4. Partial triangulation comparison

Another example were the avoidance of duplicates plays an important rule is the linear order polytope for $n = 7$. The complete Hilbert basis computation is at the moment not doable in a reasonable time, but tests with the partial triangulation showed that after the first 1000 generators of 5040 we already have about 35 million unique candidates out of 250 million.

In the presented combination the partial triangulation is extended in order to make the duplicate avoidance via order vector applicable. Another possibility is to adjust the order vector criterion. If we give each pyramid its own order vector inside the pyramid, we can get the candidates of that pyramid without duplicates. And the union of these sets again creates the monoid of the superior cone, but this union is not disjoint. To get a disjoint union, the candidates in the boundary of the pyramid must then be considered with respect to the order vector of the superior cone. For this we not only have to compute which candidates in the pyramids are in the boundary, but more importantly, we must keep track of the (sub-)pyramid structure. In this way we would loose the independence of the simplicial cones and the pyramids, which makes the parallelization so effective. Therefore we prefer the extension of the partial triangulation.

The data collected from the examples shows that the duplicate avoidance approach is clearly preferable for large examples. And it also brings another advantage when the number of unique candidates grows and it is not feasible to store them all in the memory. In this case we we can do intermediate global reductions of the candidate set and really forget about the reducible candidates. They will not come into the candidate set again later thanks to the duplicate avoidance. This became necessary for 7nog and also for cut monoids of graphs with 10 vertices, see Section 5.2.

CHAPTER 4

Hilbert series and function

In this chapter we will cover the topic of counting lattice points of certain degree inside a cone. As for the Hilbert basis we start again with the simplicial case and use a triangulation to put the result together for a non-simplicial cone. Section 4.5 handles the special case of computing the volume of a polytope.

4.1. Hilbert series computation with order vector

We use a degree function which is given by a surjective grading linear form $\deg : \mathbb{Z}^d \to \mathbb{Z}$ with $\deg(x) = 0$ if and only if $x = 0$, see Section 1.2. We could work with a sublattice of $\mathbb{Z}^d$, but, as already discussed, for algorithmic purposes it is better to apply a linear transformation first. Therefore in this section always the lattice $\mathbb{Z}^d$ will be used.

Definition 4.1. For any subset $C \subset \mathbb{R}^d$ and $i \in \mathbb{Z}$ we define

$$H(C, i) = |\{x \in C \cap \mathbb{Z}^d : \deg(x) = i\}|.$$

If $H(C, i)$ is finite for all $i \in \mathbb{Z}$ we call $H(C, i)$ the *Hilbert function* and

$$H_C(t) = \sum_{x \in C \cap \mathbb{Z}^d} t^{\deg(x)} = \sum_{i \in \mathbb{Z}} H(C, i) t^i$$

the *Hilbert series* of C.

For a pointed cone C we will assume that the generators of $M = C \cap \mathbb{Z}^d$ and thus all elements in $M \setminus \{0\}$ all have positive degree. The theory of Hilbert functions and series is well developed, and therefore, we will in some cases just state the results and refer to [**5**, Ch. 4] and [**4**, Ch. 6] for more background on the topic.

Example 4.2. Let $P = \operatorname{conv}(x_1, \dots, x_n) \subset \mathbb{R}^d$ be a rational polytope. For the cone over P, that is the cone $C = \operatorname{cone}((x_1, 1), \dots, (x_n, 1)) \subset \mathbb{R}^{d+1}$, and the projection to the last component as the grading, the Hilbert function $H(C, i)$ counts the lattice points in the i-th dilation iP. This situation was studied by E. Ehrhart [**22**]. Therefore, in this context often the terms *Ehrhart series* and *Ehrhart function* are used.

Now we want to describe the computation of the Hilbert series of a pointed cone C. In Section 4.4 we will then discuss how we can get the Hilbert function from the

Hilbert series. If $C = \dot{\bigcup}_{i=1}^{m} D_i$ is the disjoint union of $D_1, \dots, D_m$ we directly get

$$H_C(t) = \sum_{i=1}^{m} H_{D_i}(t).$$

Such a disjoint decomposition of C can be obtained from a triangulation, which is a non-disjoint decomposition of C, using the order vector criterion as described in Section 2.5. Then the D_i are simplicial cones with some facets excluded. The Hilbert series for such objects is the topic of the next section.

4.2. Hilbert series of a semi-open simplicial cone

In this section D is a semi-open simplicial cone in which some facets are excluded. Our goal is it to compute $H_D(t)$.

In a simplicial cone σ generated by $x_1, \dots, x_d \in \mathbb{R}^d$ we set the facet opposite of x_i to $F_i = \operatorname{cone}(\{x_1, \dots, x_d\} \setminus \{x_i\})$. To exclude a union of facets S we define

$$\varepsilon(y) = \sum_{i:\, y \in F_i \subset S} x_i$$

which is used to shift y outside of S.

Theorem 4.3. *Let σ be a simplicial cone generated by $x_1, \dots, x_d \in \mathbb{Z}^d$, S a union of some facets of σ and $D = \sigma \setminus S$. The Hilbert series of D is given by the rational function*

$$H_D(t) = \frac{\sum_{y \in E} t^{\deg(y+\varepsilon(y))}}{(1-t^{k_1}) \cdots (1-t^{k_d})}$$

with $E = \{q_1x_1 + \cdots + q_dx_d \;:\; 0 \le q < 1\} \cap \mathbb{Z}^d$ and $k_i = \deg(x_i)$. If not all facets are excluded, the degree of H_D as a rational function is negative.

Proof. Theorem 3.5 gives us the disjoint union

$$\sigma \cap \mathbb{Z}^d = \dot{\bigcup_{y \in E}} y + N$$

where N is the free monoid generated by $x_1, \dots, x_d$. Now we have to exclude S from the components $y + N$. The intersection of $y + N$ with the single facet F_i is empty if $y \notin F_i$ and otherwise $(y+N) \cap F_i = \{y + a_1x_1 + \cdots + a_dx_d : a_1, \dots, a_d \in \mathbb{Z}_+, a_i = 0\}$. Excluding these intersections for all $F_i \subset S$ gives the set

$$(y+N) \setminus S = \{y + a_1x_1 + \cdots + a_dx_d : a_1, \dots, a_d \in \mathbb{Z}_+, a_i > 0 \text{ for all } i : \, y \in F_i \subset S\}$$

which can also be interpreted as shifting $y + N$ by $\varepsilon(y)$,

$$(y+N) \setminus S = y + \varepsilon(y) + N.$$

Now the computation of the Hilbert series of $D = \dot{\bigcup}_{y \in E} y + \varepsilon(y) + N$ is easy. It is well known that the free monoid N has the Hilbert series

$$H_N(t) = \frac{1}{(1-t^{k_1}) \cdots (1-t^{k_d})}$$

with $\deg(x_i) = k_i$, see e.g. [4]. In the shifted version $y+\varepsilon(y)+N$ we have to multiply by $t^{\deg(y+\varepsilon(y))}$, hence we get the rational function $H_D(t)$ as in the theorem.

For the condition on the degree of $H_D(t)$ note that the coefficients in the linear combination $y + \varepsilon(y) = q_1x_1 + \cdots + q_dx_d$ satisfy $q_i \le 1$ for all i and $q_i = 1$ if and only if $y \in F_i \subset S$. If not all facets are excluded, we have $\deg(y + \varepsilon(y)) < \deg(x_1 + \cdots + x_d) = k_1 + \cdots + k_d$ and the degree of $H_D(t)$ as a rational function is negative. □

Note that Remark 2.9 shows that the order vector criterion does not exclude all facets of a simplicial cone, therefore $H_D(t)$ will have a negative degree in our applications.

The efficient creation of the points in E has already been discussed in Section 3.3. The only relevant information on the points in E is the degree and the degree of the shift. To compute those we do not need to know their representation in standard basis, the vector b from (1) in Section 3.3 with respect to the basis $\frac{1}{\delta}x_1, ..., \frac{1}{\delta}x_d$ is sufficient. With this information we can compute

$$\deg(y) = \frac{1}{\delta}\sum_{i=0}^{d} b_ik_i \quad \text{and} \quad \deg(\varepsilon(y)) = \sum_{i:\, y\in F_i\subset S} k_i. \tag{2}$$

Example 4.4. We continue Examples 3.7 and 3.10 with the computation of the Hilbert series of the semi-open cone D where the facet $F_1 = \operatorname{cone}(x_2)$ is excluded. As grading we use the total degree $\deg(y_1, y_2) = y_1 + y_2$, thus $k_1 = \deg(x_1) = 4$ and $k_2 = \deg(x_2) = 3$. To apply formula (2) one has to take the scalar products of the b vectors with the vector of degrees $(4, 3)$ and divide the result by $\delta = 5$. To exclude F_1 means to shift those vectors with $b_1 = 0$. In this example only $(0, 0)$ has to be shifted by x_1 and $\deg(\varepsilon(0, 0)) = \deg(x_1) = 4$.

b	$(0,0)$	$(4,3)$	$(3,1)$	$(2,4)$	$(1,2)$
$\deg(y)$	0	5	3	4	2
$\deg(\varepsilon(y))$	4	0	0	0	0
$\deg(y+\varepsilon(y))$	4	5	3	4	2

Using Theorem 4.3 we get

$$H_D(t) = \frac{t^2 + t^3 + 2t^4 + t^5}{(1 - t^4)(1 - t^3)}.$$

4.3. Collecting the Hilbert series

In subsumption, for the cone C with triangulation Γ and monoid $M = C \cap \mathbb{Z}^d$ we obtain M as a disjoint decomposition

$$M = \dot{\bigcup_{\sigma\in\Gamma}} \dot{\bigcup_{y\in E_\sigma}} (y + \varepsilon(y))M_\sigma,$$

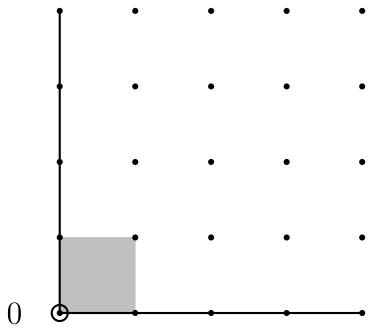
where the M_σ are free monoids of rank d, generated by the extreme integral generators of σ. Such a disjoint union is called a *Stanley decomposition*, named after R. Stanley who proved its existence in 1982 [**34**].

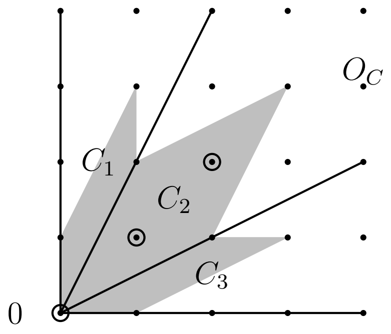
For the Hilbert series this implies

$$H_C(t) = \sum_{\sigma\in\Gamma} H_{D_\sigma}(t) = \sum_{\sigma\in\Gamma} \frac{\sum_{y\in E_\sigma} t^{\deg(y+\varepsilon(y))}}{(1-t^{k_1})\cdots(1-t^{k_d})}.$$

So we now have to sum all the $H_{D_\sigma}(t)$ up to get $H_C(t)$. In the homogeneous case, i.e. all generators have the same degree, all denominators are the same and the task is easy. But otherwise we have to extend the rational functions in order to add them.

Example 4.5. Consider the positive orthant $C = \operatorname{cone}((1,0),(0,1))$ in $\mathbb{R}^2$ with lattice $\mathbb{Z}^2$ and the standard grading $\deg(y_1,y_2) = y_1+y_2$. Here we easily see that the Hilbert series is $1/(1-t)^2$. To show how the algorithm works we will compute the series in a different, more complicated way. We use the lexicographical triangulation with the vertices $x_1 = (0,1), x_2 = (1,2), x_3 = (2,1), x_4 = (1,0)$ which consists of the simplicial cones $C_1 = \operatorname{cone}(x_1,x_2), C_2 = \operatorname{cone}(x_2,x_3), C_3 = \operatorname{cone}(x_3,x_4)$.





Let the order vector O_C be inside the interior of C_2, e.g. $O_C = (4,3)$. Then in D_2 no facet of C_2 is excluded, the elements in E are $(0,0),(1,1),(2,2)$ and both generators have degree 3. In D_1 the facet to C_2 is excluded so we must move $(0,0)$ out of the facet to $(1,0)$. D_3 is symmetric to D_1. Thus we get

$$H_{D_1}(t) = H_{D_3}(t) = \frac{t}{(1-t)(1-t^3)}, \qquad H_{D_2}(t) = \frac{1+t^2+t^4}{(1-t^3)^2},$$

$$H_C(t) = H_{D_1}(t) + H_{D_2}(t) + H_{D_3}(t) = \frac{1+t+t^2-t^3-t^4-t^5}{(1-t^1)(1-t^3)^2} = \frac{1}{(1-t)^2}.$$

The denominator for a semi-open simplicial cone depends only on the degrees of the generators, and typically there are not so many different degrees occurring. Therefore the denominators are the same for multiple semi-open simplicial cones. To avoid the extension of the rational functions for each addition we collect the summands for each denominator separately and add these classes of denominators at the end.

Example 4.6. The 24-dimensional example P_4 from Section 5.3 has 3928 generators in the seven different degrees 2, 3, 4, 6, 8, 10 and 12. The size of the computed triangulation is above $3 \cdot 10^{11}$ but only 3600 different denominators appear.

As said before, the rational functions must be extended in order to add them up. We do not extend to the least common multiple of the denominators but to a common multiple which is a product of terms $1 - t^k$. This is advantageous, since the multiplication with $1 - t^k$ is cheap. So we end up with a representation

$$H_C(t) = \frac{f(t)}{(1-t^{k_1})\cdots(1-t^{k_u})} \tag{3}$$

with some $f \in \mathbb{Z}[t]$ and $u \geq d$.

In Example 4.5 the result of the summation could be simplified to $1/(1-t)^2$. This is a nice and simple representation, but in general it is not obvious which is the best representation.

One possibility of course is to cancel as far as possible to get coprime numerator and denominator. Recall that $t^k - 1$ is the product of all cyclotomic polynomials Φ_i where i divides k and the i-th cyclotomic polynomial Φ_i is the normed, irreducible polynomial (in $\mathbb{R}[X]$) whose roots are the i-th primitive roots of unity. Thus, after cancellation, the denominator will be a product of cyclotomic polynomials and we can write

$$H_C(t) = \frac{g(t)}{\Phi_{i_1}^{l_1} \cdots \Phi_{i_s}^{l_s}} \tag{4}$$

with numerator $g \in \mathbb{Z}[t]$ that is coprime to $\Phi_{i_1}, \ldots, \Phi_{i_s}$. This representation is easy to compute and the smallest in terms of the degrees of the numerator and denominator polynomials.

Another possible representation of the Hilbert series is the following. For each simplicial cone σ the computed denominator of $H_{\sigma \setminus S_\sigma}(t)$ is $\prod_{i=1}^d (1 - t^{k_i})$ where k_i is the degree of the i-th generator of σ. Let π be the least common multiple of the degrees e_i of the generators of C. By extending with suitable cyclotomic polynomials we can get $(1 - t^\pi)^d$ as the denominator of $H_{\sigma \setminus S_\sigma}(t)$ and therefore also for $H_C(t)$. This shows:

Theorem 4.7. *Let C be a pointed rational cone whose generators have positive degrees $e_1, \ldots, e_n$ and $\dim C = d$. Then there exists a polynomial $h \in \mathbb{Z}[t]$ and $k_1, \ldots, k_d > 0$ which divide $\operatorname{lcm}(e_1, \ldots, e_n)$, such that*

$$H_C(t) = \frac{h(t)}{(1-t^{k_1})\cdots(1-t^{k_d})}.$$

The degree of $H_C(t)$ as a rational function is negative.

One possibility for the denominator is $(1 - t^\pi)^d$ as just explained. But there are other choices. We construct such a representation from the one with cyclotomic denominator $\Phi_{i_1}^{l_1} \cdots \Phi_{i_s}^{l_s}$ as in (4). For $j = 1, \dots, d$ let $I_j = \{i_j : l_j \geq j\}$ and $k_j = \operatorname{lcm}(I_j)$. Then $\prod_{i \in I_j} \Phi_i$ divides $(1 - t^{k_j})$. Therefore $(1 - t^{k_1}) \cdots (1 - t^{k_d})$ is a valid choice for the denominator.

Example 4.8. The cone C_2 in Example 4.5 has the Hilbert Series

$$H_{C_2}(t) = \frac{1 + t^2 + t^4}{(1 - t^3)^2} = \frac{1 - t + t^2}{\Phi_1^2 \Phi_3} = \frac{1 - t + t^2}{(1 - t)(1 - t^3)}$$

and the denominator can directly be rewritten as desired.

In general this method will not construct a representation with minimal denominator degree among those fitting the description of Theorem 4.7.

Example 4.9. The Hilbert Series of a subcone of Example 4.6 has in the most canceled form the denominator $\Phi_1^{24}\Phi_2^{23}\Phi_3^{12}\Phi_4^{16}\Phi_6^{12}$. Here different choices are possible to get 24 factors $(1 - t^k)$, but we have to use $k > 6$, otherwise we would get at least $16 + 12 = 28$ factors. Our algorithm produces the degree 175 denominator

$$(1 - t)(1 - t^2)^7(1 - t^4)^4(1 - t^{12})^{12},$$

an alternative with lower degree is $(1 - t^4)^{12}(1 - t^6)^8(1 - t^{12})^4$.

4.4. Hilbert quasipolynomial

The values of the Hilbert function for $k \in \mathbb{Z}_+$ are the coefficients in the expansion of the Hilbert series at $t = 0$. We now want to show how to compute the Hilbert quasipolynomial from the Hilbert series represented as a rational function.

We start with the simpler homogeneous case in which the generators are of degree 1 and later reduce the general case to it. In this case Theorem 4.7 gives us the denominator of the Hilbert series as $(1 - t)^d$. By the following lemma we see that the Hilbert function then is a polynomial.

Lemma 4.10. *Let $a : \mathbb{Z}_+ \mapsto \mathbb{Z}$ be a function such that its power series $A(t) = \sum_{i \in \mathbb{Z}_+} a(i) t^i$ can be represented by*

$$A(t) = \frac{h(t)}{(1 - t)^d},$$

where $h \in \mathbb{Z}[t]$ and $\deg A < 0$.

Then there exists a unique polynomial $P \in \mathbb{Q}[X]$ of degree less than d such that $a(i) = P(i)$ for all $i \in \mathbb{Z}_+$.

Proof. It is enough to consider the case $h(t) = t^w$. It is well known that

$$\frac{t^w}{(1 - t)^d} = t^w \sum_{i=0}^{\infty} \binom{i + d - 1}{d - 1} t^i = \sum_{i=w}^{\infty} \binom{i - w + d - 1}{d - 1} t^i.$$

We choose

$$P(X) = \frac{(X - w + d - 1)(X - w + d - 2) \cdots (X - w + 1)}{(d - 1)!}$$

and have $P(i) = a(i)$ for all $i \in \mathbb{Z}_+$. □

Following the proof of Lemma 4.10, it is easy to see that $P_w(X) = P_0(X - w)$ when P_j is the polynomial for $h(t) = t^j$.

Corollary 4.11. *In the homogeneous case of a cone C with Hilbert series*

$$H_C(t) = \frac{h_0 + h_1 t + \cdots + h_{d-1} t^{d-1}}{(1-t)^d},$$

the Hilbert function is a polynomial of degree $d-1$ with leading coefficient

$$\frac{h_0 + \cdots + h_{d-1}}{(d-1)!}.$$

Proof. Together with Theorem 4.7 and Lemma 4.10 it is only left to see that the potential leading coefficient is not zero. But that is assured since all the h_i are nonnegative in the homogeneous case. □

Back to the general case. Let π be the least common multiple of the degrees $e_1, \ldots, e_n$ of the generators. Every $1 - t^{e_i}$ divides $1 - t^\pi$, hence the Hilbert Series can be written as

$$H_C(t) = \frac{h(t)}{(1-t^\pi)^d},$$

with $h \in \mathbb{Z}[t]$.

Now the Hilbert function is not a polynomial anymore, but a quasipolynomial.

Definition 4.12. A *quasipolynomial* Q of *period* π is a polynomial with periodic coefficients, in other words, a collection of polynomials $Q_0, \ldots, Q_{\pi-1}$ with $Q(i) = Q_j(i)$ when $i \equiv j \mod \pi$.

Theorem 4.13. *In the setting of Theorem 4.7 there is a quasipolynomial Q of period dividing $\pi = \operatorname{lcm}(e_1, \ldots, e_n)$ such that*

$$H(C, i) = Q(i), \qquad \textit{for all } i \in \mathbb{Z}_+.$$

Proof. We give a constructive proof that reduces the problem to the homogeneous case. For this purpose we decompose the monoid $M = C \cap \mathbb{Z}^d$ into components $M^j = \{x \in M : \ deg(x) \equiv j \mod \pi\}$ and equip each component a new grading $\deg_j(x) = (\deg(x) - j)/\pi$ for $x \in M^j$. Note that the M^j are not monoids since they are not closed under addition. But they are closed under addition with elements of degree π, hence they are M_π-modules.

The Hilbert function of M is then

$$H(M, i) = H(M^j, \frac{i-j}{\pi}) \quad \text{if } i \equiv j \mod \pi,$$

and the Hilbert series

$$H_M(t) = \sum_{i \in \mathbb{Z}} H(M, i) t^i = \sum_{j=0}^{\pi-1} \sum_{i \in \mathbb{Z}} H(M^j, i) t^{i\pi + j} = \sum_{j=0}^{\pi-1} t^j H_{M^j}(t^\pi).$$

This segmentation of course also holds for the representation of H_M as rational function with denominator $(1-t^\pi)^d$ because multiplication with $(1-t^\pi)$ does not cause interferences between the π summands. Conversely we know from Theorem 4.7 that the Hilbert Series can be expressed as

$$H_M(t) = \frac{h(t)}{(1-t^\pi)^d},$$

with $h \in \mathbb{Z}[t]$ and $\deg(h) < d\pi$. We decompose $h(t) = \sum_{i=0}^{d\pi-1} h_i t^i$ into $g'_j(t) = \sum_{i=0}^{d-1} h_{i\pi+j} t^{i\pi+j}$ and get

$$t^k H_{M^j}(t^\pi) = \frac{g'_j(t)}{(1-t^\pi)^d}.$$

Dividing by t^j and then substituting t^π by t (this transformation corresponds to the regrading), yields

$$H_{M^j}(t) = \frac{g_j(t)}{(1-t)^d}, \qquad \text{with } g_j(t) = \sum_{i=0}^{d-1} h_{i\pi+j} t^i.$$

Now apply Lemma 4.10 to get the $H(M^j, X)$ as a polynomial $P''_j(X)$, so that $H(M,i) = P''_j(\frac{i-j}{\pi})$ when $i \equiv j \mod \pi$. In order to undo the regrading and get the Hilbert quasipolynomial of M we apply two substitutions to $P''_j(X)$. First substitute X by X/π and on the resulting polynomial P'_j apply the linear substitution $X \mapsto X - j$ to obtain P_j such that $P_j(i) = P''_j(\frac{i-j}{\pi})$.

The Hilbert quasipolynomial of M is then Q with $Q(i) = P_j(i)$ when $i \equiv j \mod \pi$. □

In the proof we outlined a way to compute the Hilbert quasipolynomial. Algorithm 6 describes this computation using only integer arithmetic. The function LinearSubstitution uses the well-known Horner's rule (see e.g. [**41**]) in the special case of evaluating a polynomial at $X - a$.

Example 4.14. We compute the Hilbert quasipolynomial for the cone C_2 of Example 4.5. Its Hilbert Series is

$$H_{C_2}(t) = \frac{1-t+t^2}{(1-t)(1-t^3)} = \frac{1+t^2+t^4}{(1-t^3)^2}.$$

The algorithm produces $g_0(X) = 1$, $g_1(X) = X$, $g_2(X) = 1$ and $P''_0(X) = P''_2(X) = 1 + X$, $P''_1(X) = X$ and after the substitutions

$$\begin{array}{lcrcr} P_0 & = & 1+\frac{1}{3}X & = & 1+\frac{1}{3}X \\ P_1 & = & \frac{1}{3}(X-1) & = & -\frac{1}{3}+\frac{1}{3}X \\ P_2 & = & 1+\frac{1}{3}(X-2) & = & \frac{1}{3}+\frac{1}{3}X. \end{array}$$

Now we can easily evaluate the Hilbert function at integers:

$$Q(0) = P_0(0) = 1,\ Q(1) = P_1(1) = 0,\ Q(2) = P_2(2) = 1,\ Q(3) = P_0(3) = 2,\ \ldots$$

Algorithm 6 Compute the Hilbert quasipolynomial

```
Require: $\pi, d \in \mathbb{Z}_+$, $h = (h_0, \dots, h_{d\pi-1})$
Return: the coefficients of $(d-1)!\cdot\pi^{d-1}\cdot Q$
 1: function HilbertQuasipolynomial($h, \pi, d$)
 2:     for $j \leftarrow 0$ to $\pi - 1$ do
 3:         $g_j \leftarrow (h_j, h_{j+\pi}, \dots, h_{j+(d-1)\pi})$
 4:         $p_j \leftarrow$ HilbertPolynomial($g_j, d$)        ▷ $(d-1)!\cdot P_j''$, see Lemma 4.10
 5:         for $i \leftarrow 0$ to $d - 1$ do
 6:             $p_{j,i} \leftarrow \pi^{d-1-i}\cdot p_{j,i}$        ▷ yield coefficients of $(d-1)!\cdot\pi^{d-1}\cdot P_j'$
 7:         $p_j \leftarrow$ LinearSubstitution($p_j, j$)
 8:     return $(p_0, \dots, p_{\pi-1})$

Require: coefficients $f = (f_0, \dots, f_{d-1})$ of a polynomial $F(X) \in \mathbb{Z}[X]$ and $a \in \mathbb{Z}$
Return: the coefficients of $F(X-a)$
 9: function LinearSubstitution($f, a$)
10:     for $step \leftarrow 0$ to $d - 2$ do
11:         for $i \leftarrow d - 2$ to $step$ do
12:             $f_i \leftarrow f_i - a\cdot f_{i+1}$
13:     return $(f_0, \dots, f_{d-1})$
```

We will now generalize Corollary 4.11 to arbitrary $\mathbb{Z}_+$-graded cones.

Theorem 4.15. *Let C be a cone with Hilbert series*

$$H_C(t) = \frac{h_0 + h_1 t + \cdots + h_{d\pi-1}t^{d\pi-1}}{(1-t^\pi)^d}.$$

Then the coefficients in the numerator are nonnegative and $h_0 = 1$. Furthermore, the Hilbert quasipolynomial of C has degree $d-1$ and the constant leading coefficient

$$\frac{h_0 + \cdots + h_{d\pi-1}}{\pi^d(d-1)!}.$$

Proof. According to Theorem 4.3 the coefficients of the numerator polynomial are nonnegative for a semi-open simplicial cone D. Moreover, $h_0(D) = 1$ if and only if the 0 vector is not excluded; this only happens in the start simplicial cone, where no hyperplane is excluded. Otherwise we have $h_0(D) = 0$. In order to change a factor $1 - t^k$ in the denominator to $1 - t^\pi$, the numerator is multiplied by

$$\frac{(1-t^\pi)}{(1-t^k)} = 1 + t^k + t^{2k} + \cdots + t^{\pi-k},$$

which has no influence on the statements about the coefficients.

Since we have required that the group generated by M is $\mathbb{Z}^d$ and the grading is surjective, there is a degree 1 element $v = \sum_{i=1}^n a_i x_i \in \mathbb{Z}^d$ with $a_i \in \mathbb{Z}$ and $x_i \in M$.

Adding a large enough element of M whose degree is divisible by π, for instance $\pi \sum_{i=1}^{n} |a_i|\, x_i$, gives $y \in M$ with degree congruent to 1 modulo π, say $\deg(y) = 1{+}b\pi$.

Consider again the M^j from the proof of Theorem 4.13. The addition with y gives us the following homogeneous sequence of inclusions:

$$M^0 \xhookrightarrow{+y} M^1(-b) \xhookrightarrow{+y} M^2(-2b) \xhookrightarrow{+y} \cdots \xhookrightarrow{+y} M^{\pi-1}(-(\pi-1)b) \xhookrightarrow{+y} M^0(-\pi b).$$

Therefore we have the chain of inequalities of the Hilbert polynomial values

$$H(M^0, i) \le H(M^1, i{+}b) \le H(M^2, i{+}2b) \le \cdots \le H(M^1, i{+}(\pi{-}1)b) \le H(M^0, i{+}\pi b),$$

which shows that all the $H(M^j, i)$ exhibit the same behavior in the limit $i \to \infty$, hence all the $P_j''(X)$ have the same degree and leading coefficient.

The same argument as in Corollary 4.11 shows that the P_j'' are of degree $d-1$ with leading coefficient

$$\frac{s_j}{(d-1)!}, \quad \text{where } s_j = \sum_{i=0}^{d-1} h_{i\pi+j}.$$

So the s_j are the same for all $j = 0, \ldots, \pi-1$ and with $s = \sum_{k=0}^{d\pi-1} h_k = \sum_{j=0}^{\pi-1} s_j$ we have $s_j = s/\pi$. The subsequent substitutions do not change the degree, but the first one, $X \mapsto X/\pi$, multiplies the leading coefficient by $1/\pi^{d-1}$. This completes the proof. $\square$

4.5. Multiplicity and volume

Theorem 4.15 ensures that the leading coefficient of the quasipolynomial is constant. This interesting invariant characterizes the growth of the number of lattice points and is an estimate how "large" the cone is. We will now make this more precise.

Definition 4.16. Let C be a subset of $\mathbb{R}^n$ which spans a linear space of dimension d. We define, if the limit exists,

$$\mu(C) = (d-1)! \cdot \lim_{k\to\infty} \frac{H(C,k)}{k^{d-1}}$$

as the *multiplicity* of C.

Be aware that the multiplicity depends on the lattice, the grading and also the dimension. For a cone C it is $(d-1)!$ times the leading coefficient of its Hilbert (quasi)polynomial. If the cone is generated by degree 1 elements, the multiplicity is an integral value, see Corollary 4.11.

Now let $C \subset \mathbb{R}^d$ be a full-dimensional cone and $P = C \cap A_1$ be the rational polytope at degree 1, in this context we also say $\mu(C)$ is the *normalized volume* of P. This term is justified. It is connected to the usual volume $\mathrm{vol}_{d-1}(P)$ of P in $\mathbb{R}^{d-1}$, which is the Riemann integral $\int_P 1\, dx$. We can compute the integral via approximating P by $(d-1)$-dimensional cubes with edge length $1/k$ and counting the cubes as k goes to infinity. We will use the cubes that sit between neighboring lattice points of the lattice $(1/k)\mathbb{Z}$ and approximate P by those cubes whose lexicographically smallest vertex is in P. This vertex is a lattice point in $(1/k)\mathbb{Z}$. In this way computing the integral becomes a lattice point counting problem. Each cube has volume $(1/k)^{d-1}$, so

$$\operatorname{vol}_{d-1}(P) = \int_P 1\,dx = \lim_{k\to\infty} \frac{\left|P \cap \frac{1}{k}\mathbb{Z}^d\right|}{k^{d-1}} = \lim_{k\to\infty} \frac{\left|kP \cap \mathbb{Z}^d\right|}{k^{d-1}} = \lim_{k\to\infty} \frac{H(C,k)}{k^{d-1}} = \frac{\mu(C)}{(d-1)!}.$$

The multiplicity is normalized to the standard simplicial cone generated by the unit vectors in $\mathbb{R}^d$. This cone has Hilbert series $1/(1-t)^d$ and Hilbert polynomial $H(\mathbb{R}^d_+, k) = (k+1)^{d-1}/(d-1)!$ and therefore normalized volume 1.

Fortunately it is possible to compute the multiplicity without computing the complete Hilbert quasipolynomial.

Lemma 4.17. *Let σ be a simplicial cone generated by elements $x_1, \dots, x_d$ of degrees $k_1, \dots, k_d$, S a union of some facets of σ and $D = \sigma \setminus S$. Furthermore, let G be the matrix with the generators $x_1, \dots, x_d$ as rows. Then we have*

$$\mu(D) = \frac{\det G}{k_1 \cdots k_d}.$$

Proof. By Theorem 4.3, combined with Proposition 3.8, the Hilbert Series of D has coefficient sum $|E| = \det G$ and denominator $(1-t^{k_1})\cdots(1-t^{k_d})$. Following Theorem 4.13 the leading coefficient of the Hilbert quasipolynomial then is

$$\frac{\det G}{\pi^d \cdot (d-1)!} \cdot \frac{\pi}{k_1} \cdots \frac{\pi}{k_d} = \frac{\det G}{k_1 \cdots k_d \cdot (d-1)!}.$$

□

The lemma also shows that the multiplicity is independent of the excluded facets. For a non-simplicial cone again the triangulation is used and the multiplicities of all simplicial cones in the triangulation are added up.

CHAPTER 5

Challenging examples

In this chapter we want to give some challenging examples that have been mastered with the algorithms in this thesis. The first two sections demonstrate the utility of the partial triangulation by examples for which it was previously unknown whether the monoids are normal. In Section 5.3 the volume and the Ehrhart series are used to determine the probabilities of some events discussed in voting theory that are related to Condorcet's paradox. Finally, Section 5.4 demonstrates the improvements of the presented algorithms in conjunction with parallelization.

5.1. Contingency tables

Monoids related to 3-way contingency tables were the motivating examples for the development of the partial triangulation in Section 3.4. Let us give some background.

A $d_1 \times \cdots \times d_r$ *contingency table* is an r-way array of nonnegative integers. We will interpret it as an element of $\mathbb{Z}_+^{d_1 \cdots d_r}$. Contingency tables are used in statistics to represent the sampling result of r discrete random variables $X_1, \ldots, X_r$, where X_j has d_j possible outcomes. The entry at position $i_1, \ldots, i_r$ counts how often the combined outcome has occurred.

Example 5.1. Assume a statistical survey asks persons for their favorite color (out of four) and their age (divided into three age groups). The result of such a survey would be a 4×3 contingency table. If additionally the sex is recorded, the result could be the $4 \times 3 \times 2$ contingency table 5.1.

	male			female		
	age$_1$	age$_2$	age$_3$	age$_1$	age$_2$	age$_3$
blue	14	7	8	11	15	3
red	2	11	4	13	3	2
orange	8	12	2	10	10	5
green	2	2	10	4	11	0

Table 5.1. A $4 \times 3 \times 2$ contingency table.

A *marginal distribution* of a contingency table to the index set $F \subset \{1, \ldots, r\}$ is the contingency table considering only the random variables X_j with $j \in F$.

This corresponds to summing over the dropped random variables. In the examples considered here, only one random variable is dropped and every resulting count is the summation along every line, leading to an $(r-1)$-way contingency table. The *monoid of possible marginal distributions by taking line sums* of $d_1 \times \cdots \times d_r$ contingency tables is given by all these line sums to the index sets $F_j = \{1, \ldots, r\} \setminus \{j\}$ for $j = 1, \ldots, r$.

Example 5.2. For 2-way contingency tables the marginal distribution by taking line sums is the combination of the two 1-way contingency tables which are the row sums and the column sums.

The marginals by taking line sums of the $4 \times 3 \times 2$ table from example 5.1 are the combination of the three tables in Table 5.2.

	age_1	age_2	age_3
blue	25	22	11
red	15	14	6
orange	18	22	7
green	6	13	10

	male	female
blue	29	29
red	17	18
orange	22	25
green	14	15

	male	female
age_1	26	37
age_2	32	38
age_3	24	10

TABLE 5.2. The marginals by taking line sums of Table 5.1.

The contingency tables of a fixed size form a monoid which is generated by the tables with a single entry 1. Their marginal distributions are 0-1-vectors which have a single entry 1 in each of the r components. They generate the monoid of possible marginal distributions. Therefore, the monoid, derived from $a \times b \times c$ contingency tables by taking line sums, is generated by $a \cdot b \cdot c$ vectors in dimension $a \cdot b + a \cdot c + b \cdot c$. In this thesis such monoids were used multiple times as test examples and denoted by Aabc.

Example 5.3. The monoid derived from 3×2 contingency tables by taking line sums is generated by the following 6 generators.

$$\begin{array}{ccc|cc}
1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1
\end{array}$$

Hibi and Ohsugi examined in [**30**] whether these monoids are normal. If such a monoid is normal, it implies that the possible marginal distributions are only restricted by linear equations and inequalities, and all integral tables inside the cone defined by the constraints are actually marginals of contingency tables.

The normality classification of Hibi and Ohsugi [**30**, Theorem 6.4] left the cases $4 \times 4 \times 3$, $5 \times 4 \times 3$ and $5 \times 5 \times 3$ open. The computations of R. Hemmecke and

M. Köppe [**6**] and those presented in Section 3.4 independently showed they are all normal. Therefore we have the following theorem.

Theorem 5.4. *Let $d_1 \geq \cdots \geq d_r \geq 2$ be integer numbers. The monoid of taking line sums of $d_1 \times d_2 \times \cdots \times d_r$ contingency tables is normal if and only if its size is*

(i) $d_1 \times d_2$, or $d_1 \times d_2 \times 2 \times \ldots \times 2$, or
(ii) $d_1 \times d_2 \times 3$ with $d_2 \leq d_1 \leq 5$.

The computation times and additionally data on A553 and A643, which is not normal, are given in Table 3.2.

5.2. Cut monoids of graphs

In this section we will examine the normality of cut monoids of graphs, which are of interest in algebraic statistics.

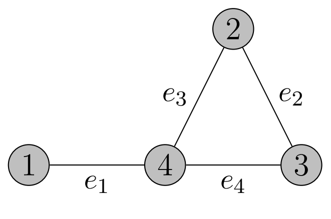
In the following let $\mathcal{G} = (V, E)$ be a simple (without loops or multiple edges) undirected graph on a vertex set V with edges E. We will use the vertex set $V = \{1, \ldots, n\}$ and label the edges $E = \{e_1, \ldots, e_m\}$. We assume the reader is familiar with the basic notations in graph theory and refer to [**18**] for more information.

Definition 5.5. A *cut* of $\mathcal{G}$ is a decomposition of the vertex set $V = A \cup B$ into disjoint subsets A and B. Each cut $\{A, B\}$ defines a 0-1-vector $\delta_{\mathcal{G}}(\{A, B\}) \in \mathbb{Z}^{2m}$ where, for $1 \leq i \leq m$,

(i) the i-th entry is 1 if and only if the vertices of edge e_i belong to different sets of the cut, and
(ii) the $(i+m)$-th entry is 1 if and only if the vertices of edge e_i belong to the same set of the cut.

The *cut monoid* $M_{\mathcal{G}}$ is the monoid generated by all $\delta_{\mathcal{G}}(\{A, B\})$.

Example 5.6. Consider the graph with vertex set $V = \{1, 2, 3, 4\}$ and edges $e_1 = \{1, 4\}$, $e_2 = \{2, 3\}$, $e_3 = \{2, 4\}$, $e_4 = \{3, 4\}$, as pictured below. Its cut monoid is generated by the eight vectors $\delta_{\mathcal{G}}(\{A, B\})$ listed in the table below.



A in cut $\{A, B\}$	$\delta_{\mathcal{G}}(\{A, B\})$							
$\emptyset$	0	0	0	0	1	1	1	1
$\{1\}$	1	0	0	0	0	1	1	1
$\{2\}$	0	1	1	0	1	0	0	1
$\{3\}$	0	1	0	1	1	0	1	0
$\{4\}$	1	0	1	1	0	1	0	0
$\{1, 2\}$	1	1	1	0	0	0	0	1
$\{1, 3\}$	1	1	0	1	0	0	1	0
$\{2, 3\}$	0	0	1	1	1	1	0	0

It is easy to see that the cut monoid of a graph with n vertices and m edges has $2^{(n-1)}$ generators in $\mathbb{Z}^{2m}$. The second half of the cut vectors is the same as the first half with all bits flipped. This implies that the generators are homogeneous in degree m with respect to the total grading. Moreover we get an embedding of the cut monoid into $\mathbb{Z}^{m+1}$ given by $(a_1, \dots, a_{2m}) \mapsto (a_1, \dots, a_m, (a_1 + \dots + a_{2m})/m)$.

In the literature cut monoids appear in terms of their *cut ideals* [**37**]. Also the *cut cone* and the *cut polytope* which are generated by the first halves of $\delta_{\mathcal{G}}(\{A, B\})$, are well-studied, e.g. they are central objects in [**16**]. Here it is important to differentiate between the "non-homogeneous" cut cone and the cone over the cut polytope, which we will discuss. It is known that the cut polytope is full-dimensional, which implies that the cut monoid has indeed rank $m + 1$.

Sturmfels and Sullivant ([**37**]) stated the following very interesting conjecture. It uses the concept of a *minor* of a graph $\mathcal{G}$, that is a graph which can be formed from $\mathcal{G}$ by deleting edges and vertices and by contracting edges.

Conjecture 5.7. *The cut monoid of a graph $\mathcal{G}$ is normal if and only if $\mathcal{G}$ is free of K_5 minors.*

Here K_5 is the complete graph on 5 vertices. A computation with Normaliz showed that the cut monoid of K_5 is not normal. In [**6**] we validated the conjecture for all graphs up to 8 vertices directly by checking the normality of all cut monoids of graphs without K_5 minors and at most 8 vertices.

It is known that the following operations on a graph $\mathcal{G}$ preserve normality of the cut monoid:

(i) deletion of a vertex with all attached edges, equivalently, taking an induced subgraph [**37**, Lemma 3.2 (1)],
(ii) contraction of an edge [**37**, Lemma 3.2 (2)],
(iii) deletion of an edge [**29**, Theorem 2.3],
(iv) taking a minor (follows from (ii) and (iii)).

Furthermore, Ohsugi showed that the cut monoid of the *clique sum*, also called *k-sum*, for $k = 0, 1$ or 2 is normal if and only if the cut monoids of the summands are normal [**29**, Theorem 3.2]. This allows to utilize a decomposition result from graph theory, that characterizes the edge-maximal graphs with at least 3 vertices and without K_5 minors as 1 or 2 sums of K_3, K_4, 4-connected plane triangulations, and the Wagner graph, see Diestel [**17**, p. 181]. The cut monoids of K_3 and K_4 are known to be normal. The Wagner graph is a non-planar graph that is formed from an 8-cycle by adding edges between opposing vertices in the cycle. Ohsugi used Normaliz to show the normality of the Wagner graph's cut monoid and concluded that it is enough to show the following conjecture in order to prove Conjecture 5.7.

Conjecture 5.8. *The cut monoid of a graph $\mathcal{G}$ is normal if $\mathcal{G}$ is a 4-connected plane triangulation.*

Plane triangulations are exactly the edge-maximal planar graphs. These graphs have $m = 3n - 6$ edges if n is the number of vertices, see [**18**, Proposition 4.4.1]. Recall that planar graphs never have a K_5 minor.

To give more evidence for the conjecture we have verified it computationally for graphs with up to $n = 10$ vertices. The result of Ohsugi is here very helpful to reduce the number of graphs to check. There are 85, 767 non-isomorphic, connected graphs without K_5 minor on 9 vertices, but only 4 plane triangulations that are 4-connected. Also note that proving the Conjecture 5.8 for graphs up to n vertices proves the original Conjecture 5.7 for graphs up to n vertices. This allowed us to show computationally the following.

Proposition 5.9. *Let $\mathcal{G}$ be a graph with not more than* 10 *vertices. Then the cut monoid of $\mathcal{G}$ is normal if and only if $\mathcal{G}$ is free of K_5 minors.*

We use the software nauty 2.5 [**28**] to generate all maximal 4-connected planar graphs with n vertices for $n \leq 10$. For $n \leq 5$ there exists no such graph and there are 1 graph each for $n = 6$ and $n = 7$, 2 graphs for $n = 8$, 4 graphs for $n = 9$ and 10 graphs for $n = 10$. The command

```
geng n m:m -C -d4 | planarg > n_maxplanar
```

uses `geng` from nauty to generate all 2-connected graphs with n vertices, $m = 3n-6$ edges and minimal vertex degree of at least 4. Afterwards `planarg` writes only the planar graphs into the file `n_maxplanar`. Then the command

```
labelg n_maxplanar | listg -A n_maxplanar.adj
```

labels them in some canonical way and prints the adjacency matrices to the file `n_maxplanar.adj`.

The correct number of edges ensures that these graphs are maximal planar. There can be graphs in the list that are not 4-connected, but all are at least 2-connected and every vertex has at least 4 edges attached which are necessary conditions. We tested the produced graphs with an own small program and found that for n up to 8 all these were actually 4-connected and for $n = 9$ there was 1 of the 5 generated graphs not 4-connected. Figure 1 shows the 4-connected graphs for $n = 9$.

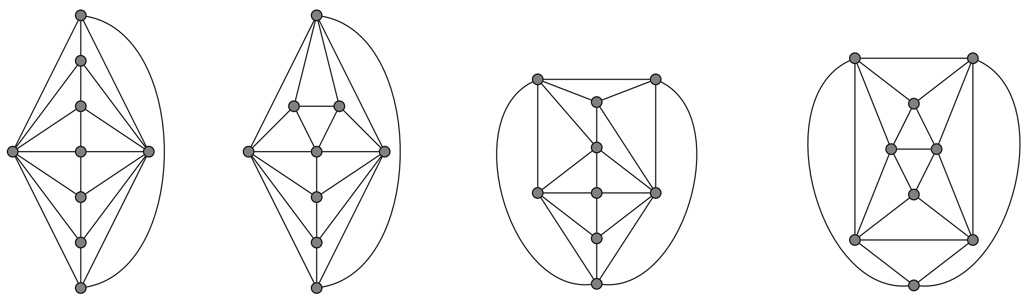


FIGURE 1. The 4-connected plane triangulations with 9 vertices.

To create the input for Normaliz we are going over all cuts $\{A, B\}$ by starting with the empty set A and its incidence vector $(0, \ldots, 0)$, interpreted as a binary representation of a natural number and increasing it by one to go over all subsets. We take only those subsets with at most $n/2$ elements whose complement was not already considered. This ordering of the generators lead for the cut monoids of the 4 open graphs with 9 vertices to triangulations that generate more than $4 * 10^8$

(unique) candidates and more than $3 \cdot 10^9$ evaluated simplicial cones in the partial triangulation with duplicate avoidance, see Section 3.6. The final cones have 100-300 support hyperplanes while during the computation the support hyperplane numbers exceed 10^6. The computation for one example takes more than 16 hours with 20 threads on our Sun Fire X4450.

For these monoids it has proven useful to first compute the support hyperplanes and use them as input to Normaliz. Normaliz will then compute the extreme rays and use them as cone generators. This leads to another ordering of the generators which in these cases gives a much simpler triangulation. The support hyperplane numbers in the process stay below 10^5 and the candidates below 10^7. The preliminary support hyperplane computation for one example takes about 3 hours and the following much easier Hilbert basis computation (including the preceding extreme ray computation) takes only 30 minutes, which in total is much faster than the direct computation.

It is not clear to us how we could find an order of the generators that yields an easy (partial) triangulation and small computation times, but we often observe that some kind of "natural" orderings of the generators give good computation times. Of course this not a mathematical precise formulation and will be an interesting topic for further investigations.

The small number of support hyperplanes at the end and the advantage of the reordering suggest that it might be possible to find an even better ordering of the generators. Tests showed that the following generation rule for the cuts leads to a favorable order of the generators. We go over all subsets with the incidence vector as before but now take all subsets that do not include the last vertex. Then the highest number of support hyperplanes in the computations is $10,792$ and all 4 examples together need just 5 minutes.

With this ordering we were also able to show the normality for the cut monoids of all 10 maximal planar 4-connected graphs with 10 vertices. The biggest of these examples produced a partial triangulation with more than $15 \cdot 10^9$ simplicial cones and almost $7 \cdot 10^8$ candidates for the global reduction. To handle the huge number of candidates we had to make intermediate global reductions. For the intermediate reductions the duplicate free creation of the candidates as described in Section 3.6 was a crucial point. The computation of this example took 30 hours.

One remark on the conjecture. Since the property "having a normal cut monoid" is a minor-closed property, the theorem of Robertson and Seymour [**18**, Theorem 12.5.1] ensures that there is only a finite set of minimal forbidden minors. Since we could not find another forbidden minor with $n \leq 10$ vertices it is quite possible that the conjecture holds. Even if not, it would be possible to reformulate the conjecture with a finite set of graphs that have to be excluded as minors.

5.3. Condorcet's paradox and related examples from voting theory

In this section we present voting schemes that were discussed by Schürmann in [**33**]. We compute the probabilities of interest via normalized volumes of cones and additionally the Ehrhart series with Normaliz.

Consider an election with n candidates and k voters, where every voter i chooses a linear preference ordering $a_{i,1} \succ_i \cdots \succ_i a_{i,n}$ of the candidates. We record the outcome of such an election by counting the multiplicities of the $N = n!$ different orderings in $v = (v_1, \dots, v_N)$.

For a fixed number of candidates n the possible outcomes are the integral points in the cone $\mathbb{R}^N_+$. Those with k votes are exactly the integral points in the polytope

$$A_k \cap \mathbb{R}^N_+,$$

where A_k is the affine hyperplane of degree k elements with respect to the grading $\deg((v_1, \dots, v_N)) = v_1 + \cdots + v_N$. In this perspective the Ehrhart function counts the number of possible outcomes depending on the number of voters.

There are different ways to decide which candidate wins the election. We say candidate a *beats* candidate b when more voters prefer a over b than the other way round, i.e.

$$|\{i : a \succ_i b : i = 1, \dots, k\}| > |\{i : b \succ_i a : i = 1, \dots, k\}|,$$

and a is the *Condorcet winner* if a beats all other candidates. It is named after the Marquise de Condorcet who observed that a Condorcet winner does not have to exist, since the relation "beats" is not transitive in general. The possible non-existence of a Condorcet winner is called *Condorcet's paradox.*

We now want to determine the probability of Condorcet's paradox under the Impartial Anonymous Culture assumption which presupposes that all possible voting results are equally probable. Under this assumption the computation of probabilities translates to counting lattice points. With this in mind let $c_{n,k}(a)$ be the probability that candidate a is the Condorcet winner. Because the candidates are interchangeable and the events are mutually exclusive, the probability that there exist any Condorcet winner is $c_{n,k} = nc_{n,k}(a)$.

We will fix $a = 1$. Let C_n be the semi-open cone, which includes all results with Condorcet winner a. It is cut out in $\mathbb{R}^N_+$ by the $(n-1)$ strict homogeneous inequalities given by the conditions that a beats all other candidates. For $n = 4$ they are the inequalities $\lambda_i(v) > 0$ with the first three linear forms in Table 5.3. The preferences orders are sorted lexicographically, the first column corresponds to $1 \succ 2 \succ 3 \succ 4$ and the last to $4 \succ 3 \succ 2 \succ 1$.

Typically one is interested in the probabilities for a large number of voters. In the limit for $k \to \infty$, the probability of a being the Condorcet winner is

$$c_n(a) = \lim_{k\to\infty} c_{n,k}(a) = \lim_{k\to\infty} \frac{|A_k \cap C_n \cap \mathbb{Z}^N|}{|A_k \cap \mathbb{R}^N_+ \cap \mathbb{Z}^N|} = \lim_{k\to\infty} \frac{H(C_n, k)}{H(\mathbb{R}^N_+, k)} = \frac{\mu(C_n)}{\mu(\mathbb{R}^N_+)} = \mu(C_n).$$

λ_1:	1	1	1	1	1	1	−1	−1	−1	−1	−1	−1	1	1	−1	−1	1	−1	1	1	−1	−1	1	−1
λ_2:	1	1	1	1	1	1	1	1	−1	−1	1	−1	−1	−1	−1	−1	−1	−1	1	1	1	−1	−1	−1
λ_3:	1	1	1	1	1	1	1	1	1	−1	−1	−1	1	1	1	−1	−1	−1	−1	−1	−1	−1	−1	−1
λ_4:	−1	−1	−1	−1	−1	−1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
λ_5:	0	0	0	0	0	0	1	1	1	1	1	1	−1	−1	−1	−1	−1	−1	0	0	0	0	0	0
λ_6:	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	−1	−1	−1	−1	−1	−1

TABLE 5.3. Inequalities for C_4 and P_4

The semi-open cone C_n has the same multiplicity as its closed version $\overline{C_n}$. One way to see this is, that only full dimensional faces can contribute to the leading coefficient of the Hilbert (quasi)polynomial, see also Section 4.5. Therefore we have $c_n(a) = \mu(C_n) = \mu(\overline{C_n})$, and the probability for having a Condorcet winner is

$$c_n = n\mu(\overline{C_n}).$$

For two candidates there always is a Condorcet winner (or a draw, what we can neglect for large k) and for three candidates the probability 15/16 is easy to compute. For $n = 4$ it is more challenging, but Normaliz can now compute it and gives

$$\mu(\overline{C_4}) = \frac{1717}{8192}, \text{ hence } c_4 = \frac{1717}{2048} \approx 0.8384.$$

The cone $\overline{C_4}$ is of dimension 24 and has 234 extreme rays. This computation is done within seconds with Normaliz, see Table 5.5 for more information on the size of the example and computation times.

A simpler and more widely used voting scheme is *plurality voting*. There the voters only vote for their favorite candidate. In other words, the *plurality winner* is the candidate who has more first places in the preference orderings than every other candidate. The *Condorcet efficiency* of plurality voting is the conditional probability that the Condorcet winner is also the plurality winner, under the precondition that a Condorcet winner exists. We compute the complementary probability of a being the Condorcet winner but another candidate b being the plurality winner. The cone P_n containing these outcomes for $a = 1$ and $b = 2$ is a subcone of C_n with the additional $n-1$ inequalities ensuring that 2 wins the plurality vote against all other candidates. For $n = 4$ these are given by λ_4, λ_5 and λ_6 in Table 5.3.

Therefore, the Condorcet efficiency of plurality voting is

$$p_n = \frac{c_n - n(n-1)\mu(P_n)}{c_n}.$$

The computation of the normalized volume for P_4 is significantly more complicated. This cone has 3928 vertices and Normaliz computes

$$\mu(\overline{P_4}) = \frac{36940371852901635506 81491}{2054269543278182400000000 00}.$$

As a result the Condorcet efficiency for four candidates is

$$p_4 = \frac{c_4 - 12\mu(\overline{P_4})}{c_4} = \frac{1065809825501191644931850 9}{14352135440302080000000000} \approx 0.7426.$$

In *plurality voting versus plurality cutoff* two rounds of plurality voting take place. The first two candidates of the first round get into a run-off election. In this voting scheme one asks for the probability that the second placed candidate of the first round wins the run-off vote. We will work with the cone V where we fix that candidate 1 wins in the plurality vote (i.e. has the most first places), 2 is second, and in general candidate a gets ranked a-th; but 2 wins the run-off vote. In the case of four candidates the inequalities that are additional to those of $\mathbb{R}_+^N$ are given in Table 5.4.

1	1	1	1	1	1	−1	−1	−1	−1	−1	−1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	−1	−1	−1	−1	−1	−1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	1	1	1	1	1	1	−1	1	−1	1	−1	1	−1	1	−1	1	−1	1

TABLE 5.4. Inequalities for V_4

In this case the cone has 1872 generators and normalized volume

$$\mu(V_4) = \frac{2988379676768359}{292162779488452608}.$$

Since the outcome of the first plurality vote was fixed, we have to multiply this value by $n!$ to get the total probability that the winner of the first round looses the run-off election. In this case it is $24 \cdot \mu(V_4) \approx 0.2455$.

All these three values have been determined also in other ways, see [**33**] for references. Normaliz can even compute the complete Ehrhart series for the closures of these cones. The size of the triangulation and the needed computation times are presented in Table 5.5. The computations are done with the recent development version of Normaliz, see the next section. Only the Ehrhart series computations of $\overline{P_4}$ and $\overline{V_4}$ are done with earlier versions and not redone because of the high computation times.

	number of		computation times	
	extreme rays	simplicial cones	volume	Ehrhart series
$\overline{C_4}$	234	1,344,671	3 sec	4 sec
$\overline{P_4}$	3,928	361,150,173,336	78:14 h	218:14 h
$\overline{V_4}$	1,872	257,744,341,008	59:58 h	175:11 h

TABLE 5.5. Cones of voting events (times on x4450 with 20 threads)

The order of the generators changes the triangulation and, like in the cut monoid examples from the previous section, this has a considerable influence on the computation times of these examples. We have observed that a good ordering is obtained by sorting the generators by degree, but within a degree keeping the input ordering. Therefore we have added the sorting by degree as a general preliminary step in Normaliz.

Schürmann suggested in [**33**] to exploit the symmetry in the examples to simplify the computations. This can reduce the dimension significantly, but instead of "simply" counting the lattice points, they have now to be counted with the right weight, which is given by a polynomial. For this kind of generalized Ehrhart series computations we have developed NmzIntegrate, an offspring of Normaliz, see [**11**].

5.4. Computation time improvements

This section contains computation time comparisons to document the striking effect of the presented algorithms. For this purpose we compare two versions of Normaliz. The version 2.2 reflects the state of development before the author joined the development team. We compare it with the recent development version from November 21, 2013, available at github[1].

Besides significant performance gains also the functionality was extended. Additional input possibilities have been implemented: congruences to define a lattice, and combinations of inequalities, equations and congruences. A very important extension is the possibility to use arbitrary $\mathbb{Z}$-gradings to compute the multiplicity and Hilbert series, which beforehand was only possible for monoids that were generated in degree 1. This was for example necessary to compute the multiplicities and Ehrhart series for the voting schemes in the previous section. Furthermore, Normaliz was connected to the computer algebra system CoCoA [**15**] via its library CoCoALib [**2**].

	Hilbert basis		Hilbert Series		both	
	v 2.2	now	v 2.2	now	v 2.2	now
small	7.2 sec	2.8 sec	3.5 sec	0.3 sec	17.0 sec	3.2 sec
medium	10.1 sec	2.3 sec	25.8 sec	1.4 sec	23.9 sec	1.8 sec
big	3:02 min	3:18 min	—	1.4 sec	—	3:25 min
huge	4:52 min	2:58 min	7 sec	0.7 sec	3:00 min	2:23 min
A443	52 min	0.7 sec	51 min	49.9 sec	54 min	49.8 sec
A543	>3 days	39.9 sec	—	43 min	—	43 min

TABLE 5.6. Timing comparisons between Normaliz version 2.2 and the recent development version (serial on x4450)

[1]https://github.com/csoeger/Normaliz

The timing comparisons are listed in Table 5.6. The first four examples are from the Normaliz distribution, additionally two contingency table monoids are used. All computations are done using 64 bit arithmetic. The Hilbert series computation of "big" fails in version 2.2 with the 64 bit arithmetic because of an overflow during the lifting, see [**9**] for details on the lifting process. It is doable with higher precision, but we do not include the time since it is not suitable for comparisons.

It is possible to compute the Hilbert basis of A543 with version 2.2, but it takes some days and more than 50 GB of RAM. The Hilbert series computation was not possible, because during the lifting, the number of hyperplanes explodes and quickly exceed the 128 GB memory on our system. In contrast to this, Normaliz now needs only 50 MB for the Hilbert basis computation and 600 MB for the Hilbert series.

In the successful computation of previously unreachable examples the parallelization plays an important role. The theoretical aspects of the parallel algorithms have already been discussed. On the implementation side, we use the OpenMP programming interface [**31**]. It offers the possibility to make only small changes to the source code and reduces the technical difficulty for the programmer. Nevertheless, one has to exercise caution in the implementation to avoid performance problems. As compiler we use g++ from the GNU Compiler Collection [**19**], which implements the OpenMP API.

Table 5.7 shows the times for serial execution, with 4 threads, and with 20 threads for some examples with acceptable serial time. Furthermore, the efficiency of the parallelization, which is the factor of speedup per used thread, is given.

		serial	4 threads		20 threads	
	computation	time	time	effic.	time	effic.
A553	Hilbert basis	213:26 min	50:03 min	1.07	10:17 min	1.04
A543	both	43:24 min	12:32 min	0.87	3:04 min	0.71
lo6 facet	Hilbert series	21:17 min	7:49 min	0.68	2:25 min	0.44

TABLE 5.7. Parallelization efficiency (on x4450)

In the A553 computation most of the time is spent in the global reduction. Normally an efficiency of 1 would be optimal. But in this example, and to some extent also in others, the global reduction profits overproportionally from the parallelization. From the algorithmic point of view there is no reason identifiable, see Algorithm 4. Most probably it is due to some internal optimizations that happen to work more efficiently with parallelization.

In the other two examples the time is almost completely spent with triangulating the cone and evaluating the triangulation. For A543 the efficiency is still very good, especially considering that the efficiency of our x4450 system is only about 0.9 when running 20 serial computations at the same time. In the example "lo6 facet" we observe lower efficiency. One reason is that it is simply too small to profit from high levels of parallelization, for example the number of hyperplanes does not exceed 12,000. Still the parallelization reduces the execution time of this example by a factor of 8.8.

The combination of algorithmic improvements and parallelization already helped to solve open problems, like the normality of monoids of marginal distributions of contingency tables, and gave evidence for open conjectures, like Conjecture 5.7. They also provide the possibility to increase the range of examples that can be computed, and in this way support mathematicians in their research. In conclusion we think these algorithms have proved to be very useful and successful.

CHAPTER 6

Bounds

Normaliz provides two different possibilities for the used integer type. One is the (at least) 64 bit C++ integer type `long long`, the other is the arbitrary precision type `mpz_class` from the GMP library [**40**]. Table 3.1 demonstrates how big the advantage of machine integers over multiple precision integers is; machine integers can be faster by a factor of 10 or more. Of course we can only use them if the numbers in the calculations are small enough to fit into the 64 bit range.

Normaliz offers the possibility to perform checks for arithmetic overflow (via command line option `-e`). In this case for most calculations one of the following two checks are performed.

(i) The calculation is also done modulo a prime p and the results are compared modulo p, e.g. for scalar products.
(ii) The result of the computation is validated directly, e.g. for solutions of systems of linear equations it is checked if the result indeed solves the system.

Since all algorithms have been detailed in this work we now want to examine how big the integers in the algorithms can grow and give bounds that can be used to decide whether it is safe to use machine integers. And also where, and where not, tests for overflow are necessary to ensure correct results.

6.1. Bounds for the determinants

For a vector $y = (y_1, \dots, y_d) \in \mathbb{R}^d$ we will use the 2-norm $\|y\|_2 = \sqrt{y_1^2 + \cdots + y_d^2}$ and the maximum norm $\|y\|_\infty = \max\{|y_1|, \dots, |y_d|\}$. In order to take the norm of a matrix we will consider the matrix as a single long vector.

As we saw in Section 3.3 the determinant δ of a $d \times d$ matrix G of generators of a simplicial cone is an important size since most computations in that simplicial cone can be done modulo δ. The size of the determinant can easily be bounded by

$$|\det G| \le d! \cdot \|G\|_\infty^d ,$$

using for example the Leibniz formula. Two better bounds are given by *Hadamard's inequality*, see e.g. [**41**, Theorem 16.6].

Theorem 6.1. *Let $A \in \mathbb{R}^{d\times d}$ be a matrix with row vectors $x_1, \dots, x_d \in \mathbb{R}^d$. Then*

$$|\det A| \le \|x_1\|_2 \cdots \|x_d\|_2 \le d^{\frac{d}{2}} \|A\|_\infty^d .$$

The prove of the first inequality uses the Gram-Schmidt orthogonalization and the second one is the simple estimate $\|x_i\|_2 \leq \sqrt{d}\,\|x_i\|_\infty$.

Consider the cone generated by $x_1, \ldots, x_n \in \mathbb{Z}^d$. For the further analysis we define the maximal entry $M = \max\{\|x_1\|_\infty, \ldots, \|x_n\|_\infty\}$ and 2-norm of the generators $N_i = \|x_i\|$ for $i = 1, \ldots, n$. In this chapter we will assume that $N_1 \geq N_2 \geq \cdots \geq N_n$. The absolute values of all determinants that can occur in the course of the computation can be bounded with Hadamard's inequality by $D = N_1 \cdots N_d$, the product of the d largest norms of the generators.

Example 6.2. In Table 6.1 we compare the presented determinant bounds for the random examples that were also used in 3.3, random7x70 which are 70 generators in dimension 7 with random entries from 0 to 9, and the $4 \times 4 \times 3$, $5 \times 5 \times 3$ and $6 \times 4 \times 3$ contingency tables.

example	dim.	$d!M$	$d^{\frac{d}{2}}M$	$D = N_1 \cdots N_d$
random7	7	35 bits	32 bits	27 bits
random7x70	7	35 bits	32 bits	29 bits
random12	12	67 bits	60 bits	50 bits
random20	20	125 bits	107 bits	91 bits
A443	30	108 bits	74 bits	33 bits
A553	43	176 bits	117 bits	54 bits
A643	42	170 bits	114 bits	48 bits

TABLE 6.1. Comparison of determinant bounds

Note that the values are taken after the transformation into a full-dimensional cone. For the contingency table examples in the original lattice we have $\|v_i\|^2 = 3$. After transforming into lower dimension the norm of the vectors is not 3 anymore, but the bound for the determinant is still smaller because of the smaller dimension.

In practice these bounds are still far of from the actual values most of the time. In random7x70 the biggest determinant in the full triangulation is 1624079, which has 8 bit less than the Hadamard bound. For A443 this is more extreme, there biggest determinant is 2.

The Hadamard bound is significantly better than the other bounds and its computation is very cheap in comparison to the steps that follow it. Therefore we will always use it.

With this knowledge we can already bound the norms of most vectors in the computation. All vectors v that appear as candidates for the Hilbert basis are from some set E as in Theorem 3.5, i.e. they are of the form $v = \sum_{i=1}^{d} q_i x_{j_i}$ with $0 \leq q_i < 1$. Hence we have

$$\|v\|_2 < N_1 + \cdots + N_d \leq dN_1 \text{ and } \|v\|_\infty < dM.$$

The coefficients of s and b in the computation of the v, see Section 3.3, are reduced modulo the determinant, so especially bounded by D.

6.2. Bounds for the support hyperplanes

Now we have a look at vectors in the dual space, namely the linear forms λ of support hyperplanes. One possibility to compute these is to solve the system $G\lambda'' = e_i$ with some unit vector e_i and matrix of generators G as in Section 2.1. Then *Cramer's rule* gives the solution as $\lambda''_j = (\det G_{i,j})/(\det G)$ where $G_{i,j}$ is obtained from G by deletion of row i and column j. In the algorithm from Section 2.1 we compute the integer version $\lambda' = \det G \cdot \lambda'' = (\det G_{i,j})_{j=1,\dots,d}$ and afterwards divide by the greatest common divisor of the entries to get the coprime version $\lambda = \lambda'/(\gcd(\lambda'))$.

Even if we do not use Cramer's rule in practice, we have $\left|\lambda'_j\right| = |\det G_{i,j}| \leq N_1 \cdots N_{d-1}$. This bounds the norms by

$$\|\lambda'\|_\infty \leq N_1 \cdots N_{d-1} \text{ and } \|\lambda'\|_2 \leq \sqrt{d} \cdot N_1 \cdots N_{d-1}.$$

These bounds certainly also hold for every support hyperplane λ with coprime entries that may occur.

When solving a system of linear equations $Gy = b$ with a general vector $b \in \mathbb{Z}^d$ we get analogously $\|y'\|_\infty \leq N_1 \cdots N_{d-1} \cdot \|b\|_2$ for the integer vector $y' = \det G \cdot y$.

6.3. Bounds for scalar products

The most important values beside the entries of the vectors and hyperplane linear forms are the scalar products between them, $\lambda(v) = \langle \lambda, v \rangle$. To bound these we use the *Cauchy-Schwartz inequality*

$$|\langle v, w \rangle| \leq \|v\|_2 \cdot \|w\|_2 .$$

The value $\lambda(x)$ with a hyperplane λ and a generator x appears in the Fourier-Motzkin elimination and also as the height of a generator in the partial triangulation criteria. The Cauchy-Schwartz inequality gives us $|\lambda(x)| \leq \|\lambda\|_2 \cdot \|x\|_2 \leq \sqrt{d} N_1 \cdots N_{d-1} \cdot N_1$. We can simplify and slightly sharpen it to

$$|\lambda(x)| \leq \sqrt{d} D.$$

This uses the fact that when $G\lambda = e_i$ we have $\lambda(x) = 0$ whenever x is one of the rows of G and not row i.

In the global reduction for candidates $v = \sum_{i=1}^{d} q_i x_{j_i}$ with $0 \leq q_i < 1$ the values $\lambda(v)$ are used to decide reducibility. Using the triangle inequality and previous results we get

$$|\lambda(v)| = \left|\sum_{i=1}^{d} q_i \lambda(x_{j_i})\right| \leq \sum_{i=1}^{d} q_i \left|\lambda(x_{j_i})\right| < \sum_{i=1}^{d} \left|\lambda(x_{j_i})\right| \leq d\sqrt{d} D.$$

Two further types of scalar products come up in the algorithms. Firstly those with a grading linear form deg. This grading can be user defined, or it is found by Normaliz such that all generators of the cone have the same degree, or it is a grading used in the global reduction. In any case it is a single linear form, and its norm can be explicitly computed. We can bound the resulting scalar products by

$$|\deg(v)| \le \|\deg\|_2 \cdot (N_1 + \cdots + N_d) \le \|\deg\|_2 \cdot dN_1.$$

So we have that $|\deg(v)|$ is also bounded by $d\sqrt{d}D$ as long $\|\deg\|_2 \le \sqrt{d}N_2 \cdots N_d$.

The second type is the evaluation of a hyperplane at the order vector O_C. In Normaliz the vector O_C is chosen as linear combination of the generators of the start simplicial cone with positive integer coefficients. If we choose the coefficients to be 1 it is also ensured that $|\lambda(O_C)| \le d\sqrt{d}D$. Depending on how large the difference between the allowed size and the bound is, we can also allow larger coefficients.

6.4. Bounds for intermediate values

From the last sections we know that the results of the algorithms and values that are used in comparisons can be bounded by $d\sqrt{d}D$. But so far we have not considered intermediate values, for example in the Gaussian reduction or in the Fourier-Motzkin elimination. And, in fact, the bounds do not apply to them.

Large intermediate values which cause an overflow do not necessarily imply a wrong result. In computation that only apply addition, subtraction, and multiplication, i.e. do not divide, it is possible to calculate modulo 2^{64} and the result will be correct (in $\mathbb{Z}$) as long its maximum norm is bounded by 2^{63}. But comparisons between two numbers, divisions, and divisibility tests rely on the correct value and not only on the congruence class.

For instance in the Fourier-Motzkin elimination, on computes

$$\mu' = \lambda_1(x)\lambda_2 - \lambda_2(x)\lambda_1$$

to find a new linear form $\mu = \mu' / \gcd(\mu')$. We can ensure that the coefficients of $\lambda_1, \lambda_2, x, \lambda_1(x), \lambda_2(x)$ and μ are bounded by $\sqrt{d}D$, but not $\lambda_1(x)\lambda_2$, $\lambda_2(x)\lambda_1$, and especially not μ'.

Example 6.3. We will show this effect on the cone C generated by $x_1 = (-1, a)$, $x_2 = (1, a)$ and $x_3 = (a, a-1)$ with a large integer a. Then we have $D < \sqrt{2}a^2$ and $d\sqrt{d}D < 4a^2$ and these are representable with 64 bits if we choose $a = 2^{30}$. Via inverting we get the hyperplanes $\lambda_1 = (a, 1)$ and $\lambda_2 = (-a, 1)$. In the first Fourier-Motzkin elimination step we compute

$$\begin{aligned}
\mu' &= \lambda_1(x_3)\lambda_2 - \lambda_2(x_3)\lambda_1 \\
&= (a^2 + a - 1)(-a, 1) - (-a^2 + a - 1)(a, 1) \\
&= (-a^3 - a^2 + a, a^2 + a - 1) + (a^3 - a^2 + a, a^2 - a + 1) \\
&= (-2a^2 + 2a, 2a^2) \ = \ 2a(-a + 1, a)
\end{aligned}$$

In this example the intermediate values would cause an overflow in 64 bit arithmetic, but the result would still be correct, because $\|\mu'\|_\infty = 2a^2 = 2^{61} < 2^{63}$.

The intermediate values and μ' are bounded by

$$\|\mu'\|_\infty \le |\lambda_1(x)| \cdot \|\lambda_2\|_\infty + |\lambda_2(x)| \cdot \|\lambda_1\|_\infty \le 2\sqrt{d}D \cdot N_1 \cdots N_{d-1}.$$

If this bound is exceeded, we have to apply some control whether the result is correct. More on this in Section 6.6.

When solving linear systems using Gaussian reduction the intermediate values are more complicated to bound. But if we restrict to certain row operations it is possible. Bareiss [**3**] showed that, when using only a special type of row operation, the coefficients in the intermediate systems are all minors of the input matrix and can therefore be bounded by the Hadamard bound D. Bareiss also gave more advanced algorithms for which this bound applies, for instance the division free two-step algorithm [**3**, (2.8)].

6.5. Bounds for the Hilbert series

In the Hilbert series computation the numerator is the critical component. The denominator is very well controllable as we saw in Chapter 4. The degree of the numerator in a simplicial cone generated by $x_1, \dots, x_d$ is bounded by

$$\deg(y + \varepsilon(y)) < \deg(x_1 + \cdots + x_d),$$

compare Theorem 4.3. A bound for this degree already has been discussed in Section 6.3.

The sum of the (nonnegative) coefficients in the numerator of the Hilbert series of a simplicial cone is the determinant of the generator matrix, hence it is bounded by D. Thereby the coefficients of the sum in one denominator class is bounded by the sum of the determinants. But exceeding the precision limit here would mean to enumerate and evaluate more points than the precision can store. At the moment this is practically not possible in reasonable time. Basing on an evaluation speed of 20 million points per second on one core (measured speed on a modern system) it would take 1000 such cores still 14 years to reach the limit of 64 bit integers.

When summing series with different denominators and calculating the (quasi)-polynomial the situation changes. Because of multiplications and other operations, the size of the coefficients can grow more quickly. Since those operations use only a very small part of the time, we always use the arbitrary precision type here.

6.6. Application of the bounds in Normaliz

In Normaliz we want to use 64 bit signed integers since they give the best performance on most modern computers. With this integer type we can represent all values that are bounded in absolute value by $B = 2^{63}$. In this chapter we have gathered the following bounds.

Theorem 6.4. *Let $C \subset \mathbb{R}^d$ be a cone generated by integer vectors $X = \{x_1, \dots, x_n\}$ and let the d largest 2-norms of the generators be $N_1, \dots, N_d$. With $D = N_1 \cdots N_d$ we have:*

(i) $|\det G| \le D$ *for every matrix G of d generators,*
(ii) $\|\lambda\|_\infty \le N_1 \cdots N_d \le D$ *for every support hyperplane H_λ of a cone generated by a subset of X,*
(iii) $|\lambda(x)| \le \sqrt{d}D$ *for every $x \in X$, and*
(iv) $\|v\|_2 < dN_1$, $|\lambda(v)| \le d\sqrt{d}D$ *for every candidate v for Hilbert basis, especially for Hilbert basis elements.*

If for a cone all bounds of Theorem 6.4 stay below the precision bound B, the results will not cause arithmetic overflow. Otherwise we have to check the results for correctness. A possible error check strategy is the following:

If $2\sqrt{d}D \cdot N_1 \cdots N_{d-1}$ (the bound for $\|\mu'\|_\infty$ in a Fourier-Motzkin step) is bigger than B, perform error checks in the Fourier-Motzkin elimination in the following way. First test $|\lambda_1(x)| < B_{sp} - |\lambda_2(x)|$ with $B_{sp} = B/(N_1 \cdots N_{d-1})$. If this is not the case there are two possibilities: Either do a second computation modulo p and compare the results. Or take the scalar products with the involved generators to verify the desired properties of the new linear form, i.e. $\mu(x_i) = 0$ for the generators in the intersection $H_{\lambda_1} \cap H_{\lambda_2}$ and $\mu(x) > 0$ for the new generator x.

If $d\sqrt{d}D > B$, we have to verify the values $\lambda(v)$ in global reduction. For every computed hyperplane λ we first check $\|\lambda\|_2 \le \sqrt{d}N_2 \cdots N_d$, also see Section 6.3. If this is not the case, we do a second computation of every $\lambda(v)$ modulo a large prime p and compare the results.

Is even $\sqrt{d}D > B$ we have to check every computation or directly use higher precision arithmetic.

For solving systems of linear equations one can use adequate algorithms as described in Section 6.4. In the case that an algorithm is used for which the intermediate values cannot be bounded easily or the existing bounds exceed B, but still the computation should be done with fixed precision, we can validate the solution in the following way. First we check if the solution matches the norm bound; this has to hold even if the intermediate values in the computation can grow larger. Then we make a test multiplication with the input matrix, also here a bound was given since it is a scalar product of bounded values. In this way we can verify the result without the need of higher precision arithmetic.

If the values get too big we use the arbitrary precision arithmetic of the GMP library. An alternative are modular algorithms, see e.g. [**41**]. Considering the timing results of Table 3.1 it might be faster to do computation modulo a few primes than with `mpz_class` directly. Here the bounds again play an important rule to see how many primes are necessary to ensure the correct result.

In conclusion the presented bounds give the possibility to ensure from the beginning that no overflow can occur for a wide range of examples. Even if it cannot be excluded completely, we have presented which computations are the critical ones, that have to be checked. On the other hand the bounds give indications for which examples it might be necessary to use arbitrary precision arithmetic.

Bibliography

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

[2] J. Abbott and A.M. Bigatti, *CoCoALib: a C++ library for doing Computations in Commutative Algebra*, Available at `http://cocoa.dima.unige.it/cocoalib`.

[3] E. Bareiss. *Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination.* Mathematics of computation **22 (102)** (1968), 565–578.

[4] W. Bruns and J. Gubeladze. *Polytopes, rings, and K-theory.* Springer Monographs in Mathematics (2009).

[5] W. Bruns and J. Herzog. *Cohen-Macaulay rings.* Cambridge University Press (1993/8).

[6] W. Bruns, R. Hemmecke, B. Ichim, M. Köppe, and C. Söger. *Challenging Computations of Hilbert Bases of Cones Associated with Algebraic Statistics.* Experimental Mathematics **20** (2011), 25–33.

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

[8] W. Bruns, B. Ichim and C. Söger. *The power of pyramid decomposition in normaliz.* Preprint, arXiv:1206.1916

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

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

[11] W. Bruns, B. Ichim and C. Söger. *The computation of generalized Ehrhart series in Normaliz.* Preprint, arXiv:1211.5178

[12] B. Burton *Enumerating fundamental normal surfaces: Algorithms, experiments and invariants.* ALENEX 2014: Proceedings of the Meeting on Algorithm Engineering & Experiments To appear, arXiv:1111.7055.

[13] B. Burton, R. Budney, W. Pettersson, et al. *Regina: Software for 3-manifold topology and normal surface theory.* Available at `http://regina.sourceforge.net/`.

[14] B. Burton, M. Ozlen *Computing the crosscap number of a knot using integer programming and normal surfaces.* ACM Transactions on Mathematical Software **39:1** (2012), 4:1-–4:18.

[15] CoCoATeam. *CoCoA: a system for doing Computations in Commutative Algebra.* Available at `http://cocoa.dima.unige.it`.

[16] M. Deza and M. Laurent. *Geometry of Cuts and Metrics.* Springer (1997).

[17] R. Diestel. *Graph Decompositions.* Oxford University Press (1990).

[18] R. Diestel. *Graph Theory*, Third Edition, Springer (2005).

[19] GCC team. *GCC – The GNU Compiler Collection* Available at `http://gcc.gnu.org`.

[20] M. Donten-Bury and M. Michalek *Phylogenetic invariants for group-based models* J. Alg. Stat. **3** (2012), 44–63

[21] H. Edelsbrunner. *Algorithms in combinatorial geometry.* Springer, (1987).

[22] E. Ehrhart. *Sur les systèmes d'inéquations diophantiennes linéaires.* J. Reine Angew. Math., **262/263**, (1973), 45–57.

[23] F.R. Giles and W.R. Pulleyblank. *Total dual integrality and integer polyhedra.* Linear Algebra Appl., **25**, (1979), 191–196.

[24] R. Hemmecke. *On the computation of Hilbert bases of cones.* In: Mathematical Software, ICMS 2002, A. M. Cohen, X.-S. Gao, N. Takayama (eds.), World Scientific (2002), 307–317.

[25] R. Hemmecke, J. Morton, A. Shiu, B. Sturmfels, and O. Wienand. *Three counterexamples on semigraphoids.* Comb. Probab. Comput. **17** (2008), 239–257.

[26] R. Kappl, M. Ratz, and C. Staudt. *The Hilbert basis method for D-flat directions and the superpotential.* Journal of High Energy Physics **10** (2010) 1–12.

[27] M. Köppe and S. Verdoolaege. *Computing parametric rational generating functions with a primal Barvinok algorithm.* Electr. J. Comb. **15** (2008), R16, 1–19.

[28] B. McKay. `nauty` *– A program for computing automorphism groups of graphs and digraphs.* Available at `http://cs.anu.edu.au/~bdm/nauty/`.

[29] H. Ohsugi. *Normality of cut polytopes of graphs is a minor closed property.* Discrete Math. **310** (2010), 1160–1166.

[30] 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.

[31] OpenMP Architecture Review Board. *OpenMP Application Program Interface* Available at `http://www.openmp.org`.

[32] L. Pottier. *The Euclidian Algorithm in Dimension n.* Research report, ISSAC 96, ACM Press (1996).

[33] A. Schürmann, *Exploiting polyhedral symmetries in social choice.* Social Choice and Welfare, **40** (2013), 1097–1110

[34] R. P. Stanley, *Linear Diophantine equations and local cohomology.* Invent. math. **68** (1982), 175–193.

[35] A. Ştefan. *The type of the base ring associated to a product of transversal polymatroids* Romanian Journal of Mathematics and Computer Science, **3** (2013), 205–220.

[36] M. Studený. *Probabilistic conditional independence structures.* Springer Series in Information Science and Statistics, Springer, London (2005).

[37] B. Sturmfels and S. Sullivant. *Toric geometry of cuts and splits.* Mich. Math. J. **57** (2008), 689–709.

[38] B. Sturmfels and V. Welker. *Commutative Algebra of Statistical Ranking.* J. Algebra **361** (2012), 264–286.

[39] S. Sullivant. *Compressed polytopes and statistical disclosure limitation.* Tohoku Math. J. **58** (2006), 433–445.

[40] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library.* Available at `http://gmplib.org/`.

[41] J. von zur Gathen, J. Gerhard. *Modern Computer Algebra.*, second ed., Cambridge University Press (2003).