Better triangulations in Normaliz 3.0

Christof Söger

Institute of Mathematics, University of Osnabrück

September, 2015

Normaliz

- Open source software (GPL)
- developed by W. Bruns, B. Ichim, T. Römer, R. Sieg, and C.S.
- written in C++ (using Boost and GMP/MPIR)
- runs under Linux, MacOs and MS Windows

Normaliz

- Open source software (GPL)
- developed by W. Bruns, B. Ichim, T. Römer, R. Sieg, and C.S.
- written in C++ (using Boost and GMP/MPIR)
- runs under Linux, MacOs and MS Windows
- parallelized with OpenMP
- C++ library libnormaliz

Normaliz

- Open source software (GPL)
- developed by W. Bruns, B. Ichim, T. Römer, R. Sieg, and C.S.
- written in C++ (using Boost and GMP/MPIR)
- runs under Linux, MacOs and MS Windows
- parallelized with OpenMP
- C++ library libnormaliz
- interfaces to most common algebra software systems
- file based interfaces for Singular, Macaulay 2 and Sage
- C++ level interfaces to CoCoA, polymake, Regina and GAP

Normaliz

- Open source software (GPL)
- developed by W. Bruns, B. Ichim, T. Römer, R. Sieg, and C.S.
- written in C++ (using Boost and GMP/MPIR)
- runs under Linux, MacOs and MS Windows
- parallelized with OpenMP
- C++ library libnormaliz
- interfaces to most common algebra software systems
- file based interfaces for Singular, Macaulay 2 and Sage
- C++ level interfaces to CoCoA, polymake, Regina and GAP

Applications in: commutative algebra, toric geometry, combinatorics, integer programming, invariant theory, elimination theory, mathematical logic, algebraic topology and theoretical physics.

The objectives of Normaliz

Normaliz computes

- in geometric terms: lattice points of polyhedra,
- in algebraic terms: solutions of linear diophantine systems.

The objectives of Normaliz

Normaliz computes

- in geometric terms: lattice points of polyhedra,
- in algebraic terms: solutions of linear diophantine systems.

The polyhedron and the lattice can be defined

- by generators: extreme rays of cones, vertices of polyhedra, generators of the lattice,
- by constraints: inequalities, equations, congruences.

The objectives of Normaliz

Normaliz computes

- in geometric terms: lattice points of polyhedra,
- in algebraic terms: solutions of linear diophantine systems.

The polyhedron and the lattice can be defined

- by generators: extreme rays of cones, vertices of polyhedra, generators of the lattice,
- by constraints: inequalities, equations, congruences.

The conversion between generators and constraints is an important part of Normaliz.

In this talk we restrict ourselves to the homogeneous case: the polyhedron is a cone, $0 \in$ lattice, constraints are homogeneous.

The objectives of Normaliz

Normaliz computes

- in geometric terms: lattice points of polyhedra,
- in algebraic terms: solutions of linear diophantine systems.

The polyhedron and the lattice can be defined

- by generators: extreme rays of cones, vertices of polyhedra, generators of the lattice,
- by constraints: inequalities, equations, congruences.

The conversion between generators and constraints is an important part of Normaliz.

In this talk we restrict ourselves to the homogeneous case: the polyhedron is a cone, $0 \in$ lattice, constraints are homogeneous.

Offspring NmzIntegrate computes weighted Ehrhart series and integrals of polynomials over rational polytopes.

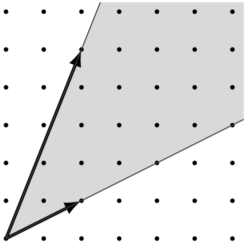
Rational cones

Definition

A lattice L is a subgroup of $\mathbb{Z}^d$. A (rational polyhedral) cone C is a subset

$$\begin{aligned} C &= \operatorname{cone}(x_1, \ldots, x_n) \\ &= \{a_1 x_1 + \cdots + a_n x_n : a_1, \ldots, a_n \in \mathbb{R}_+\} \end{aligned}$$

with a generating system $x_1, \ldots, x_n \in \mathbb{Z}^d$.



Rational cones

Definition

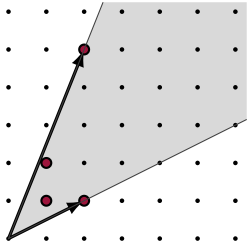
A lattice L is a subgroup of $\mathbb{Z}^d$. A (rational polyhedral) cone C is a subset

$$C = \mathrm{cone}(x_1, \dots, x_n) = \{a_1 x_1 + \dots + a_n x_n : a_1, \dots, a_n \in \mathbb{R}_+\}$$

with a generating system $x_1, \dots, x_n \in \mathbb{Z}^d$.

Theorem (Gordan's lemma)

Let $C \subset \mathbb{R}^d$ be the cone generated by $x_1, \dots, x_n \in \mathbb{Z}^d$. Then $C \cap L$ is an affine monoid M, i.e. a finitely generated submonoid of $\mathbb{Z}^d$.



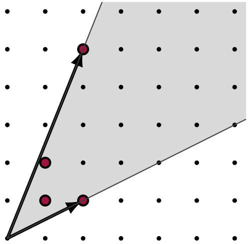
The tasks of Normaliz: Hilbert basis

From now on we assume that C is a pointed cone, i.e.

$$x, -x \in C \implies x = 0.$$

A lattice point $x \in M = C \cap L, x \neq 0$ is irreducible if

$$x = y + z \implies y = 0 \text{ or } z = 0.$$



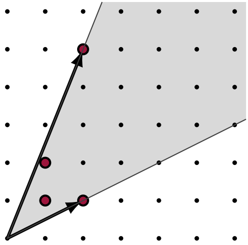
The tasks of Normaliz: Hilbert basis

From now on we assume that C is a pointed cone, i.e.

$$x, -x \in C \implies x = 0.$$

A lattice point $x \in M = C \cap L, x \neq 0$ is irreducible if

$$x = y + z \implies y = 0 \text{ or } z = 0.$$



Theorem

There are only finitely many irreducible elements in $C \cap L$ and they form the unique minimal system of generators, the Hilbert basis.

The tasks of Normaliz: Hilbert series

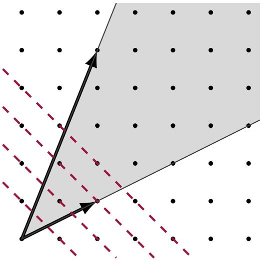
The second main task is to count the lattice points by degree.

The Hilbert (Ehrhart) function is given by

$$H(M, k) = \#\{x \in M : \deg x = k\}$$

and the Hilbert (Ehrhart) series is

$$H_M(t) = \sum_{k=0}^{\infty} H(M, k) t^k.$$



The tasks of Normaliz: Hilbert series

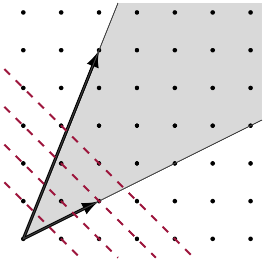
The second main task is to count the lattice points by degree.

The Hilbert (Ehrhart) function is given by

$$H(M,k) = \#\{x \in M : \deg x = k\}$$

and the Hilbert (Ehrhart) series is

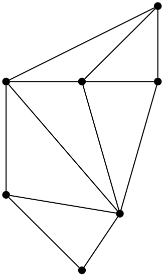
$$H_M(t) = \sum_{k=0}^{\infty} H(M,k)t^k.$$



Theorem (Hilbert-Serre, Ehrhart)

- *$H_M(t)$ is a rational function.*
- *$H(M,k)$ is a quasi-polynomial for $k \geq 0$.*

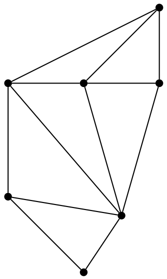
Normaliz Algorithm



In the Normaliz algorithm:

- Preparatory coordinate transformation, s.t. the cone is full dimensional and $L = \mathbb{Z}^d$.

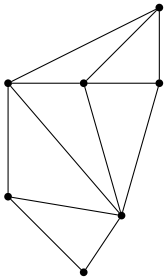
Normaliz Algorithm



In the Normaliz algorithm:

- Preparatory coordinate transformation, s.t. the cone is full dimensional and $L = \mathbb{Z}^d$.
- Compute a triangulation of the cone, that is a face-to-face decomposition into simplicial cones, interleaved with the computation of the support hyperplanes.

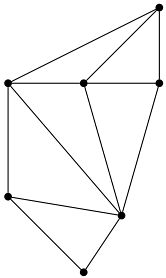
Normaliz Algorithm



In the Normaliz algorithm:

- Preparatory coordinate transformation, s.t. the cone is full dimensional and $L = \mathbb{Z}^d$.
- Compute a triangulation of the cone, that is a face-to-face decomposition into simplicial cones, interleaved with the computation of the support hyperplanes.
- Evaluate the simplicial cones in the triangulation independently from each other.

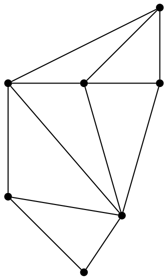
Normaliz Algorithm



In the Normaliz algorithm:

- Preparatory coordinate transformation, s.t. the cone is full dimensional and $L = \mathbb{Z}^d$.
- Compute a triangulation of the cone, that is a face-to-face decomposition into simplicial cones, interleaved with the computation of the support hyperplanes.
- Evaluate the simplicial cones in the triangulation independently from each other.
- Collect the data from the simplicial cones and process it globally.

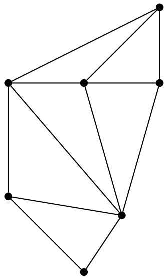
Normaliz Algorithm



In the Normaliz algorithm:

- Preparatory coordinate transformation, s.t. the cone is full dimensional and $L = \mathbb{Z}^d$.
- Compute a triangulation of the cone, that is a face-to-face decomposition into simplicial cones, interleaved with the computation of the support hyperplanes.
- Evaluate the simplicial cones in the triangulation independently from each other.
- Collect the data from the simplicial cones and process it globally.
- Inverse coordinate transformation.

Normaliz Algorithm



In the Normaliz algorithm:

- Preparatory coordinate transformation, s.t. the cone is full dimensional and $L = \mathbb{Z}^d$.
- Compute a triangulation of the cone, that is a face-to-face decomposition into simplicial cones, interleaved with the computation of the support hyperplanes.
- Evaluate the simplicial cones in the triangulation independently from each other.
- Collect the data from the simplicial cones and process it globally.
- Inverse coordinate transformation.

The two points in blue are the main steps that require the most time.

Recent developments

Recent developments, available in Normaliz 3.0:

- handling of inhomogeneous systems / polyhedra
- Improved input and output

Recent developments

Recent developments, available in Normaliz 3.0:

- handling of inhomogeneous systems / polyhedra
- Improved input and output
- stable integer arithmetic

Recent developments

Recent developments, available in Normaliz 3.0:

- handling of inhomogeneous systems / polyhedra
- Improved input and output
- stable integer arithmetic
- massive parallelization with Xeon Phi cards

Recent developments

Recent developments, available in Normaliz 3.0:

- handling of inhomogeneous systems / polyhedra
- Improved input and output
- stable integer arithmetic
- massive parallelization with Xeon Phi cards
- algorithmic improvements for the computation of the fixed lexicographic triangulation: pyramid decomposition and partial triangulations

Recent developments

Recent developments, available in Normaliz 3.0:

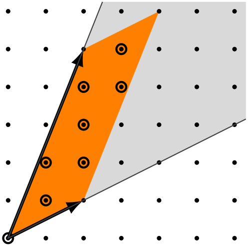
- handling of inhomogeneous systems / polyhedra
- Improved input and output
- stable integer arithmetic
- massive parallelization with Xeon Phi cards
- algorithmic improvements for the computation of the fixed lexicographic triangulation: pyramid decomposition and partial triangulations
- algorithms that allow us to find and use "better" triangulations

Simplicial cones

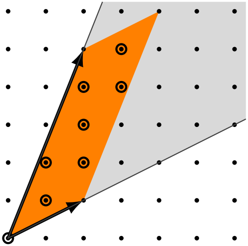
Let $x_1, \ldots, x_d$ be linearly independent and $S = \operatorname{cone}(x_1, \ldots, x_d)$. Then

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

together with $x_1, \ldots, x_d$ generate the monoid $S \cap \mathbb{Z}^d$.



Simplicial cones



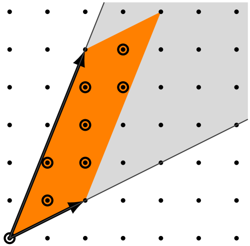
Let $x_1, \dots, x_d$ be linearly independent and $S = \operatorname{cone}(x_1, \dots, x_d)$. Then

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

together with $x_1, \dots, x_d$ generate the monoid $S \cap \mathbb{Z}^d$.

Every residue class in $\mathbb{Z}^d / U$, $U = \mathbb{Z}x_1 + \dots + \mathbb{Z}x_d$, has exactly one representative in E.

Simplicial cones



Let $x_1, \dots, x_d$ be linearly independent and $S = \mathrm{cone}(x_1, \dots, x_d)$. Then

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

together with $x_1, \dots, x_d$ generate the monoid $S \cap \mathbb{Z}^d$.

Every residue class in $\mathbb{Z}^d / U$, $U = \mathbb{Z}x_1 + \cdots + \mathbb{Z}x_d$, has exactly one representative in E.

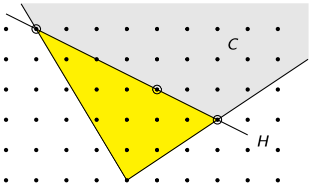
The elements in E are candidates for the Hilbert basis and their number is given by

$$|E| = \det(x_1, \dots, x_d).$$

Therefore the sum of the determinants of the simplices it is a critical size for the runtime of Normaliz.

Bottom decomposition

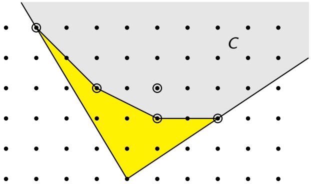
The determinant sum of the triangulation computed by Normaliz depends considerably on the order of the generators of the cone C. Unless they lie in a hyperplane H, then the determinant is exactly the normalized (lattice) volume of the polytope spanned by 0 and $C \cap H$.



This observation helps to find an optimal triangulation in the general case.

Bottom decomposition

We look at the bottom of the polyhedron generated by $x_1, \dots, x_n$ as vertices and C as recession cone, and take the volume underneath the bottom:



With the option `BottomDecomposition`, `-b`, Normaliz 3.0 computes a triangulation that respects the bottom facets. This gives the optimal determinant sum for the given generators.

While bottom decomposition is not used automatically for C, it is used for large simplicial cones in the triangulation if Normaliz can subdivide them.

Ordering of generators

The order of the vectors can play an enormous role.
Normaliz 3.0 orders the input vectors (after coordinate transformation) as follows:

1. If a triangulation is to be computed, first by degree (if present) or L_1-norm (otherwise).
2. Then lexicographically.

Ordering of generators

The order of the vectors can play an enormous role.
Normaliz 3.0 orders the input vectors (after coordinate transformation) as follows:

1. If a triangulation is to be computed, first by degree (if present) or L_1-norm (otherwise).
2. Then lexicographically.

The ordering by degree or L_1-norm reduces the size of the determinants of the simplicial cones. The lexicographic order is beneficial for the Fourier-Motzkin algorithm, at least for 0-1-input.

The user can block the ordering by setting `KeepOrder`, `-k`.

Ordering of generators

The order of the vectors can play an enormous role.
Normaliz 3.0 orders the input vectors (after coordinate transformation) as follows:

1. If a triangulation is to be computed, first by degree (if present) or L_1-norm (otherwise).
2. Then lexicographically.

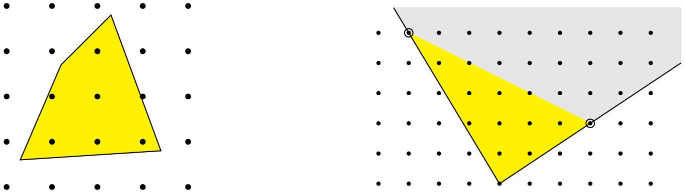
The ordering by degree or L_1-norm reduces the size of the determinants of the simplicial cones. The lexicographic order is beneficial for the Fourier-Motzkin algorithm, at least for 0-1-input.

The user can block the ordering by setting `KeepOrder`, `-k`.

Computation time reductions for the linear ordering polytope for $n = 6$: support hyperplanes: $35s \to 5s$, Hilbert basis: $72s \to 7s$.

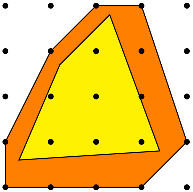
Approximation of rational polytopes

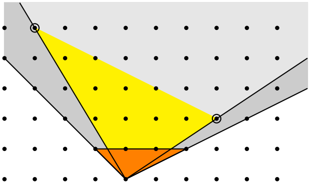
Often one wants to compute lattice points in rational polytopes. If the denominators of the vertices are large, a direct application of the Normaliz primal algorithm can easily fail because the determinants of the simplicial cones are enormous.



Approximation of rational polytopes

Often one wants to compute lattice points in rational polytopes. If the denominators of the vertices are large, a direct application of the Normaliz primal algorithm can easily fail because the determinants of the simplicial cones are enormous.

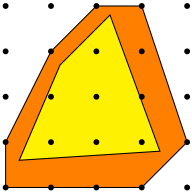


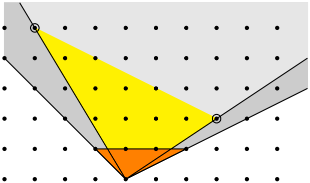


One way out: we approximate the rational polytope P by a larger integral polytope, compute its lattice points, and select those in in P. Often this has an overwhelming effect.

Approximation of rational polytopes

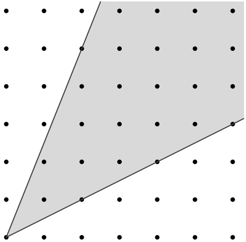
Often one wants to compute lattice points in rational polytopes. If the denominators of the vertices are large, a direct application of the Normaliz primal algorithm can easily fail because the determinants of the simplicial cones are enormous.





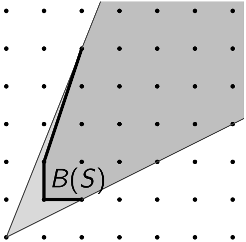
One way out: we approximate the rational polytope P by a larger integral polytope, compute its lattice points, and select those in in P. Often this has an overwhelming effect.
In order to use this method "globally" for P, one uses the option `Approximate`, `-r`. It is not used automatically since it could increase the geometric complexity in an unpredictable way.

Decompose simplicial cones



For a simplex with big volume, we decompose it into smaller simplices such that the sum of their volumes decreases remarkably.
For this purpose we compute points from the cone and use them for a new triangulation.

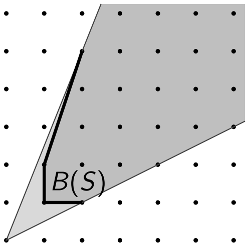
Decompose simplicial cones



For a simplex with big volume, we decompose it into smaller simplices such that the sum of their volumes decreases remarkably.
For this purpose we compute points from the cone and use them for a new triangulation.

Theoretically the best choice for these points are the vertices of the bottom $B(S)$ of the simplex which is defined as the union of the bounded faces of the polyhedron $\operatorname{conv}((S \cap \mathbb{Z}^d) \setminus \{0\})$.

Decompose simplicial cones

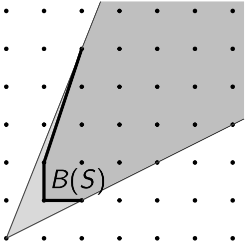


For a simplex with big volume, we decompose it into smaller simplices such that the sum of their volumes decreases remarkably.
For this purpose we compute points from the cone and use them for a new triangulation.

Theoretically the best choice for these points are the vertices of the bottom $B(S)$ of the simplex which is defined as the union of the bounded faces of the polyhedron $\operatorname{conv}((S \cap \mathbb{Z}^d) \setminus \{0\})$.

To determine some points from the bottom, we use:

Decompose simplicial cones



For a simplex with big volume, we decompose it into smaller simplices such that the sum of their volumes decreases remarkably.
For this purpose we compute points from the cone and use them for a new triangulation.

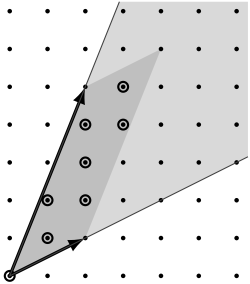
Theoretically the best choice for these points are the vertices of the bottom $B(S)$ of the simplex which is defined as the union of the bounded faces of the polyhedron $\operatorname{conv}((S \cap \mathbb{Z}^d) \setminus \{0\})$.

To determine some points from the bottom, we use:

- the approximation of the simplex, or
- integer programming methods.

Decomposition algorithm

$$S = \mathrm{cone}(x_1, \ldots, x_d)$$



UNIVERSITÄT OSNABRÜCK

Decomposition algorithm

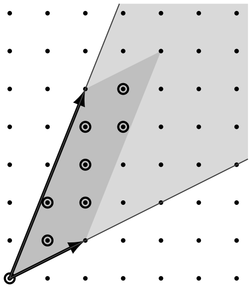
GOAL

compute a point x that minimizes the sum of determinants:

$$\sum_{i=1}^{d} \det(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_d) = N^T x,$$

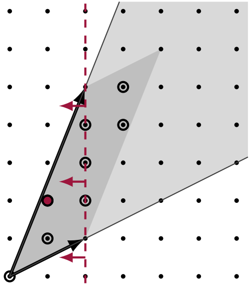
where N is a normal vector on the hyperplane spanned by $x_1, \ldots, x_d$.

$S = \mathrm{cone}(x_1, \ldots, x_d)$



Decomposition algorithm

$S = \mathrm{cone}(x_1, \ldots, x_d)$



GOAL

compute a point x that minimizes the sum of determinants:

$$\sum_{i=1}^{d} \det(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_d) = N^T x,$$

where N is a normal vector on the hyperplane spanned by $x_1, \ldots, x_d$.

SOLVE THE IP

$$\min\{N^T x : x \in S \cap \mathbb{Z}^d, x \neq 0, N^T x < N^T x_1\} \qquad (\star)$$

Decomposition algorithm

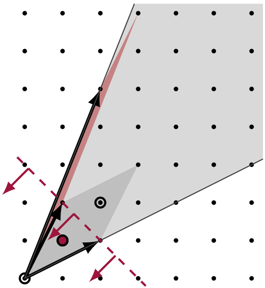
GOAL

compute a point x that minimizes the sum of determinants:

$$\sum_{i=1}^{d} \det(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_d) = N^T x,$$

where N is a normal vector on the hyperplane spanned by $x_1, \ldots, x_d$.

$S = \mathrm{cone}(x_1, \ldots, x_d)$



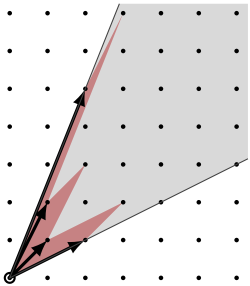
SOLVE THE IP

$$\min\{N^T x : x \in S \cap \mathbb{Z}^d, x \neq 0, N^T x < N^T x_1\} \qquad (\star)$$

If problem can be solved: form a stellar subdivision with the solution.

Decomposition algorithm

$S = \mathrm{cone}(x_1, \ldots, x_d)$



GOAL

compute a point x that minimizes the sum of determinants:

$$\sum_{i=1}^{d} \det(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_d) = N^T x,$$

where N is a normal vector on the hyperplane spanned by $x_1, \ldots, x_d$.

SOLVE THE IP

$$\min\{N^T x : x \in S \cap \mathbb{Z}^d, x \neq 0, N^T x < N^T x_1\} \qquad (\star)$$

If problem can be solved: form a stellar subdivision with the solution.

Decomposition algorithm

- use SCIP (3.2.0) via its C++ interface

Decomposition algorithm

- use SCIP (3.2.0) via its C++ interface
- decompose until determinant reaches 10^6

Decomposition algorithm

- use SCIP (3.2.0) via its C++ interface
- decompose until determinant reaches 10^6
- the collected subdivision points are then used to compute the bottom decomposition of the simplicial cone

Decomposition algorithm

- use SCIP (3.2.0) via its C++ interface
- decompose until determinant reaches 10^6
- the collected subdivision points are then used to compute the bottom decomposition of the simplicial cone
- parallelization with OpenMP

Decomposition algorithm

- use SCIP (3.2.0) via its C++ interface
- decompose until determinant reaches 10^6
- the collected subdivision points are then used to compute the bottom decomposition of the simplicial cone
- parallelization with OpenMP

Decomposition algorithm

- use SCIP (3.2.0) via its C++ interface
- decompose until determinant reaches 10^6
- the collected subdivision points are then used to compute the bottom decomposition of the simplicial cone
- parallelization with OpenMP

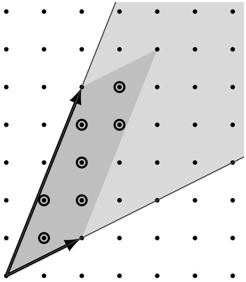
	hickerson-16	hickerson-18	knapsack_11_60
simplex volume	9.83×10^7	4.17×10^{14}	2.8×10^{14}
bottom volume	8.10×10^5	3.86×10^7	2.02×10^7
our volume	3.93×10^6	5.47×10^7	2.39×10^7
old runtime	2s	$>$12d	$>$8d
new runtime	0.5s	46s	5.1s

SUN xFire 4450, 4 Intel Xeon X7460, 20 threads

Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

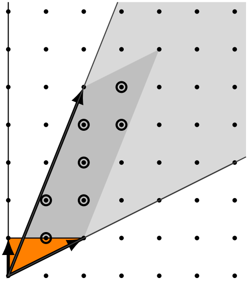
Approximate simplicial cone



Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

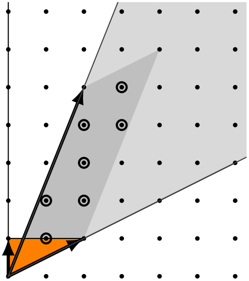
Approximate simplicial cone



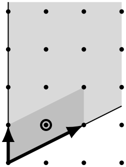
Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

Approximate simplicial cone



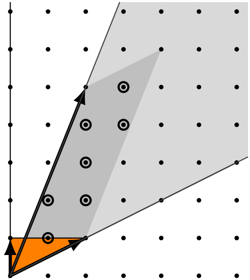
Compute the approximating cone



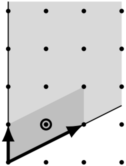
Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

Approximate simplicial cone



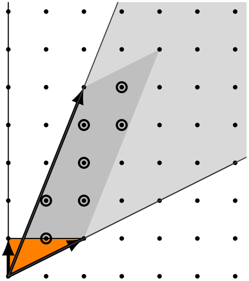
Compute the approximating cone



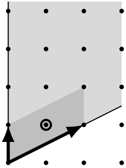
Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

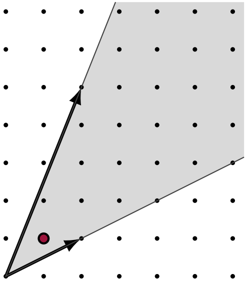
Approximate simplicial cone



Compute the approximating cone



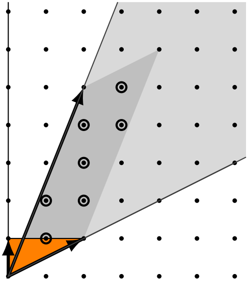
Subdivide simplicial cone



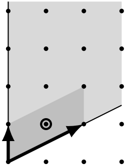
Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

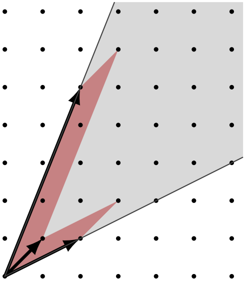
Approximate simplicial cone



Compute the approximating cone



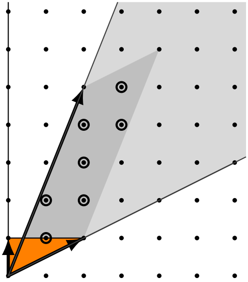
Subdivide simplicial cone



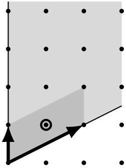
Decomposition via approximation

Instead of solving the IP, we can approximate the simplicial cone to find possible subdivision elements.

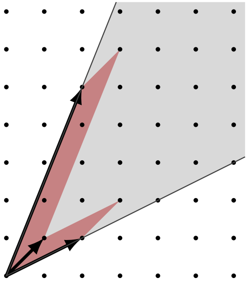
Approximate simplicial cone



Compute the approximating cone



Subdivide simplicial cone



Using SCIP and the approximation might be used in combination and the approximation might be redone in a higher level.

Decomposition via approximation

Note: After subdivision the decomposition of the cone may no longer be a triangulation in the strict sense, but a decomposition that we call a nested triangulation.