Computations of Integer Hulls of Polyhedra

Jüergen Gerhard[1], Marc Moreno Maza[2] and Linxiao Wang[2]

[1]Maplesoft, [2]University of Western Ontario

Overview

We present a new algorithm for computing the integer hull of a rational polyhedral set, together with its implementation in MAPLE, as the `PolyhedralSets:-IntegerHull` command, and in the C programming language in the `BPAS` library. Our experimental results show that our algorithm can deal with polyhedral sets with large number of integer points, which are out of reach for state-of-the-art software. More details can be found in our CASC2022 paper [1].

Main ideas

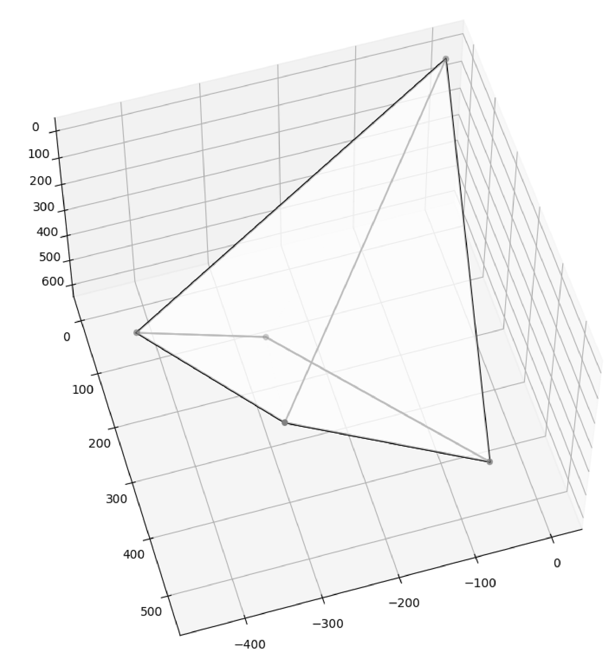
Let $P \subseteq \mathbb{Q}^d$ be a rational polyhedron that is, the solution set of a system of linear inequalities. In practice, P is given by its faces of dimension 0, called *vertices*, or its faces of dimension $d-1$, called *facets*. The *integer hull* P_I of P is the intersection of all polyhedra containing $P \cap \mathbb{Z}^d$. P_I is itself a rational polyhedron and Algorithm 1 computes its vertices. With the polyhedron on Figure (1a) as input, we illustrate the three main steps of our algorithm.

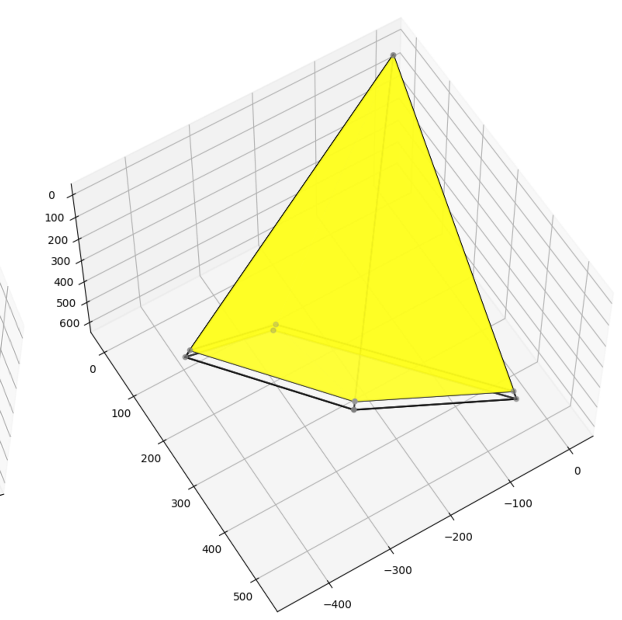
Normalization. By means of Hermite normal form, we construct a rational polyhedron $Q \subseteq \mathbb{Q}^d$ such that $Q_I = P_I$ and each supporting hyperplane of a facet has integer points, see Figure (1b).

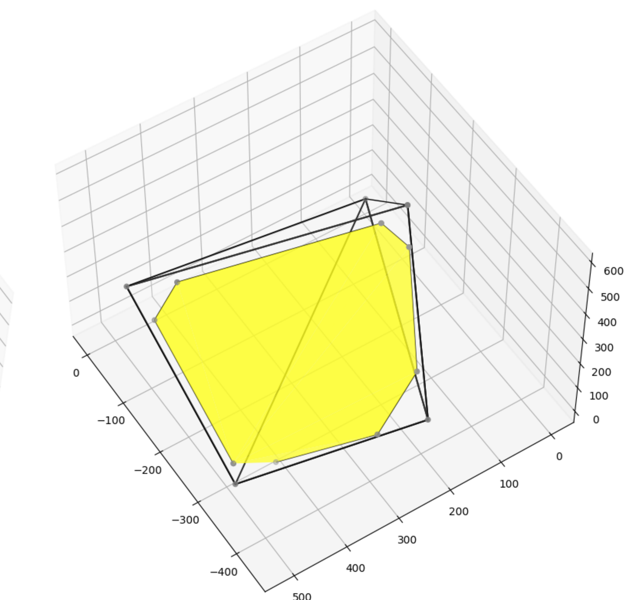
Partitioning. We search for integer points inside Q so as to partition Q into smaller polyhedral sets, the integer hulls of which can easily be computed. We observe that every vertex of Q which is an integer point is also a vertex of Q_I. Now, for every vertex v of Q which is not an integer point we look, on each facet F to which v belongs, for an integer point $C_{v,F}$ that is "close" to v (ideally as close as possible to v). This is achieved by a recursive call to our algorithm so as to compute the integer hull of F, see Figure (1c). All the points $C_{v,F}$ together with the vertices of Q are used to build that partition of Q, see Figures (1d), (1e), (1f), (1g).

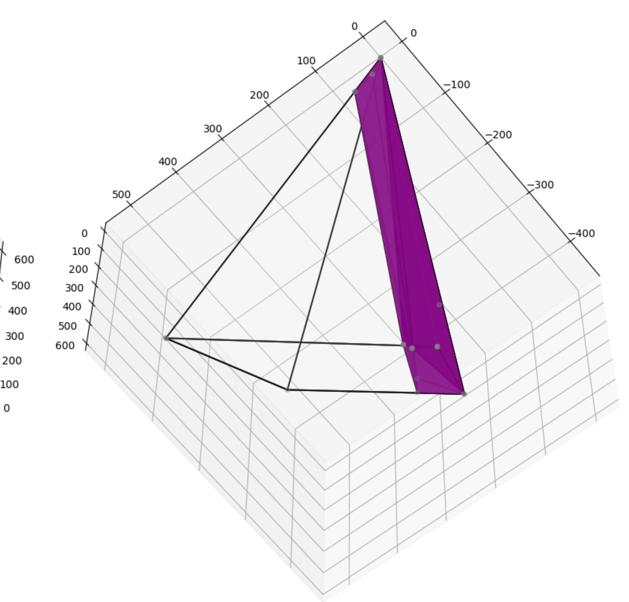
Merging. Once the integer hull of each part is there, a convex-hull procedure (`QuickHull`) yields P_I. The output polyhedron is on Figures (1h). Note that Q_I has often far more vertices than P.

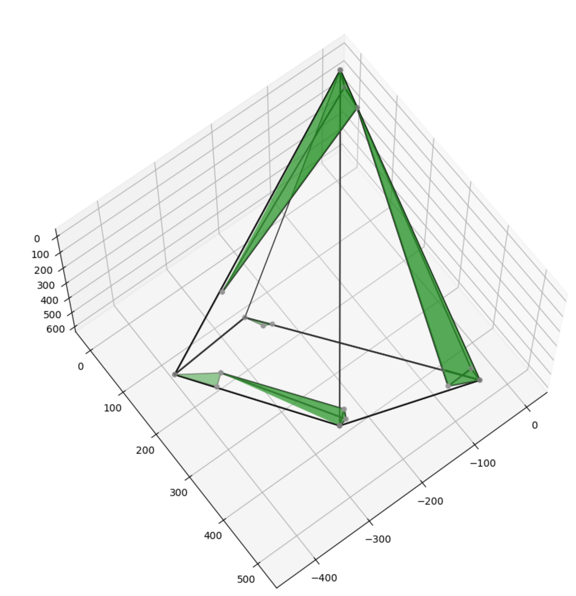
An example and benchmarks

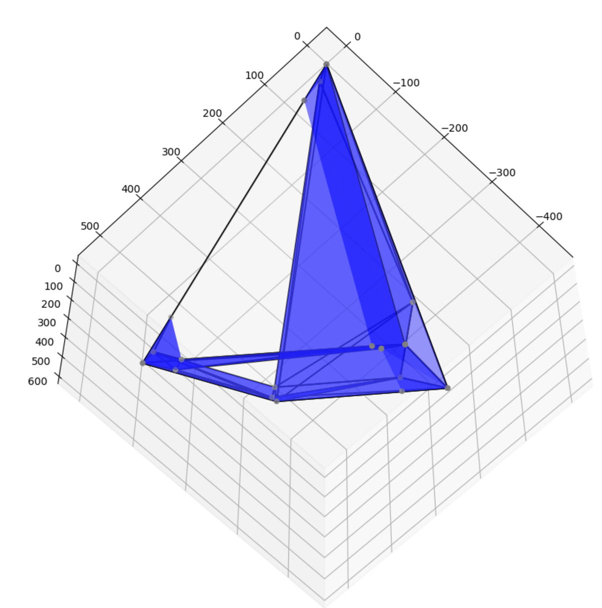

(a) The input polyhedral set

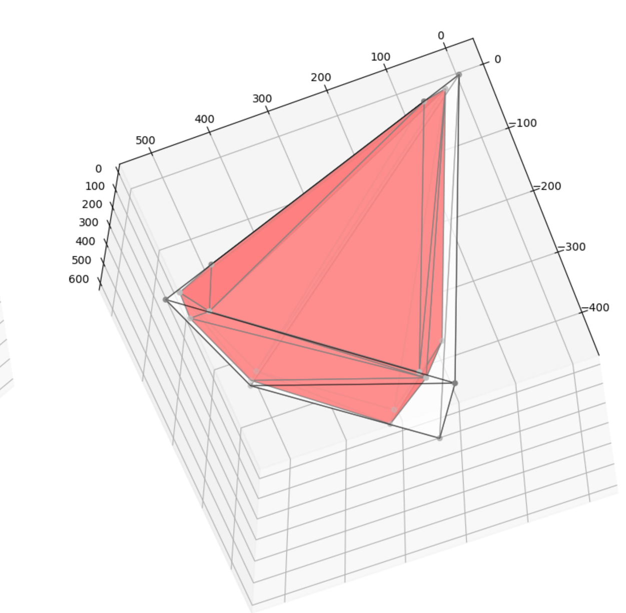

(b) Normalization


(c) Integer hull of a facet


(d) One vertex for which no adjacent facet has integer points


(e) Every non-integer vertex and its "closest" integer points form a *green* part of the partition


(f) We make one or two blue *blue* parts for each edge


(g) The parts for which all vertices are integer, thus for which the integer hull is obvious

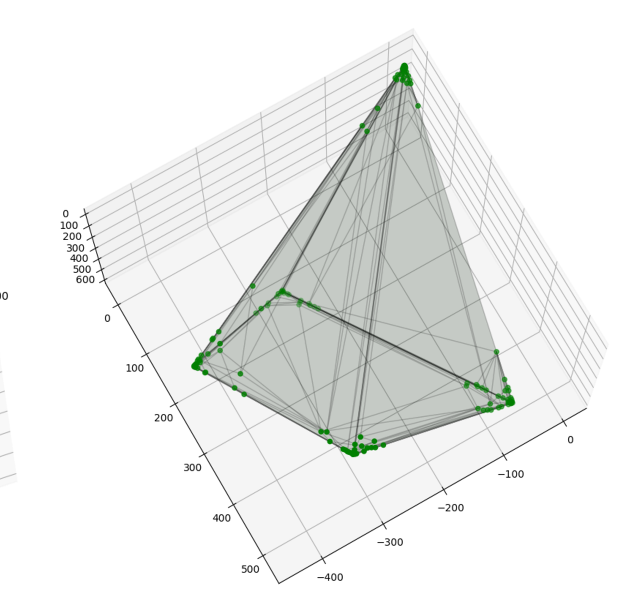

(h) Integer hull of the input

Figure: A 3D example: the input has 5 vertices, 8 edges and 5 facets; its integer hull has 139 vertices.

Volume	447.48		6991.89		55935.2	
Algorithm	IntegerHull	EIP+CH	IntegerHull	EIP+CH	IntegerHull	EIP+CH
Time(s)	1.202	6.892	1.498	67.814	1.517	453.577

Table: Integer hulls of tetrahedra (4 facets, 4 vertices and 6 edges)

Volume	412.58		7050.81		60417.63	
Algorithm	IntegerHull	EIP+CH	IntegerHull	EIP+CH	IntegerHull	EIP+CH
Time(s)	1.476	5.711	1.573	60.233	1.728	512.101

Table: Integer hulls of triangular bipyramids (6 facets, 5 vertices and 9 edges)

Tables 1 and 2 show the benchmarks of our MAPLE implementation. It is accessible in MAPLE2022 as the `PolyhedralSets:-IntgerHull` command. The cost for finding all the integer points is related to the volume of the input and we can see the trend in the "EIP+CH" columns. The complexity of our algorithm depends on the number of facets and the number of fractional vertices in the input.

example	IntegerHull	Naive	Normaliz
3d1_0	51.727	11.396	274.364
3d1_1	52.034	13.483	1018.449
3d1_2	60.821	21.106	2330.534
3d1_3	54.350	79.219	15346.996
3d2_0	4.488	0.826	851.495
3d2_1	4.615	0.923	956.666
3d2_2	4.624	1.527	793.192
3d2_3	5.522	4.394	1318.150
3d3_0	11.049	21.235	7862.109
3d3_1	16.001	145.068	N/A
3d3_2	23.822	2082.559	N/A
3d3_3	24.162	N/A	N/A

Table: Timing (ms) for computing integer hull of 3D examples.

Table 3 show the benchmarks of our C/C++ implementation for some 3D inputs. We compare our results with that of the Normaliz library.

General algorithm

Algorithm 1: Compute the integer hull of a polyhedralset

Function `IntegerHull`(P)
  Input: P, a PolyhedralSet
  Output: I, a list of the vertices of the integer hull of P
if *P is not fully dimensional* **then**
  $R_F, G \leftarrow$ `HNFProjection(P)`
  `/* make projection` G `of` P `to a dimension where` G `is full dimensional */`
  $V_G \leftarrow$ `IntegerHull(G)`
  $V_P \leftarrow \mathtt{R_F(V_G)}$
  return V_P
$P \leftarrow$ `Normalization`(P)
$D \leftarrow$ `Dimension`(P)
$L \leftarrow$ `FaceLattice`(P)
for *each f in L* **do**
  $V_f \leftarrow$ `IntegerHull(f)`
  $V \leftarrow$ `Vertices(f)`
  for *each v in V* **do**
    find the closest point to v in V_f
$V_{set} \leftarrow \{\}$
for *i from 0 to $D-2$* **do**
  $F \leftarrow$ `Faces(L, i)`
  for *each f in F* **do**
    $V \leftarrow$ `Vertices(f)`
    if *there are integer points on f* **then**
      for *each v in V* **do**
        $C \leftarrow$ `CornerPolySet(v)`
        $P_T \leftarrow$ `Enumeration(C)`
        $V_T \leftarrow \mathtt{ConvexHull(P_T)}$
        $V_{set} \leftarrow V_{set} \cup V_T$
    else
      $C \leftarrow$ `CornerPolySet(f)`
      $P_T \leftarrow$ `Enumeration(C)`
      $V_T \leftarrow \mathtt{ConvexHull(P_T)}$
      $V_{set} \leftarrow V_{set} \cup V_T$
return `ConvexHull`(V_{set})

[1] Marc Moreno Maza and Linxiao Wang. Computing the integer hull of convex polyhedral sets. In *Computer Algebra in Scientific Computing - 24th International Workshop, CASC 2022, Gebze, Turkey, August 22-26, 2022, Proceedings*, pages 246–267. Springer, 2022.