Computing the Integer Hull of Convex Polyhedral Sets

Marc Moreno Maza and Linxiao Wang(✉)

University of Western Ontario, London, ON, Canada
{mmorenom,lwang739}@uwo.ca

Abstract. In this paper, we discuss a new algorithm for computing the integer hull P_I of a rational polyhedral set P, together with its implementation in Maple and in the C programming language. Our implementation focuses on the two-dimensional and three-dimensional cases. We show that our algorithm computes the integer hull efficiently and can deal with polyhedral sets with large numbers of integer points.

Keywords: Polyhedral set · Integer hull · Parametric polyhedron

1 Introduction

The integer points of rational polyhedral sets are of great interest in various areas of scientific computing. Two such areas are *combinatorial optimization* (in particular integer linear programming) and *compiler optimization* (in particular, the analysis, transformation and scheduling of for-loop nests in computer programs), where a variety of algorithms solve questions related to the points with integer coordinates belonging to a given polyhedron. Another area is at the crossroads of computer algebra and polyhedral geometry, with topics like toric ideals and Hilbert bases, see for instance [24] by Thomas.

One can ask different questions about the integer points of a polyhedral set, ranging from "whether or not a given rational polyhedron has integer points" to "describing all such points". Answers to that latter question can take various forms, depending on the targeted application. For plotting purposes, one may want to enumerate all the integer points of a 2D or 3D polytope. Meanwhile, in the context of combinatorial optimization or compiler optimization, more concise descriptions are sufficient and more effective.

For a rational convex polyhedron $P \subseteq \mathbb{Q}^d$, defined either by the set of its facets or by that of its vertices, one such description is the *integer hull* P_I of P, that is, the convex hull of $P \cap \mathbb{Z}^d$. The set P_I is itself polyhedral and can be described either by its facets, or its vertices. One important family of algorithms for computing the vertex set of P_I relies on the so-called *cutting plane method*, originally introduced by Gomory in [10] to solve integer linear programs (ILP) and mixed-integer programming (MILP) problems. This method is based on finding a sequence of linear inequalities (cuts) to reduce the feasible region to the

F. Boulier et al. (Eds.): CASC 2022, LNCS 13366, pp. 246–267, 2022.
https://doi.org/10.1007/978-3-031-14788-3_14

original ILP problem. Chvátal [6] and Schrijver [22] gave a geometrical description of the cutting plane method and developed a procedure to compute P_I based on it. Schrijver gave a full proof and a complexity study of this method in [20]. Another approach for computing P_I uses the *branch and bound method*, introduced by Land and Doig in the early 1960s in [15]. This method recursively divides P into sub-polyhedra, then the vertices of the integer hull of each part of the partition are computed.

There are also authors studying the relations between the vertices of P_I and the vertices of P. The authors of [11] provided an algorithm for finding the vertices of a polytope associated to the Knapsack integer programming problem. This algorithm computes boxes covering the input polyhedron and such that each box contains at most one vertex of P_I. Following that same approach, the authors [4] could give an upper bound on the number of those boxes, as well as a running estimate for enumerating the integer vertices of a polytope.

Since an integer hull is the convex hull of all the integer points within a polyhedral set, a straightforward way of computing the integer hull is enumerating all its integer points, followed by a convex hull computation. There is a family of studies focusing on enumerating or counting the lattice points of a given polyhedral set. A well-known theory on that latter subject was proposed by Pick [18]. In particular, the celebrated Pick's theorem provides a formula for the area of a simple polygon P with integer vertex coordinates, in terms of the number of integer points within P and on its boundary. In the 1990s, Barvinok [3] created an algorithm for counting the integer points inside a polyhedron, which runs in polynomial time, for a fixed dimension of the ambient space. Later studies such as [27] gave a simpler approach for lattice point counting, which divides a polygon into *right-angle triangles* and calculates the number of lattice points within each such triangle.

Verdoolaege et al. present in [25] a novel method for lattice point counting, based on Barvinok's decomposition for counting the number of integer points in a non-parametric polytope. In [23], Seghir, Loechner and Meister deal with the more general problem of counting the number of images by an affine integer transformation of the lattice points contained in a parametric polytope. In 2004, the software package `LattE` presented in [16] for lattice point enumeration offers the first implementation of Barvinok's algorithm. Other algorithms, such as [12] by Jing and Moreno Maza, compute an irredundant representation of the integer points of P in terms of "simpler" polyhedral sets, each of them given by a triangular-by-block system of linear inequalities.

`Normaliz` [5] is a software library for the computation of Hilbert bases of rational cones and the normalizations of affine monoids. The Hilbert basis of a convex cone C is a minimal set of integer vectors such that every integer vector in C is a conical combination of the vectors in the Hilbert basis with integer coefficients. The computation of a Hilbert basis of a simplicial cone can be done by enumerating all lattice points of paralleltopes. From there, `Normaliz` provides a command for computing the integer hull of a given polyhedral set based on enumeration and convex hull computation.

`Polymake` [1] is a software system that includes several algorithms for convex hull computation and lattice points enumeration (including those of `LattE`

and `Normaliz`). `Polymake` uses these algorithms to compute the integer hulls of various kinds of input polyhedral sets.

Since the integer hull P_I of P is completely determined by its vertices, it is natural to ask for the number of vertices in an integer hull of a polyhedron. The earliest study by Cook, Hartmann, Kannan and McDiarmid, in [8], shows that the number of vertices of P_I is related to the *size* (as defined in [20]) of the coefficients of the inequalities that describe P. Let $x = p/q$ be a rational number, where p and q are coprime integers, the size of x is defined as

$$size(x) = 1 + \lceil(\log(|p| + 1))\rceil + \lceil(\log(|q| + 1))\rceil.$$

For a linear inequality $a_n x_n + \cdots + a_1 x_1 + a_0 \leq 0$, its size is $\sum size(a_i)$. For a polyhedron $P = \{\mathbf{x} \mid A\mathbf{x} \leq \vec{b}\}$ where $A \in \mathbb{Q}^{m \times n}$ and $\vec{b} \in \mathbb{Q}^m$. Cook, Hartmann, Kannan and McDiarmid showed that the number of vertices of the integer hull of P is bounded over by $2m^n(6n^2\varphi)^{n-1}$ where φ is the maximum size of any of the m inequalities. More recent studies such as [26] and [4] use different approaches to reach similar or slightly improved estimates. We also discussed this question in our CASC 2021 paper [17].

In this paper, we present our algorithm for computing P_I and we report on the performance of its implementation as a new command of Maple's library `PolyhedralSets` [19] as well as in the C programming language. We present benchmarks for both implementations in Sect. 6. Our results show that our algorithm is very efficient comparing to the well known library Normaliz [5] especially when the input polyhedral set is large in volume.

Our algorithm has three main steps:

Normalization: during this step, we construct a new polyhedral set Q from P as follows. Consider in turn each facet F of P:

1. if the hyperplane H supporting F contains an integer point, then H is a hyperplane supporting a facet of Q,
2. otherwise one slides H towards the center of P along the normal vector of F, stopping as soon as one hits a hyperplane H' containing an integer point, then making H' a hyperplane supporting a facet of Q.

The resulting polyhedral set Q clearly has the same integer hull as P; computing Q is a preparation phase for the following step.

Partitioning: during this step, we search for integer points inside Q so as to partition P 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). All the points $C_{V,F}$ together with the vertices of Q are used to build that partition of Q. Each part of the partition is a polyhedron R which:

1. either has integer points as vertices (making the computation of the integer hull R_I trivial),
2. or has a small volume so that any algorithm (including exhaustive search) can be applied to compute R_I.

Merging: Once the integer hull of each part of the partition is computed and given by the list of its vertices, an algorithm for computing the convex hull of a set points, such as `QuickHull` [2], can be applied to deduce P_I.

The paper is organized as follows. Section 2 is a brief review of polyhedral geometry. Sections 4 and 5 present our algorithms in the 2D and 3D cases, respectively. Section 3.2 gathers key arguments supporting our algorithm, essentially based on the concept of the Hermite Normal Form of a matrix. Section 6 reports on our experimentation with the proposed algorithms.

2 Preliminaries

In this review of polyhedral geometry, we follow the concepts and notations of Schrijver's book [20], As usual, we denote by $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ the ring of integers, the field of rational numbers and the field of real numbers. Unless specified otherwise, all matrices and vectors have their coefficients in $\mathbb{Z}$. A subset $P \subseteq \mathbb{Q}^d$ is called a *convex polyhedron* (or simply a *polyhedron*) if $P = \{\mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \leq \vec{b}\}$ holds, for a matrix $A \in \mathbb{Q}^{m \times d}$ and a vector $\vec{b} \in \mathbb{Q}^m$, where m and d are positive integers; we call the linear system $\{A\mathbf{x} \leq \vec{b}\}$ an *H-representation* of P. Hence, a polyhedron is the intersection of finitely many affine half-spaces. Here an affine half-space is a set of the form $\{\mathbf{x} \in \mathbb{Q}^d \mid \vec{w}^t\mathbf{x} \leq \delta\}$ for some nonzero vector $\vec{w} \in \mathbb{Z}^d$ and an integer number δ.

A non-empty subset $F \subseteq P$ is a *face* of P if $F = \{\mathbf{x} \in P | A'\mathbf{x} = \vec{b'}\}$ for some subsystem $A'\mathbf{x} \leq \vec{b'}$ of $A\mathbf{x} \leq \vec{b}$. A face of P, distinct from P, and with maximum dimension is a *facet* of P. The *lineality space* of P is $\{\mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} = \vec{0}\}$ and P is said *pointed* if its lineality space has dimension zero. Note that, in this paper, we only consider pointed polyhedra. For a pointed polyhedron P, the inclusion-minimal faces are the *vertices* of P.

We are interested in computing P_I the *integer hull* of P, that is, the smallest convex polyhedron containing the integer points of P. In other words, P_I is the intersection of all convex polyhedra containing $P \cap \mathbb{Z}^d$. Assume that P is pointed. Then, $P = P_I$ if and only if every vertex of P is integral, see [21]. Thus, the convex hull of all the vertices of P_I is P_I itself.

3 Two Core Constructions of our Algorithm

In this section, we emphasize two constructions supporting respectively the *normalization* and *partitioning* steps of our algorithm. Both constructions deal with "algebraic aspects", that is, with the fact that we are solving for the integer solutions of a system of linear inequalities. These two constructions are inspired respectively by [17] and [12].

3.1 Normalization

Considering the rational polyhedron $P = \{\mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \leq \vec{b}\}$, with the notations of Sect. 2, we observe that one can compute a vector $\vec{e} \in \mathbb{Z}^m$ so that the rational polyhedron $Q = \{\mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} \leq \vec{e}\}$ satisfies:

1. $P_I = Q_I$, and
2. the supporting hyperplane of every facet of Q has at least one integer point. Notice that this does not necessarily means that the new facet has an integer point.

In the introduction, the construction of Q is referred as the *normalization* step. We construct Q from P as follows:

1. consider each facet F of P in turn; if the hyperplane H supporting F does not contain an integer point, then one "slides" H towards the center of P along the normal vector of F, stopping as soon as a hyperplane H' containing an integer point is reached, otherwise keep H unchanged;
2. the resulting polyhedron is Q, for which rational consistency must be checked, which can be done efficiently using a method based on linear programming.

The "sliding process" described above informally is performed as follows. Let the equation below define the hyperplane H supporting F:

$$a_1 x_1 + \cdots + a_d x_d = b, \tag{1}$$

where $a_1, \ldots, a_d, b$ can be assumed to be integers. The fact that $\mathbb{Z}$ is an Euclidean domain (and thus a principal ideal domain) implies that H has integer points if and only if we have:

$$\gcd(a_1, \ldots, a_d) \mid b. \tag{2}$$

If the hyperplane H supporting F does not have integer points and P is included in the half-space $a_1 x_1 + \cdots + a_d x_d \leq b$, then H' is given by:

$$a_1 x_1 + \cdots + a_d x_d \leq g \lfloor \frac{b}{g} \rfloor, \tag{3}$$

with $g := \gcd(a_1, \ldots, a_d)$.

Summing things up, we denote by `Normalization`(P) a function call returning the polyhedron Q.

3.2 Partitioning

The other algebraic construction in our algorithm supports the *partition* step briefly explained in the introduction. The underlying question is the following: given a vertex V of P which is not an integer point and given a facet F of P to which V belongs, find on F an integer point $C_{V,F}$, if any, which is "close" to V (ideally as close to V as possible).

If P is two-dimensional, thus, if F is one-dimensional, then the question is easily answered by elementary arguments, see our previous paper [17]. If P has

dimension $d \geq 3$, thus, if F has dimension $d-1$, then we take advantage of the *Hermite normal form* of a matrix. In the sequel of this section, we review this concept. and use it to compute the integer hull of a facet of a polyhedron. Finally, we solve the question of finding an integer point $C_{V,F}$ on F (if any) as close as possible to V.

Hermite Normal Form. Consider a positive integer $p \leq d$ and a linear system $C\mathbf{x} = \mathbf{s}$ where $C \in \mathbb{Z}^{p \times d}$ is a full row-rank matrix and $\mathbf{s} \in \mathbb{Z}^p$ is a vector. There exists a uni-modular matrix $U \in \mathbb{Z}^{d \times d}$ so that $CU = [\mathbf{0}H]$ where $\mathbf{0} \in \mathbb{Z}^{p \times (d-p)}$ is the null matrix and H is the column-style Hermite normal form of C. We write $U = [U_L U_R]$ where $U_L \in \mathbb{Z}^{d \times (d-p)}$ and $U_R \in \mathbb{Z}^{d \times p}$. Therefore, the matrix $H \in \mathbb{Z}^{p \times p}$ is non-singular and the following properties hold:

1. $C\mathbf{x} = \mathbf{s}$ has integer solutions if and only if $H^{-1}\mathbf{s}$ is an integer vector,
2. every integer solution of $C\mathbf{x} = \mathbf{s}$ has the form $U_R H^{-1}\mathbf{s} + U_L\mathbf{z}$, where $\mathbf{z} \in \mathbb{Z}^{d-p}$ is arbitrary.

Determining the Integer Hull of a Facet. Let $\vec{c}^t\mathbf{x} = s$ be the equation of the hyper-plane supporting F, thus with $\vec{c}^t \in \mathbb{Z}^d$ and $s \in \mathbb{Z}$. Let $U \in \mathbb{Z}^{d \times d}$ be a uni-modular matrix so that $\vec{c}^t U = [\mathbf{0}H]$ where $\mathbf{0} \in \mathbb{Z}^{1 \times (d-1)}$ is the null matrix and H is the column-style Hermite normal form of $\vec{c}^t$ regarded as a matrix of $\mathbb{Z}^{1 \times d}$. We write $U = [U_L U_R]$ where $U_L \in \mathbb{Z}^{d \times (d-1)}$ and $U_R \in \mathbb{Z}^{d \times 1}$. Let $\mathbf{v} := U_R H^{-1} s$. Then, from the above paragraph on Hermite Normal Form, we know that the integer points of the hyper-plane supporting F are of the form $\mathbf{x} = \mathbf{v} + U_L\mathbf{z}$ where $\mathbf{z} \in \mathbb{Z}^{d-1}$ is arbitrary. The facet F is described by a system of linear inequalities in $\mathbb{Q}^d$ with $\mathbf{x}$ as unknown vector. Substituting $\mathbf{v} + U_L\mathbf{z}$ for $\mathbf{x}$ yields a system of linear inequalities in $\mathbb{Q}^{d-1}$ (with $\mathbf{z}$ as unknown vector) representing a rational polyhedron $G \subseteq \mathbb{Q}^{d-1}$. With these notations and hypotheses, we have the following.

Theorem 1. *The vertices of the integer hull G_I of G are in one-to-one correspondence with the vertices of the integer hull F_I of F via the map*

$$R_F : \begin{cases} \mathbb{Q}^{d-1} \rightarrow \mathbb{Q}^d \\ \mathbf{z} \longmapsto \mathbf{x} = \mathbf{v} + U_L\mathbf{z}. \end{cases} \tag{4}$$

In particular, we have $R_F(G_I) = F_I$.

Proof ▷ The proof follows from seven claims.

Claim 1. R_F is injective. Indeed, the matrix U is uni-modular, thus the columns of U are linearly independent, and the map $\mathbf{z} \longmapsto U_L\mathbf{z}$ is injective.

Claim 2. The image of R_F is F. Since R_F is an injective affine map from $\mathbb{Q}^{d-1}$ to $\mathbb{Q}^d$, it follows that the image of R_F is an affine space of dimension $d-1$. Therefore, in order to prove the claim, it suffices to prove that for every $\mathbf{z} \in \mathbb{Z}^{d-1}$ we have $R_F(\mathbf{z}) \in F$. Since $F \cap \mathbb{Z}^d \neq \emptyset$ (as a consequence of the normalization

step of our algorithm) there exists $\mathbf{z}_0 \in \mathbb{Z}^{d-1}$ so that $\mathbf{x}_0 := \mathbf{v} + U_L\mathbf{z}_0 \in F \cap \mathbb{Z}^d$ holds. Let $\mathbf{z} \in \mathbb{Z}^{d-1}$. Define $\mathbf{x} := R_F(\mathbf{z})$. We have:

$$\mathbf{x} = \mathbf{v} + U_L\mathbf{z}_0 + U_L(\mathbf{z} - \mathbf{z}_0) = \mathbf{x}_0 + U_L(\mathbf{z} - \mathbf{z}_0).$$

We deduce:

$$\vec{c}^t\mathbf{x} = \vec{c}^t\mathbf{x}_0 + \vec{c}^tU_L(\mathbf{z} - \mathbf{z}_0) = s + 0 = s,$$

which proves that $R_F(\mathbf{z}) \in F$ holds.

Claim 3. $R_F^{-1}(H)$ is a half-space of $\mathbb{Q}^{d-1}$ for any half-space H of $\mathbb{Q}^d$. Indeed, for any $\mathbf{x} \in \mathbb{Q}^d$ of the form $\mathbf{v} + U_L\mathbf{z}$, with $\mathbf{z} \in \mathbb{Q}^{d-1}$, we have

$$\vec{a}^t\mathbf{x} \geq b \quad \Longleftrightarrow \quad \vec{a}^tU_L\mathbf{z} \geq b - \vec{a}^t\mathbf{v},$$

where $H : \vec{a}^t\mathbf{x} \geq b$ is an arbitrary half-space of $\mathbb{Q}^d$.

Claim 4. The integer points of F are in one-to-one correspondence with the integer points of $\mathbb{Z}^{d-1}$. This claim follows directly from the properties of the Hermite Normal Form.

Claim 5. $R_F^{-1}(S)$ is a polyhedron of $\mathbb{Q}^{d-1}$ for any polyhedron S of $\mathbb{Q}^d$. Indeed, let $S := \cap_i H_i$ be a polyhedron of $\mathbb{Q}^{d-1}$ given as the intersection of finitely many half-spaces of $\mathbb{Q}^{d-1}$. We have

$$R_F^{-1}(S) = R_F^{-1}(\cap_i H_i) = \cap_i R_F^{-1}(H_i).$$

The conclusion follows with Claim 3.

Claim 6. $R_F(T)$ is a polyhedron of $\mathbb{Q}^d$ for any polyhedron T of $\mathbb{Q}^{d-1}$. The proof is similar to that of Claim 5.

Claim 7. We have: $R_F(G_I) = F_I$. Let $\mathcal{S}$ be the set of all polyhedra of $\mathbb{Q}^d$ containing $F \cap \mathbb{Z}^d$. Let $\mathcal{T}$ be the set of all polyhedra of $\mathbb{Q}^{d-1}$ containing $G \cap \mathbb{Z}^d$, where $G = R_F^{-1}(F)$. Then, by definition of F_I and G_I, we have:

$$F_I = \bigcap_{S \in \mathcal{S}} S \quad \text{and} \quad G_I = \bigcap_{T \in \mathcal{T}} T.$$

From Claim 5, we have:

$$R_F^{-1}(F_I) = \bigcap_{S \in \mathcal{S}} R_F^{-1}(S) \supseteq \bigcap_{T \in \mathcal{T}} T = G_I.$$

From Claim 6, and since R_F is injective,we have:

$$R_F(G_I) = \bigcap_{T \in \mathcal{T}} R_F(T) \supseteq \bigcap_{S \in \mathcal{S}} S = F_I.$$

Therefore, we have $R_F(G_I) = F_I$. Now we can prove the theorem. Since R_F is a bijective affine map from $\mathbb{Q}^{d-1}$ to F, it maps affine subspaces of dimension $0 \leq d' < d$ of $\mathbb{Q}^{d-1}$ to affine subspaces of dimension d' of F. Combined with Claims 5 and 6, this latter observation implies that faces of dimension $0 \leq d' < d$ of G_I are mapped to faces of dimension d' of F_I. Therefore, the vertices of G_I are in one-to-one correspondence with the vertices of F_I.

Theorem 1 shows that one can reduce the computation of the vertices of F_I to computing the vertices of G_I.

Based on that observation, we denote by `HNFProjection`(F, d) a function call returning the ordered pair (G, R_F).

Finding an Integer Point $C_{V,F}$ on F (If Any) Close to V. Let us return now to the question of finding an integer point $C_{V,F}$ on F (if any) as close as possible to V. A second consequence of Theorem 1 is that we can compute an integer point $C_{V,F}$ simply by choosing a point $R_F(W)$ at minimum Euclidean distance to V, where W ranges in the set of the vertices of G_I. As mentioned, such a point may not be an integer point of F at minimum Euclidean distance to V, but if F is large enough (that is, if its area is large enough) then $C_{V,F}$ is a good approximate solution to this optimization problem.

4 Integer Hull of a 2D Polyhedral Set

In this section, we present our algorithm for computing the integer hull of a 2D polyhedral set. We first give a high-level introduction of the algorithm, then we present its sub-routines, a more precise presentation of the general algorithm together with the implementation details.

As introduced in Sect. 1, our main idea is to partition the input 2D-polyhedral set into several smaller areas, compute the integer hulls of each area and find a convex hull of all these integer hulls.

In Sect. 2, we explained that an integer hull is a convex polyhedral set whose vertices are all integer points. Therefore, given a polyhedral set that is not an integer hull, if we can replace each fractional vertex with some integer ones, we will obtain the integer hull of the input polyhedral set. Of course, during this replacement process, we should not exclude any integer points, otherwise the result would not be valid.

To replace the fractional vertices, we need to look at the areas around those vertices that are the corners of the input polyhedral set. We do that by partitioning the input such that each fractional vertex is included in a "small" triangle, for which the integer hull is computed by a straightforward method.

Other than these corners, there is the central part of the input, ideally this should be the part that covers most of the area of the input. To make the computation of the central part easier, we construct the partition by ensuring the central area is already an integer hull. In the final step, we combine the corner parts and the central part using a convex hull algorithm to compute the final output.

To meet all the requirements above, we propose the following method to partition the input. First, we normalize the input using procedure `Normalization(P)`. For each fractional vertex, we find the closest integer point to it on each of its adjacent facets. For a 2D polyhedral set, each vertex has exactly two adjacent facets, therefore, two "closest integer points". We partition the input by connecting each of these closest integer point pairs. Thus, in most cases a corner

part would be a triangle with vertices of a fractional vertex and its two closest integer points. In some special cases when some facets contain no integer point, we combine the adjacent vertices and their closest integer points to form a polyhedral set has two and only two integer vertices. The central part is an integer hull with vertices of all these closest integer points and all the integer vertices of the input.

The details of the sub-routines as well as the general algorithm are given in the following sections.

4.1 Algorithm

In this section, we consider an input polyhedral set P defined by a system of linear inequalities

$$\begin{cases} a_{11}x_1 + a_{21}x_2 \leq b_1, \\ a_{12}x_1 + a_{22}x_2 \leq b_2, \\ \quad \cdots \\ a_{1n}x_1 + a_{2n}x_2 \leq b_n, \end{cases}$$

where $\gcd(a_{1i}, a_{2i}, b_i) = 1$ for $i \in \{1, \ldots, n\}$. We assume that this representation of P is irredudant, that is, the defining linear inequalities of P are in one-to-one correspondence with the facets of P. In this paper, we follow the convention of MAPLE's `PolyhedralSets` library and refer to these inequalities as the `relations` of P.

Following the informal description of the algorithm above, for each fractional vertex, we need to find the closest integer points on the facets adjacent to this vertex. But we first notice that it is possible that the supporting hyperplane of a facet, and therefore the facet itself, do not have any integer points. Therefore, the first step of our algorithm is to normalize the `relations` of the input using the "sliding process" described in Sect. 3.1.

In the next step, `closestIntegerPoints` (Algorithm 1), we find the closest integer point to each fractional vertex on its adjacent facets. From the proof of Lemma 1 in [17] we know that, on a line $a_1x + a_2y = b$, a point is an integer point if and only if it has x value of $x \equiv \frac{b}{a_1} \mod a_2$. We can use this observation to find the closest integer point on a line to a given point. We also deal with the case where a facet does not contain any integer point.

Next, we need to construct the corner polyhedral sets and compute their integer hulls. Then, we find the convex hull of all these integer hulls (see Algorithm 2). Lemma 4 in [17] shows that the vertices of this final convex hull are the vertices of P_I.

For a fractional vertex $V[i]$, if neither $V_C[i][1]$ nor $V_C[i][2]$ is NULL, then the corner is a triangle with vertices $[V[i], V_C[i][1], V_C[i][2]]$. If one or both of $V_C[i][1]$ and $V_C[i][2]$ are NULL, which means there is no integer point on one or both adjacent facets of $V[i]$, we construct the corner as follow.

Algorithm 1: Compute the closest integer points to each fractional vertex on its adjacent facets

```
1 Function closestIntegerPoints(V)
      Input: V, a list of the vertices of P
      Output: V_C, a list of pairs where V_C[i][1] and V_C[i][2] store the closest
              integer points of vertex V[i] on its two adjacent facets.
2     for i = 1, ..., n do
3         Let V[i_1] and V[i_2] be the vertices adjacent to V[i]
4         for j = 1, 2 do
5             if there are integer points between [V[i], V[i_j]] then
6                 V_C[i][j] ← closest integer point to V[i] on [V[i], V[i_j]]
7             else
8                 V_C[i][j] ← NULL
9     return V_C
```

1. Let V_P be an empty set.
2. Let's say facet f is adjacent to $V[i]$ and does not contain integer point, we add both vertices of f, $V[i]$ and $V[j]$, to V_P.
3. Check the adjacent facets of all the vertices in V_P, if some of them does not contain integer point go to step 2, until no new fractional vertex can be added to V_P.
4. For every vertex in V_P add any existing "closest integer point" to V_P.
5. In the end, V_P contains several fractional points and at most two integer points and we construct a polyhedral set with V_P as the vertex set.

To compute the integer hull of a corner, we use a brute-force method that searches for all the integer points within the corner polyhedral set and then compute the convex hull of all these points. [8] has showed that, the size and shape of the corner polyhedral set only depends on the coefficients, a_{ij}, of the `relations` of the input but not the constant terms b_i. This implies that the size of the area that we need to do exhaustive search on is not related to the size of the input polyhedral set P so that the time complexity of our algorithm is not related to the volume of the input polyhedral set.

With all the sub-routines introduced above, we present our integer hull algorithm (Algorithm 3) for 2D polyhedral sets. We discuss some of the implementation details in Sect. 6.

4.2 An Example

In this section, we use the following example to show how our 2D algorithm works. The input is a polyhedral set defined by

$$\begin{cases} 2x + 5y \leq 64, \\ -7x - 5y \leq -20, \\ 3x - 6y \leq -7. \end{cases}$$

Algorithm 2: Construct and compute the integer hulls of the corner polyhedral sets

```
1  Function cornerIntegerHulls(V)
       Input:
         – V, the list of the vertices of the input polyhedral set
         – V_C, the output from Algorithm 1

       Output: A list of the vertices of the integer hull of P
2      V_I ← {}
3      for i = 1, ..., n do
4          if V[i] is an integer point then
5              V_I ← V_I ∪ {V[i]}
6          else
7              T ← ConstructCorner(V[i], V_C)
                /* create a corner polyhedral set as we described above */
8              A ← AllIntegerPoints(T)  /* find all the integer points in
               T */
9              V_tmp ← ConvexHull(A)
                        /* compute the vertices of the convex hull of A */
10             V_I ← V_I ∪ {V_tmp}
11     return ConvexHull(V_I)
```

The first step we need to do is to normalize the facets. In this example, there is only one facet which is given by the `relation` $3x - 6y \leq -7$. We replace it with $3x - 6y \leq -9$ (see Fig. 1).

Next we need to find the closest integer points to each fractional vertex on its adjacent facets. In our case, all three vertices are fractional, so we need to find two integer points for each (see Fig. 2a). And as we discussed in Sect. 4, the center part of the input is already an integer hull, so no action needed for this area. As we can see in Fig. 2b, the center part takes most of the volume of the input, by doing so we cut down the size of the problem.

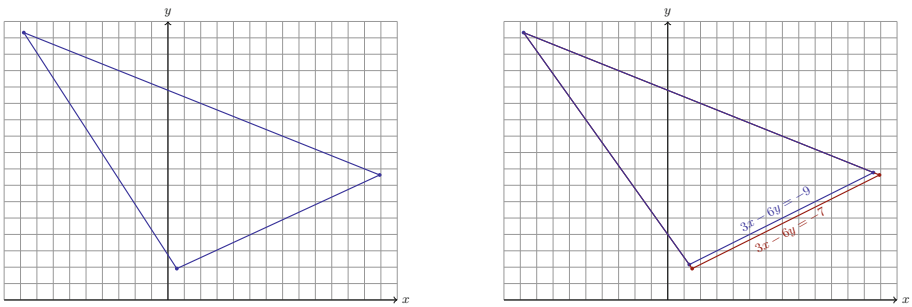
Then we just need to compute the integer hulls of the small corner triangles and use the results to compute the final output (see Fig. 3).

5 Integer Hull of a 3D Polyhedral Set

With the 2D algorithm in place, we can move on to a higher dimension. In this section, we present our integer hull algorithm for 3D polyhedral sets. The general idea behind the algorithm is the same as that of the 2D algorithm. We want to partition the input into smaller polyhedral sets and separate the parts into two categories, the ones with fractional vertices for which we need to compute the

Algorithm 3: Compute the integer hull of a given 2D polyhedral set

```
1 Function IntegerHull2D(P)
      Input: P, a 2D PolyhedralSet object
      Output: I, a list of the vertices of the integer hull of P
2     Process corner cases
3     Q ← Normalization(P)
4     V ← Vertices(Q)
5     V_C ← closestIntegerPoints(V)
6     I ← cornerIntegerHulls(V, V_C)
7     return I
```



(a) Input is a polyhedral set **(b)** Normalize the input

Fig. 1. Input and replaceNonIntegerFacets

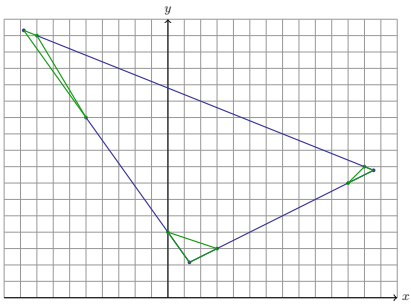
integer hulls as sub-problems and the other ones that are already integer hulls themselves. After processing all the sub-problems, we combine the results of all the parts together and compute the final result.

5.1 Algorithm

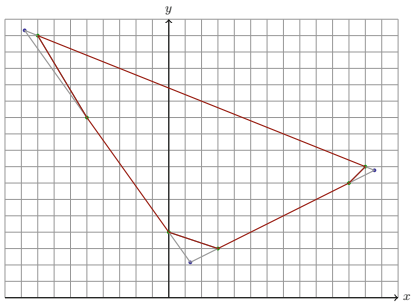
The first step of the 3D algorithm is the same as that in Sect. 4.1. We normalize the facets as is in Sect. 3.1. Similarly, we want to find the closest integer points to the fractional vertices on their adjacent facets. Every fractional vertex and its closest integer points would form a small polyhedral set. For example, Fig. 4a is an example input and the green areas in Fig. 4b are the polyhedral sets formed by fractional vertices and their closest integer points.

Figure 5a shows the center part of the input, this is a polyhedral set with vertices that are all the closest integer points. In the 2D problem, the corner polyhedral sets and the center part formed a partition of the input. But in the 3D case, there are areas that are not covered by these parts, to be precise, these are the areas near the edges (see Fig, 5b).

In order to form a complete partition, we need another set of sub-polyhedral sets. As is shown in Fig. 6a, for an edge that has at least one fractional vertex, the

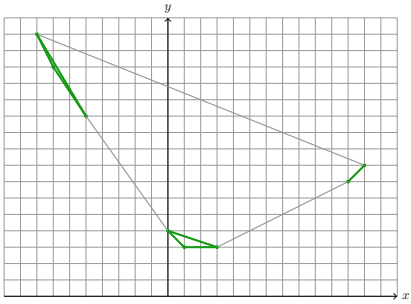


(a) For a fractional vertex, find the integer point on each adjacent facet that is closest to it and construct a triangle with the three points

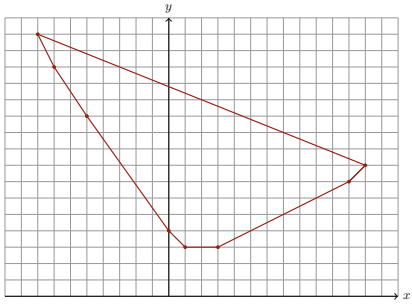


(b) The center part is already an integer hull so we don't need to do anything

Fig. 2. Partition the input



(a) Apply the previous two steps to each fractional vertex



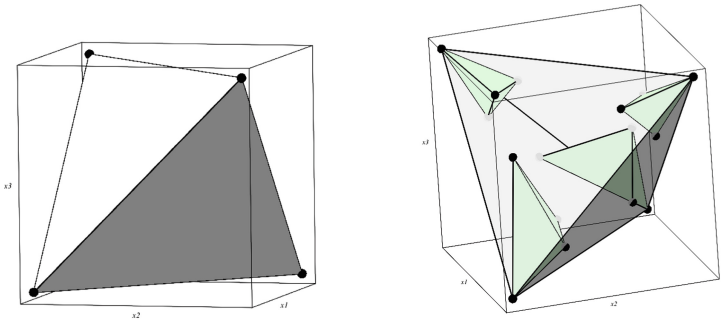
(b) Find the convex hull of all the result vertices from the previous steps

Fig. 3. Compute the integer hulls of the parts and the final result

two vertices of the edge and the closest integer points to the fractional vertices (or vertex) form a polyhedral set. If we construct one such polyhedral set for each edge, we can cover all the missing areas in Fig. 5b.

For the parts that are not integer hulls already, we use a brute-force method to compute their integer hulls, that is, we use exhaustive search to find all the integer points within the part and compute the convex hull of the points. To cut down the area that needs exhaustive search, we further partition the edge polyhedral sets if possible. If there are integer points on an edge, we find the closest one to each fractional vertex and partition the polyhedral set into three parts, see Fig. 6b for an example.

Finding, on a given segment S, the integer point closest to a given vertex of S is relatively simple in the 2D problem, but in the 3D case, we need to address the following, more complicated, question: finding, on a given bounded



(a) Input is a polyhedral set

(b) Every fractional vertex and its closest integer points form a small polyhedral set

Fig. 4. Input and fractional vertices

3D polyhedron F, an integer point closest to a given vertex of F. A natural step towards answering this question is to represent all the integer points of F, which, itself, is an integer hull problem. Since the 3D polyhedron F is "flat", we can project it to a 2D ambient space and use our algorithm from Sect. 4.

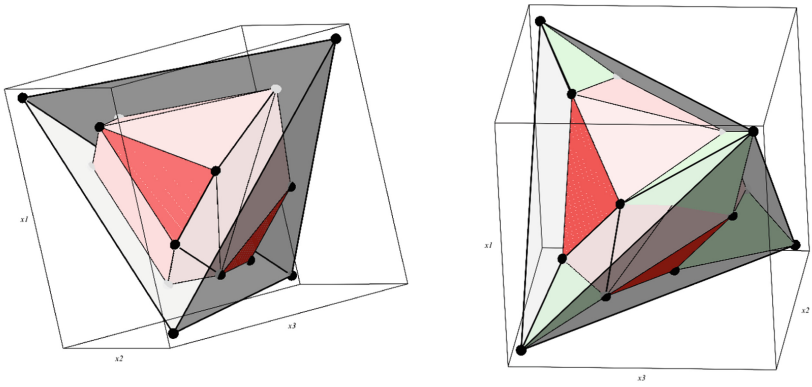
Here we use the procedure `HNFProjection(F, d)` which is introduced in detail in Sect. 3.2. Recall that this procedure will return an ordered pair (G, R_F) where R_F gives the map between a 3D point to a 2D point.

Having a 2D polyhedral set F_P, we use our Algorithm 3 to compute the vertices of the integer hull of F_P. Although the HNF method keeps the integer points in the projection, it can not keep the distance among the points in general, so we must find the original image of the vertices of the integer hull of F_P.

Now that we have the integer hull of a facet, we can search for the closest integer points to each of its vertices. Here we decide to use the closest vertex of the integer hull instead of the actual closest integer point. Using the closest integer vertex might slow down the later steps but only by a very small amount. Searching for the actual point would be another optimization problem and this would be less efficient looking at the whole picture.

As mentioned above, in order to form a complete partition of the input polyhedral set, we need to carefully consider every edge that has at least one fractional vertex. To this end, we use Algorithm 1 to find the closest integer points to a fractional vertex on its adjacent edges. Now that we have all the "closest integer points" we need, we can construct the parts that are the "blue", "red" and "green" regions in Figs. 4, 5 and 6. Since all the vertices of the "red" polyhedral set are integer points, work remains to be done only in the "green" and "blue" polyhedral sets.

Before we present the complete algorithm, there are some corner cases that need to be considered. Similar to our 2D problem, the input polyhedral set could be not fully dimensional. Again we use Hermite Normal Form (HNF) to project



(a) The center part is already an integer hull so we don't need to do anything

(b) There are areas that are not covered by any part

Fig. 5. The center part and the corners

the input to 2D space, and deal with it as a 2D problem. Another corner case would be after applying `Normalization`: no facets have integer points, in this case we use the brute-force approach for the whole input.

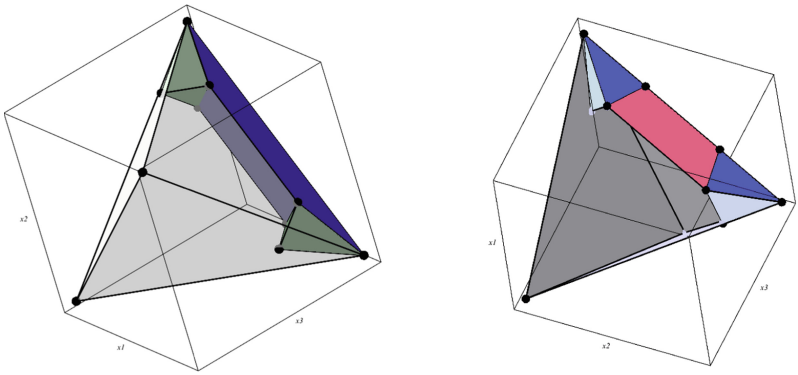
With all the sub-routines in order, here is our algorithm, Algorithm 5, for computing the integer hull of a bounded 3D polyhedral set.

6 Implementation and Experimentation

We have implemented both the 2D and 3D algorithms in both MAPLE and the C/C++ programming language. The MAPLE version is available in 2022 release of MAPLE as the `IntegerHull` command of the `PolyhedralSets` library. In this section, we discuss implementation details and the experimentation with our implementations. All the benchmarks are done on an Intel i5-8300H CPU at 2.30 GHz with 16 GB of memory. As we discussed in Sect. 1, there are studies (such as [11] and [4]) developing approaches to enumerate the vertices of P_I using their relations with the vertices of P but to our knowledge no implementation of such methods exist. So in the following sections we compare our implementation with the existing implementation of the naive method (enumeration of all integer points, followed by the computation of their convex hull) for verification and proof of concept.

6.1 The MAPLE Implementation

For the MAPLE version, we use the functions provided by the `PolyhedralSets` library for polyhedral set manipulation such as construction, getting the vertices and faces. To obtain the adjacency information among the faces we need to compute the face lattice of the input polyhedral set; the `PolyhedralSets` library



(a) The area around an edge

(b) When there are integer points on the edge

Fig. 6. Polyhedral sets that cover the edge areas

provides the command `Graph` for that task. We compare our Maple implementation with another Maple package. In the 2019 Maple Conference, Jing and Moreno-Maza introduced the `ZPolyhedralSets` package, presented in [13]. A `ZPolyhedralSet` is the intersection of a polyhedral set and a lattice. The integer hull of a polyhedral set is equal to a `ZPolyhedralSet` when the `ZPolyhedralSet` is defined using the standard integer lattice (which represents all the points with integer coordinates).

The `ZPolyhedralSet` package provides the `EnumerateIntegerPoints` command, which finds and outputs all the integer points within a `ZPolyhedralSet` object. Given a polyhedral set, to obtain the same result that our algorithm computes, which is the list of the vertices of the integer hull, we use the command `EnumerateIntegerPoints` to find all the integer points within the input, then we the command use `ConvexHull` from `ComputationalGeometry` to compute the vertices.

Table 1 shows the time spent in our algorithm (`IntegerHull`) and the above two-step method (EIP+CH) to obtain the same result. The inputs are triangles with different volumes. As we discussed in Sect. 2, 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. Time spent by our algorithm does not seem to depend on the volume of the input.

From Algorithm 3, we can see that the complexity of our algorithm depends on the number of facets and the number of fractional vertices in the input. Table 2 shows the running time of both algorithms (IntegerHull and EIP+CH) when the inputs are hexagons. The running time for `IntegerHull` is roughly double the time for triangle inputs.

Algorithm 4: Compute the closest integer points on a facet F to the vertices on it in a 3D polyhedral set

Function `closestIntegerPoints3D`(F, V)

Input:
- F, a facet of P in the form of a `PolyhedralSet` object
- V, a list of the vertices of P

Output: V_C, a list where $V_C[i]$ is the integer point on F which is the closest to $V[i]$, if $V[i]$ is in F, and [] otherwise

$F_P, R_F \leftarrow$ `HNFProjection`$(F, 3)$

`/* Make a projection` F_P `of the 3D bounded plane` F `onto 2D space using Hermite Normal Form */`

$V_{\text{tmp}} \leftarrow$ `IntegerHull2D`(F_P)

`/* Find the vertices of the integer hull of` F_P `*/`

$V_F \leftarrow$ `IntegerPointIn3D`(V_{tmp}, R_F)

`/* Find the orignal image of the points in` V_{tmp} `*/` using the map R_F

$n \leftarrow |V|$

for $i = 1, \ldots, n$ **do**

  if $V[i]$ `in` F **and** $V_F \neq []$ **then**

    $V_C[i] \leftarrow$ closest point to $V[i]$ in V_F

  else

    $V_C[i] \leftarrow []$

return V_C

Tables 3 and 4 show the running times of the same two algorithms when the input is a tetrahedron and a bipyramid respectively. The result is similar to that of the 2D algorithm where the running time increases if there are more facets and vertices. One thing that we need to notice is that the running time of our algorithm grows as the volume increases, this is due to the way we deal with the parts that are around the edges. As we discussed in Sect. 5.1, if there is no integer point on an edge, the sub-polyhedral set would include the whole edge and its volume depends on the length of the edge. Recall that we use exhaustive search for the sub-polyhedral sets thus the running time depends on the volume of the input polyhedral set.

6.2 The C/C++ Implementation

For the C/C++ implementation, we follow the representations in the C library `cddlib` by Komei Fukuda [9] for the polyhedral set computations. GMP rational arithmetic is used until the integer coordinates are obtained to ensure correctness. Our implementation can take polyhedral sets in either the V-representation or the H-representation as input; `cddlib` is used for representation conversion and some redundancy removal.

Algorithm 5: Compute the integer hull of a given 3D polyhedral set

```
1  Function IntegerHull3D(P)
       Input: P, a 3D PolyhedralSet object
       Output: I, a list of the vertices of the integer hull of P
2      Process corner case: P is not fully dimensional
3      Q ← Normalization(P)
4      V ← Vertices(Q)
5      P ← Q
6      F ← Facets(P)
7      for each F[i] in F do
8          V_C[i] ← closestIntegerPoints3D(F[i], V)
             /* V_C is a 2D list where V_C[i][j] contains the closest integer
           point to V[j] on F[i] */
9      E ← Edges(P)
10     for each E[i] in E do
11         V_E ← closestIntegerPointsOnEdge(E[i], V)
12     V_list ← PartitionP(V, V_C, V_E)
        /* V_list = [V_1, ..., V_n] where V_i contains the vertices of one part */
13     I ← {}
14     for each V_list[i] in V_list do
15         P_list ← PolyhedralSet(V_list[i])
16         A_I ← AllIntegerPoints(P_list)
17         I ← I ⋃ ConvexHull(A_I)
18     return ConvexHull(I)
```

As we have discussed in Sect. 3.2 we use part of the algorithm in [12] to partition the polyhedral sets and we follow that same article for the enumeration of the integer points in the corners. We implemented Algorithm 2.4.10 in [7] and Algorithm 3 in [12] for the procedure `HNFProjection`. We also implemented the algorithm introduced by Kaibel and Pfetsch in [14] for the computations of the face lattice.

To verify our implementation, we compare our results with that of the Normaliz library [5]. We also implemented a naive procedure based on enumeration and convex hull computation to obtain the integer hull. Note that Algorithm 3 in [12] only enumerate the integer points inside the given polyhedral set while for Normaliz, if the input is not homogeneous Normaliz homogenizes it by raising the input to a higher dimension, therefore, Normaliz enumerates more points than we do for the same input.

Tables 5 and 6 show the time spent in these three different approaches for computing the integer hulls of the same inputs. Since the I/O formats are different for Normaliz and `cddlib`, we only measured the timings for the integer hull computation part but not the I/O parts of the programs. Especially, for Normaliz we only timed the function call "MyCone.compute(ConeProperty::IntegerHull)".

Table 1. Integer hulls of triangles

Volume	27.95		111.79		11179.32	
Algorithm	IntegerHull	EIP+CH	IntegerHull	EIP+CH	IntegerHull	EIP+CH
Time(s)	0.172	0.410	0.244	0.890	0.159	58.083

Table 2. Integer hulls of hexagons

Volume	58.21		5820.95		23283.82	
Algorithm	IntegerHull	EIP+CH	IntegerHull	EIP+CH	IntegerHull	EIP+CH
Time(s)	0.303	0.752	0.275	31.357	0.304	123.159

The examples are named as xdy_z, where x is the dimension of the input (all the examples are full dimensional). Each y represents a set of examples that are of the same shape which means these polyhedral sets $A\,\mathbf{x} <= \mathbf{b}$ share the same coefficient matrix A while the vector $\mathbf{b}$ varies. xdy_0 is the smallest (volume wise) example in a set, for $z = 1, 2, 3$, vector $\mathbf{b}$ get multiplied by $2, 5, 10$ respectively. For the 2D examples, $2d1$ has 6 vertices, $2d2$ has 4 vertices and $2d3$ has 3 vertices. And for the 3D examples, $3d1$ has 12 facets, 8 vertices and 18 edges, $3d2$ has 4 facets, 4 vertices and 6 edges and $3d3$ has 6 facets, 5 vertices and 9 edges.

The result is consistent with our observation in [17]. For the same family of input, the time spent by our algorithm is relatively stable while for both our naive implementation and Normaliz, the larger the volume of the input is, the more time they need to do the computation since often time larger polyhedral sets contain more integer points for enumeration.

Table 3. Integer hulls of tetrahedrons (4 facets, 4 vertices and 6 edges)

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 4. Integer hulls of triangular bipyramids (6 facets, 5 vertices and 9 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 5. Timing (ms) for computing integer hull of 2D examples

example	IntegerHull	Naive	Normaliz
2d1_0	0.451	0.565	2.837
2d1_1	0.478	0.657	1216.238
2d1_2	0.396	0.682	740.559
2d1_3	0.443	1.134	472.447
2d2_0	0.413	1.128	1258.422
2d2_1	0.411	2.714	1242.081
2d2_2	0.393	16.079	2622.995
2d2_3	0.449	47.145	10218.368
2d3_0	0.284	0.768	835.730
2d3_1	0.339	1.676	462.116
2d3_2	0.286	6.883	1559.401
2d3_3	0.324	25.637	5072.894

Table 6. Timing (ms) for computing integer hull of 3D examples

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

7 Conclusion and Future Work

In this paper, we introduced a new algorithm for computing the integer hull of a convex polyhedral set. Our algorithm takes into consideration geometric properties of the input polyhedral set in order to make the computation more efficient. We implemented the proposed algorithm for two-dimensional and three-dimensional input in both MAPLE and C/C++. The efficiency of this algorithm depends mainly on the shape of the input while the size of the input has little

impact. We show in Sect. 6 that our algorithm can deal with inputs of very large volumes that algorithms depending on enumeration can not process.

The main steps of our algorithm are normalization, partition and merging. Our algorithm can be stated for polyhedral sets of arbitrary dimension and a MAPLE implementation is work in progress. Another on-going development is an algebraic complexity analysis of our algorithm.

We sketch below our algorithm for computing the integer hull of a d-dimensional convex polyhedral set P:

1. normalize the input using the procedure introduced in Sect. 3.1,
2. for each vertex, find the closest integer points to it on each of its adjacent faces,
3. for each face of dimension from 0 to $d-2$, construct a "corner polyhedral set" using the integer points we obtained from step 2,
4. compute the integer hull of each corner polyhedral set,
5. compute the convex hull of all the integer hulls from step 4,
6. this convex hull is the integer hull of P.

References

1. Assarf, B., Gawrilow, E., Herr, K., Joswig, M., Lorenz, B., Paffenholz, A., Rehn, T.: Computing convex hulls and counting integer points with polymake. Math. Program. Comput. **9**(1), 1–38 (2017)
2. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. **22**(4), 469–483 (1996)
3. Barvinok, A.I.: A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. Math. Oper. Res. **19**(4), 769–779 (1994)
4. Berndt, S., Jansen, K., Klein, K.: New bounds for the vertices of the integer hull. In: Le, H.V., King, V. (eds.) 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11–12, 2021. pp. 25–36. SIAM (2021)
5. Bruns, W., Ichim, B., Römer, T., Sieg, R., Söger, C.: Normaliz: algorithms for rational cones and affine monoids . J. Algebra **324** (2010)
6. Chvátal, V.: Edmonds polytopes and a hierarchy of combinatorial problems. Discret. Math. **4**(4), 305–337 (1973)
7. Cohen, H.: A Course in Computational Algebraic Number Theory, vol. 8. Springer-Verlag, Berlin (1993). https://doi.org/10.1007/978-3-662-02945-9
8. Cook, W.J., Hartmann, M., Kannan, R., McDiarmid, C.: On integer points in polyhedra. Combinatorica **12**(1), 27–37 (1992)
9. Fukuda, K.: cdd. c: C-implementation of the double description method for computing all vertices and extremal rays of a convex polyhedron given by a system of linear inequalities. Department of Mathematics, Swiss Federal Institute of Technology, Lausanne, Switzerland (1993)
10. Gomory, Ralph E..: Outline of an algorithm for integer solutions to linear programs *and* an algorithm for the mixed integer problem. In: Jünger, M., et al. (eds.) 50 Years of Integer Programming 1958-2008, pp. 77–103. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-540-68279-0_4
11. Hayes, A.C., Larman, D.G.: The vertices of the knapsack polytope. Discret. Appl. Math. **6**(2), 135–138 (1983)

12. Jing, R.-J., Moreno Maza, M.: Computing the integer points of a polyhedron, I: algorithm. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) CASC 2017. LNCS, vol. 10490, pp. 225–241. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66320-3_17
13. Jing, R., Moreno Maza, M.: The z_polyhedra library in maple. In: Gerhard, J., Kotsireas, I.S. (eds.) Maple in Mathematics Education and Research - Third Maple Conference, MC 2019, Waterloo, Ontario, Canada, October 15–17, 2019, Proceedings of the Communications in Computer and Information Science, vol. 1125, pp. 132–144. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-81698-8
14. Kaibel, V., Pfetsch, M.E.: Computing the face lattice of a polytope from its vertex-facet incidences. Comput. Geom. **23**(3), 281–290 (2002)
15. Land, A., Doig, A.: An automatic method of solving discrete programming problems. Econometric **28**, 497–520 (1960)
16. Loera, J.A.D., Hemmecke, R., Tauzer, J., Yoshida, R.: Effective lattice point counting in rational convex polytopes. J. Symb. Comput. **38**(4), 1273–1302 (2004)
17. Moreno Maza, M., Wang, L.: On the pseudo-periodicity of the integer hull of parametric convex polygons. In: Boulier, F., England, M., Sadykov, T.M., Vorozhtsov, E.V. (eds.) CASC 2021. LNCS, vol. 12865, pp. 252–271. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85165-1_15
18. Pick, G.: Geometrisches zur zahlenlehre. Sitzenber. Lotos (Prague) **19**, 311–319 (1899)
19. Maple polyhedralsets package (2021), https://www.maplesoft.com/support/help/maple/view.aspx?path=PolyhedralSets
20. Schrijver, A. (Ed.): Theory of Linear and Integer Programming. Wiley, New York (1986)
21. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, New York (1999)
22. Schrijver, A., et al.: On cutting planes. Combinatorics **79**, 291–296 (1980)
23. Seghir, R., Loechner, V., Meister, B.: Integer affine transformations of parametric Z-polytopes and applications to loop nest optimization. ACM Trans. Archit. Code Optim. **9**(2), 8:1–8:27 (2012)
24. Thomas, R.R.: Integer programming: Algebraic methods. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, 2nd edn., pp. 1624–1634. Springer, Boston (2009). https://doi.org/10.1007/978-0-387-74759-0
25. Verdoolaege, S., Seghir, R., Beyls, K., Loechner, V., Bruynooghe, M.: Counting integer points in parametric polytopes using Barvinok's rational functions. Algorithmica **48**(1), 37–66 (2007)
26. Veselov, S., Chirkov, A.Y.: Some estimates for the number of vertices of integer polyhedra. J. Appl. Ind. Math. **2**(4), 591–604 (2008)
27. Yanagisawa, H.: A simple algorithm for lattice point counting in rational polygons (2005)