An Output-sensitive Algorithm for Computing Projections of Resultant Polytopes

Ioannis Emiris and Vissarion Fisikopoulos and Christos Konaxis and Luis Penaranda

An Output-sensitive Algorithm for Computing Projections of Resultant Polytopes

Ioannis Z. Emiris* Vissarion Fisikopoulos* Christos Konaxis†
Luis Peñaranda*

June 27, 2012

Abstract

We develop an incremental algorithm to compute the Newton polytope of the resultant, aka resultant polytope, or its projection along a given direction. The resultant is fundamental in algebraic elimination and in implicitization of parametric hypersurfaces. Our algorithm exactly computes vertex- and halfspace-representations of the desired polytope using an oracle producing resultant vertices in a given direction. It is output-sensitive as it uses one oracle call per vertex. We overcome the bottleneck of determinantal predicates by hashing, thus accelerating execution from 18 to 100 times. We implement our algorithm using the experimental CGAL package `triangulation`. A variant of the algorithm computes successively tighter inner and outer approximations: when these polytopes have, respectively, 90% and 105% of the true volume, runtime is reduced up to 25 times. Our method computes instances of 5-, 6- or 7-dimensional polytopes with 35K, 23K or 500 vertices, resp., within 2hr. Compared to tropical geometry software, ours is faster up to dimension 5 or 6, and competitive in higher dimensions.

Keywords General Dimension, Convex Hull, Regular Triangulation, Secondary Polytope, Resultant, CGAL Implementation, Experimental Complexity.

1 Introduction

Given pointsets $A_0, \ldots, A_n \subset \mathbb{Z}^n$, we define the pointset

$$\mathcal{A} := \bigcup_{i=0}^{n} (A_i \times \{e_i\}) \subset \mathbb{Z}^{2n}, \tag{1}$$

where $e_0, \ldots, e_n$ form an affine basis of $\mathbb{R}^n$: e_0 is the zero vector, $e_i = (0, \ldots, 0, 1, 0, \ldots, 0), i = 1, \ldots, n$. Clearly, $|\mathcal{A}| = |A_0| + \cdots + |A_n|$, where $|\cdot|$ denotes cardinality. By Cayley's trick (Prop. 2) the regular tight mixed subdivisions of the Minkowski sum $A_0 + \cdots + A_n$ are in bijection with the regular triangulations of $\mathcal{A}$, which are the vertices of the *secondary polytope* $\Sigma(\mathcal{A})$.

The *Newton polytope* of a polynomial is the convex hull of its *support*, i.e. the exponent vectors of monomials with nonzero coefficient. It subsumes the notion of degree for sparse multivariate polynomials by providing more information, unless the polynomial is completely dense. Given $n+1$ polynomials in n variables, with fixed supports A_i and symbolic coefficients, their *sparse (or toric)*

*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece. {emiris,vissarion,lpenaranda}@di.uoa.gr. Partial support from project "Computational Geometric Learning", which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 255827.

†Archimedes Center for Modeling, Analysis & Computation (ACMAC), University of Crete, Heraklio, Greece. ckonaxis@di.uoa.gr. Enjoys support from the FP7-REGPOT-2009-1 project "Archimedes Center for Modeling, Analysis and Computation"; most of the work in this paper was done while at the University of Athens under support of "Computational Geometric Learning".

resultant $\mathcal{R}$ is a polynomial in these coefficients which vanishes exactly when the polynomials have a common root (Def. 1). The resultant is the most fundamental tool in elimination theory, and is instrumental in system solving; it is also important in changing representation of parametric hypersurfaces.

The Newton polytope of the resultant $N(\mathcal{R})$, or *resultant polytope*, is the object of our study, especially when some of the input coefficients are not symbolic, in which case we seek a projection of the resultant polytope. The lattice points in $N(\mathcal{R})$ yield a superset of the support of $\mathcal{R}$; this reduces implicitization [14, 30] and computation of $\mathcal{R}$ to sparse interpolation (Sect. 2). The number of coefficients of the $n+1$ polynomials ranges from $O(n)$ to $O(n^d d^n)$, where d bounds their total degree. In system solving and implicitization, one computes $\mathcal{R}$ when all but $O(n)$ of the coefficients are specialized to constants, hence the need for resultant polytope projections.

The resultant polytope is a Minkowski summand of $\Sigma(\mathcal{A})$. For its construction, we exploit an equivalence relation defined on the secondary vertices, where the classes are in bijection with the vertices of the resultant polytope. This yields an oracle producing a resultant vertex in a given direction, thus avoiding to compute $\Sigma(\mathcal{A})$, which has much more vertices than $N(\mathcal{R})$. Although there exist efficient software for $\Sigma(\mathcal{A})$ [28], it is useless in computing resultant polytopes. For instance, in implicitizing parametric surfaces with < 100 input terms, we compute the Newton polytope of the equations in $<$ 1sec, which includes all common instances in geometric modeling, whereas $\Sigma(\mathcal{A})$ is intractable.

Our main contribution is twofold. First, we design an output-sensitive algorithm for computing the Newton polytope of $\mathcal{R}$, or of specializations of $\mathcal{R}$. The algorithm computes both vertex (V) and halfspace (H) representations, which is important for the targeted applications. Its incremental nature implies that we also obtain a triangulation of the polytope, which may be useful for enumerating its lattice points in subsequent applications. The complexity is proportional to the number of output vertices and facets; the overall cost is dominated by computing as many regular triangulations of $\mathcal{A}$ (Thm. 10). We work in the space of the projected $N(\mathcal{R})$ and revert to the high-dimensional space of $\Sigma(\mathcal{A})$ only if needed. Our algorithm readily extends to computing $\Sigma(\mathcal{A})$, projections of $\Sigma(\mathcal{A})$ and, more generally, any polytope that can be efficiently described by a vertex oracle. A variant of our algorithm computes successively tighter inner and outer approximations: typically, these polytopes have, respectively, 90% and 105% of the true volume, while runtime is reduced up to 25 times. This may lead to an approximation algorithm.

Second, we describe an efficient implementation based on CGAL [7] and the experimental package `triangulation`. Our method computes instances of 5-, 6- or 7-dimensional polytopes with 35K, 23K or 500 vertices, respectively, in $<$ 2hr. Our code is faster up to dimensions 5 or 6, and competitive in higher dimensions, to a method computing $N(\mathcal{R})$ via tropical geometry and based on the `Gfan` library [21]. Moreover, our code in the critical step of computing convex hulls, uses `triangulation` which compared to state-of-the-art software `lrs`, `cdd`, and `polymake`, is the fastest together with `polymake`. We factor out repeated computation by reducing the bulk of our work to a sequence of determinants: this is often the case in high-dimensional geometric computing. Here, we exploit the nature of our problem to capture the similarities of the predicates, and hash the computed minors which are needed later, to speedup subsequent determinants. This is of independent interest and can be used to improve other related problems such as convex hull and triangulation computations.

Let us review previous work. Sparse (or toric) elimination theory was introduced in [18]. They show that $N(\mathcal{R})$, for two univariate polynomials with k_0+1, k_1+1 monomials, has $\binom{k_0+k_1}{k_0}$ vertices and, when both $k_i \geq 2$, it has k_0k_1+3 facets. In [29, Sec.6] is proven that $N(\mathcal{R})$ is 1-dimensional iff $|A_i| = 2$, for all i, the only planar $N(\mathcal{R})$ is the triangle, whereas the only 3-dimensional ones are the tetrahedron, the square-based pyramid, and the polytope of two univariate trinomials; we compute an instance of the latter (Fig. 2(b)). Following [29, Thm.6.2], the 4-dimensional polytopes include the 4-simplex, some $N(\mathcal{R})$ obtained by pairs of univariate polynomials, and those of 3 trinomials, which we can investigate with our code: the maximal such polytope we have computed has f-vector $(22, 66, 66, 22)$ (Fig. 2(c)).

In [26] they describe all Minkowski summands of $\Sigma(A)$. In [27] is defined an equivalence class over $\Sigma(A)$ vertices having the same mixed cells. The classes map in a many-to-1 fashion to

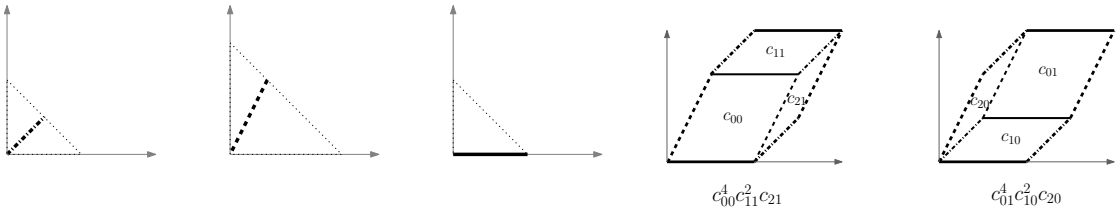

Figure 1: Let $f_0 := c_{00} - c_{01}x_1x_2$, $f_1 := c_{10} - c_{11}x_1x_2^2$, $f_2 := c_{20} - c_{21}x_1^2$. Left: Newton polygons and those of fully dense polynomials of same total degree (dashed). Right: There are 2 regular tight mixed subdivisions of the Minkowski sum giving the extreme monomials of $\mathcal{R} = -c_{00}^4 c_{11}^2 c_{21} + c_{01}^4 c_{10}^2 c_{20}$.

resultant vertices; our algorithm exploits a stronger equivalence relationship. Tropical geometry is a polyhedral analogue of algebraic geometry which gives alternative ways of recovering resultant polytopes [21] and Newton polytopes of implicit equations [30]. Sect. 5 discusses it and shows our approach is faster up to dimensions $5, 6$ and competitive in higher dimensions.

We focus on sequences of determinantal predicates. For determinants, the record bit complexity is $O(n^{2.697})$ [23]. Methods exist for the sign of general determinants, e.g. [6]. We compared linear algebra libraries LinBox [11] and Eigen [19], which seem most suitable in dimension > 100 and medium-high dimensions, respectively, whereas CGAL provides the most efficient determinant computation for the dimensions to which we focus.

The roadmap of the paper follows: Sect. 2 describes the combinatorics of resultants, and the following section presents our algorithm. Sect. 4 overcomes the bottleneck of Orientation predicates. Sect. 5 discusses the implementation, experiments, and comparison with other software. We conclude with future work.

2 Resultant polytopes and their projections

We introduce tools from combinatorial geometry [25, 31] to describe resultants [9, 18]. Let $\mathrm{vol}(\cdot) \in \mathbb{N}$ denote normalized Euclidean volume, and $(\mathbb{R}^m)^\times$ the linear m-dimensional functionals.

Let $\mathcal{A} \subset \mathbb{R}^d$ be a pointset whose convex hull is of dimension d. For any triangulation T of $\mathcal{A}$, define vector $\phi_T \in \mathbb{R}^{|\mathcal{A}|}$ with coordinate

$$\phi_T(a) = \sum_{\sigma \in T : a \in \sigma} \mathrm{vol}(\sigma), \qquad a \in \mathcal{A}, \tag{2}$$

summing over all simplices σ of T having a as a vertex; $\Sigma(\mathcal{A})$ is the convex hull of ϕ_T for all triangulations T. Let $\mathcal{A}^w$ denote pointset $\mathcal{A}$ lifted to $\mathbb{R}^{d+1}$ via a generic lifting function w in $(\mathbb{R}^{|\mathcal{A}|})^\times$. *Regular triangulations* of $\mathcal{A}$ are obtained by projecting the upper (or lower) hull of $\mathcal{A}^w$ back to $\mathbb{R}^d$.

Proposition 1. [18] *The vertices of $\Sigma(\mathcal{A})$ correspond to the regular triangulations of $\mathcal{A}$, while its face lattice corresponds to the poset of regular polyhedral subdivisions of $\mathcal{A}$, ordered by refinement. A lifting vector produces regular triangulation T (resp. a regular polyhedral subdivision of $\mathcal{A}$) iff it lies in the normal cone of vertex ϕ_T (resp. of the corresponding face) of $\Sigma(\mathcal{A})$. The dimension of $\Sigma(\mathcal{A})$ is $|\mathcal{A}| - d - 1$.*

Let $A_0, \ldots, A_n$ be subsets of $\mathbb{Z}^n$, $P_0, \ldots, P_n \subset \mathbb{R}^n$ their convex hulls, and $P = P_0 + \cdots + P_n$ their Minkowski sum. A *Minkowski (maximal) cell* of P is any full-dimensional convex polytope $B = \sum_{i=0}^n B_i$, where each B_i is a convex polytope with vertices in A_i. Minkowski cells $B, B' = \sum_{i=0}^n B_i'$ intersect properly when $B_i \cap B_i'$ is a face of both and their Minkowski sum descriptions are compatible, i.e. coincide on the common face. A *mixed subdivision* of P is any family of Minkowski cells which partition P and intersect properly. A cell is *i-mixed* or *v_i-mixed*, if it is the Minkowski sum of n one-dimensional segments from P_j, $j \neq i$, and some vertex $v_i \in P_i$.

Mixed subdivisions contain *faces* of all dimensions between 0 and n, the maximum dimension corresponding to cells. Every face of a mixed subdivision of P has a unique description as Minkowski sum of $B_i \subset P_i$. A mixed subdivision is *regular* if it is obtained as the projection of the upper (or lower) hull of the Minkowski sum of lifted polytopes $P_i^{w_i} := \{(p_i, w_i(p_i)) \mid p_i \in P_i\}$, for lifting $w_i : P_i \to \mathbb{R}$. If the lifting function $w := (w_0 \ldots, w_n)$ is sufficiently generic, then the mixed subdivision is *tight*, and $\sum_{i=0}^n \dim B_i = \dim \sum_{i=0}^n B_i$, for every cell. Given $A_0, \ldots, A_n$ and the affine basis $\{e_0, \ldots, e_n\}$ of $\mathbb{R}^n$, we define the Cayley pointset $\mathcal{A} \subset \mathbb{Z}^{2n}$ as in equation (1).

Proposition 2. [Cayley trick] [18] *There exist bijections between: the regular tight mixed subdivisions of P and the regular triangulations of $\mathcal{A}$; the tight mixed subdivisions of P and the triangulations of $\mathcal{A}$; the mixed subdivisions of P and the polyhedral subdivisions of $\mathcal{A}$.*

The family $A_0, \ldots, A_n \subset \mathbb{Z}^n$ is *essential* if they jointly affinely span $\mathbb{Z}^n$ and every subset of cardinality $j, 1 \leq j < n$, spans a space of dimension $\geq j$. It is straightforward to check this property algorithmically and, if it does not hold, to find an essential subset. In the sequel, the input $A_0, \ldots, A_n \subset \mathbb{Z}^n$ is supposed to be essential.

Definition 1. *Let $A_0, \ldots, A_n \subset \mathbb{Z}^n$ be an essential family, and $f_0, \ldots, f_n \in \mathbb{C}[x_1, \ldots, x_n]$ polynomials with these supports and symbolic coefficients $c_{ij}, i = 0, \ldots, n,\ j = 1, \ldots, |A_i|$, i.e. $f_i = \sum_{a \in A_i} c_{ij} x^a$. Their sparse (or toric) resultant is a polynomial in $\mathbb{Z}[c_{ij} : i = 0, \ldots, n,\ j = 1, \ldots, |A_i|]$, defined up to sign, which vanishes iff $f_0 = f_1 = \cdots = f_n = 0$ has a common root in $(\mathbb{C}^*)^n$, $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$.*

For $n = 1$, the resultant is named after Sylvester. For linear systems, it equals the determinant of the $(n+1) \times (n+1)$ coefficient matrix. The discriminant of a polynomial $F(x_1, \ldots, x_n)$ is given by the resultant of $F, \partial F/\partial x_1, \ldots, \partial F/\partial x_n$.

The Newton polytope $N(\mathcal{R})$ of the resultant is a lattice polytope called *resultant polytope*: the resultant has $|\mathcal{A}| = \sum_{i=0}^n |A_i|$ variables, hence $N(\mathcal{R})$ lies in $\mathbb{R}^{|\mathcal{A}|}$, though of smaller dimension (Prop. 4). The monomials corresponding to vertices of $N(\mathcal{R})$ are the extreme resultant monomials. For a sufficiently generic lifting $w \in (\mathbb{R}^{|\mathcal{A}|})^\times$, the w-extreme resultant monomial, whose exponent vector maximizes inner product with w, is recovered from the regular tight mixed subdivision S of P induced by w. Let T be the regular triangulation corresponding, via the Cayley trick, to S, and $\rho_T \in \mathbb{N}^{|\mathcal{A}|}$ the exponent of the w-extreme monomial. Then,

$$\rho_T(a) = \sum_{\substack{a-\text{mixed} \\ \sigma \in T : a \in \sigma}} \text{vol}(\sigma) \ \in \mathbb{N}, \qquad a \in \mathcal{A}, \tag{3}$$

where simplex σ is a-mixed iff the corresponding cell is a-mixed in S. Note that, $\rho_T(a) \in \mathbb{N}$, since it is a sum of volumes of mixed cells σ, each of them equal to the *mixed volume* of a set of *lattice* polytopes, the Minkowksi summands of σ:

$$\text{vol}(\sigma) = MV(\sigma_0, \ldots, \sigma_n), \quad \sigma = \sigma_0 + \cdots + \sigma_n,\ \sigma_i \subset \mathcal{A}_i,$$

where MV denotes the mixed volume function. Now, $N(\mathcal{R})$ is the convex hull of all ρ_T vectors [18, 29].

There exists a many-to-1 surjection from vertices of $\Sigma(\mathcal{A})$ to those of $N(\mathcal{R})$. One defines an *equivalence relationship* on all regular tight mixed subdivisions, where equivalent subdivisions yield the same vertex in $N(\mathcal{R})$. Thus, equivalent vertices of $\Sigma(\mathcal{A})$ correspond to the same resultant vertex. Consider $w \in (\mathbb{R}^{|\mathcal{A}|})^\times$ lying in the union of outer-normal cones of equivalent vertices of $\Sigma(\mathcal{A})$. They correspond to a resultant vertex whose outer-normal cone contains w; this defines a w-extremal resultant monomial. If w is non-generic, it specifies a sum of extremal monomials in $\mathcal{R}$, i.e. a face of $N(\mathcal{R})$ (Fig. 2(a),(b)).

We compute some orthogonal projection of $N(\mathcal{R})$, denoted Π, in $\mathbb{R}^m$:

$$\pi : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^m : N(\mathcal{R}) \to \Pi,\ \ m \leq |\mathcal{A}|.$$

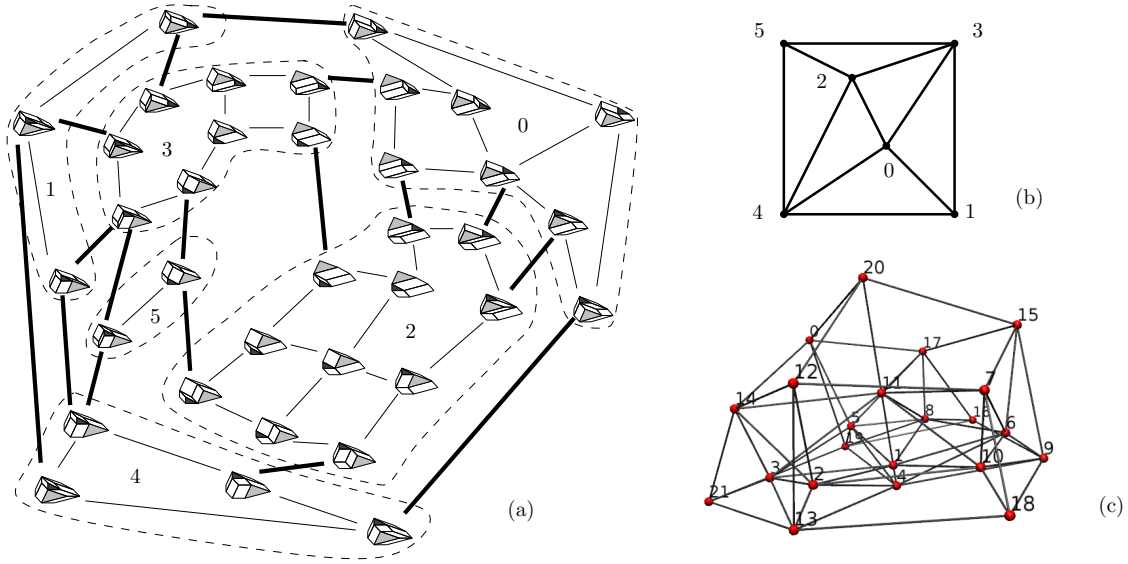

Figure 2: (a) Σ polytope of two triangles (dark, light grey) and one segment $A_0 = \{(0,0),(1,2),(4,1)\}$, $A_1 = \{(0,1),(1,0)\}$, $A_2 = \{(0,0),(0,1),(2,0)\}$; vertices correspond to mixed subdivisions of the Minkowski sum $A_0+A_1+A_2$ and edges to flips between them (b) $N(\mathcal{R})$, whose vertices correspond to the dashed classes of Σ. Bold edges of Σ, called cubical flips, map to edges of $N(\mathcal{R})$ (c) 4-dimensional $N(\mathcal{R})$ of 3 generic trinomials with f-vector $(22, 66, 66, 22)$; figure made with `polymake`.

By reindexing, this is the subspace of the first m coordinates, so $\pi(\rho) = (\rho_1, \ldots, \rho_m)$. It is possible that none of the coefficients c_{ij} is specialized, hence $m = |\mathcal{A}|$, π is trivial, and $\mathit{\Pi} = N(\mathcal{R})$. Assuming the specialized coefficients take sufficiently generic values, $\mathit{\Pi}$ is the Newton polytope of the corresponding specialization of $\mathcal{R}$. The following is used for preprocessing.

Lemma 3. [21, Lem.3.20] *If $a_{ij} \in A_i$ corresponds to a specialized coefficient of f_i, and lies in the convex hull of the other points in A_i corresponding to specialized coefficients, then removing a_{ij} from A_i does not change the Newton polytope of the specialized resultant.*

Proposition 4. [18] *$N(\mathcal{R})$ is a Minkowski summand of $\Sigma(\mathcal{A})$, of dimension $|\mathcal{A}| - 2n - 1$.*

Let us describe the $2n+1$ hyperplanes in whose intersection lies $N(\mathcal{R})$. For this, let M be the $(2n+1) \times |\mathcal{A}|$ matrix whose columns are the points in the A_i, where each $a \in A_i$ is followed by the i-th unit vector in $\mathbb{N}^{n+1}$. Then, the inner product of any coordinate vector of $N(\mathcal{R})$ with row i of M is: constant, for $i = 1, \ldots, n$, and known, and depends on i, for $i = n+1, \ldots, 2n+1$ [18, Prop.7.1.11]. This implies that one obtains an isomorphic polytope when projecting $N(\mathcal{R})$ along $2n+1$ points in $\mathcal{A}$ which affinely span $\mathbb{R}^{2n}$; this is possible because of the assumption of essential family. Having computed the projection, we obtain $N(\mathcal{R})$ by computing the missing coordinates as the solution of a linear system: we write the aforementioned inner products as $M[X\ V]^T = C$, where C is a known matrix and $[X\ V]^T$ is a transposed $(2n+1) \times u$ matrix, expressing the partition of the coordinates to unknown and known values, where u is the number of $N(\mathcal{R})$ vertices. If the first $2n+1$ columns of M correspond to specialized coefficients, $M = [M_1\ M_2]$, where submatrix M_1 is of dimension $2n+1$ and invertible, hence $X = M_1^{-1}(C - M_2 B)$.

We focus on 3 applications. First, we interpolate the resultant in all coefficients, thus illustrating an alternative method for computing resultants.

Example 1. *Let $f_0 = a_2x^2 + a_1x + a_0$, $f_1 = b_1x^2 + b_0$, with supports $A_0 = 2, 1, 0, A_1 = 1, 0$. Their (Sylvester) resultant is a polynomial in a_2, a_1, a_0, b_1, b_0. Our algorithm computes its Newton polytope with vertices $(0, 2, 0, 1, 1)$, $(0, 0, 2, 2, 0)$, $(2, 0, 0, 0, 2)$; it contains 4 points, corresponding*

to 4 potential monomials $a_1^2b_1b_0$, $a_0^2b_1^2$, $a_2a_0b_1b_0$, $a_2^2b_0^2$. *There is a parameterization of resultants [24], which yields:* $a_2 = (2t_1 + t_2)t_3^2t_4$, $a_1 = (-2t_1 - 2t_2)t_3t_4$, $a_0 = t_2t_4$, $b_1 = -t_1t_3^2t_5$, $b_0 = t_1t_5$, *where the* t_i*'s are parameters. We substitute these expressions to the monomials, evaluate at 4 sufficiently random* t_i*'s, and obtain a matrix whose kernel vector* $(1, 1, -2, 1)$ *yields* $\mathcal{R} = a_1^2b_1b_0 + a_0^2b_1^2 - 2a_2a_0b_1b_0 + a_2^2b_0^2$. □

Second, consider system solving by the rational univariate representation of roots [3]. Given $f_1, \ldots, f_n \in \mathbb{C}[x_1, \ldots, x_n]$, define an overconstrained system by adding $f_0 = u_0 + u_1x_1 + \cdots + u_nx_n$ with symbolic u_i's. Let coefficients $c_{ij}, i \geq 1$, take specific values, and suppose the roots of $f_1 = \cdots = f_n = 0$ are isolated, denoted $r_i = (r_{i1}, \ldots, r_{in})$. Then the u-resultant is $\mathcal{R}_u = a\prod_{r_i}(u_0 + u_1r_{i1} + \cdots + u_nr_{in})^{m_i}$, $a \in \mathbb{C}^*$, where m_i is the multiplicity of r_i. Computing $\mathcal{R}_u$ is the bottleneck; our method computes (a superset of) $N(\mathcal{R}_u)$.

The last application comes from geometric modeling, where $y_i = f_i(x)$, $i = 0, \ldots, n$, $x = (x_1, \ldots, x_n) \in \Omega \subset \mathbb{R}^n$, defines a parametric hypersurface. Many applications require the equivalent implicit representation $F(y_1, \ldots, y_n) = 0$. This amounts to eliminating x, so it is crucial to compute the resultant when coefficients are specialized except the y_i's. Our approach computes a polytope that contains the Newton polytope of F, thus reducing implicitization to interpolation [14]. In particular, we compute the polytope of surface equations within 1sec, assuming < 100 terms in parametric polynomials, which includes all common instances in geometric modeling, whereas the corresponding $\Sigma(\mathcal{A})$ are intractable.

3 Algorithms and complexity

This section analyzes our exact and approximate algorithms for computing orthogonal projections of polytopes whose vertices are defined by an *oracle*. This oracle computes the vertex of the polytope which is extremal in a given direction w. If there are more than one vertices extremal in w it computes exactly one of these. Moreover, we define such an oracle for the vertices of orthogonal projections Π of $N(\mathcal{R})$ which results to algorithms for computing Π without computing $N(\mathcal{R})$. Finally, we analyze the asymptotic complexity of these algorithms.

Given pointset V, reg_subdivision(V, w) computes the regular subdivision of its convex hull by projecting the upper hull of V lifted by w, and conv(V) computes the H-representation of its convex hull. The oracle Vtx$(\mathcal{A}, w, \pi)$ computes a point in Π, extremal in the direction w, by refining the output of reg_subdivision$(\mathcal{A}, \widehat{w})$, $\widehat{w} = (w, \vec{0}) \in (\mathbb{R}^{|\mathcal{A}|})^\times$, into a regular triangulation T of $\mathcal{A}$, then returning $\pi(\rho_T)$. It is clear that, triangulation T constructed by Vtx, is regular and corresponds to some secondary vertex ϕ_T which maximizes the inner product with $\widehat{w}$.

Lemma 5. *All points computed by Vtx are vertices of* Π.

Proof. Let $v = \pi(\rho_T) = \text{Vtx}(\mathcal{A}, w, \pi)$. We first prove that v lies on $\partial\Pi$. Point ρ_T of $N(\mathcal{R})$ is a Minkowski summand of vertex ϕ_T of $\Sigma(\mathcal{A})$ extremal wrt $\widehat{w}$, hence ρ_T is extremal wrt $\widehat{w}$. Since $\widehat{w}$ is perpendicular to projection π, ρ_T projects to a point in $\partial\Pi$. The same argument implies that every vertex ϕ'_T, where T' is a triangulation refining the subdivision produced by $\widehat{w}$, corresponds to a resultant vertex $\rho_{T'}$ s.t. $\pi(\rho_{T'})$ lies on the same face of Π with ρ_T, hence also lies on $\partial\Pi$.

Now we prove that v is a vertex of Π. Let w be such that the face f of $N(\mathcal{R})$ extremal wrt $\widehat{w}$ contains a vertex ρ_T which projects to relint$(\pi(f))$, where relint$(\cdot)$ denotes relative interior. Then, if Vtx$(\mathcal{A}, w, \pi)$ returns $\pi(\rho_T)$, this is a point on $\partial\Pi$ but not a vertex of Π. We resolve such degeneracies by adding an infinitesimal generic perturbation vector to w, thus obtaining w_p. Since the perturbation is arbitrarily small, $\widehat{w_p}$ shall be normal to a vertex of f, extremal wrt w, but projecting to a vertex of $\pi(f)$. This equivalent to computing a triangulation refining the regular subdivision of $\mathcal{A}$ induced by $\widehat{w}$, whose normal cone's projection is full (m) dimensional. The perturbation can be implemented in Vtx, without affecting any other parts of the algorithm. In practice, our implementation does avoid degenerate cases. □

The *initialization algorithm* computes an inner approximation of Π in both V- and H-representations (denoted Q, Q^H, resp.), and triangulated. First, it calls Vtx$(\mathcal{A}, w, \pi)$ for $w \in W \subset (\mathbb{R}^m)^\times$;

set W is either random or contains, say, vectors in the $2m$ coordinate directions. Then, it updates Q by adding $\mathrm{Vtx}(\mathcal{A}, w, \pi)$ and $\mathrm{Vtx}(\mathcal{A}, -w, \pi)$, where w is normal to hyperplane $H \subset \mathbb{R}^m$ containing Q, as long as either of these points lies outside H. We stop when these points do no longer increase $\dim(Q)$.

Lemma 6. *The initialization algorithm computes $Q \subseteq \Pi$ s.t. $\dim(Q) = \dim(\Pi)$.*

Incremental Alg. 1 computes both V- and H-representations of Π and a triangulation of Π, given an inner approximation Q, Q^H of Π computed at initialization. A hyperplane H is *legal* if it is a supporting hyperplane to a facet of Π, otherwise it is *illegal*. At every step of Alg. 1, we compute $v = \mathrm{Vtx}(\mathcal{A}, w, \pi)$ for a supporting hyperplane H of a facet of Q with normal w. If $v \notin H$, it is a new vertex thus yielding a tighter *inner approximation* of Π by inserting it to Q, i.e. $Q \subset \mathrm{CH}(Q \cup v) \subseteq \Pi$, where $\mathrm{CH}(\cdot)$ denotes convex hull. This happens when the preimage $\pi^{-1}(f) \subset N(\mathcal{R})$ of the facet f of Q defined by H, is not a Minkowski summand of a face of $\Sigma(\mathcal{A})$ having normal $\widehat{w}$. Otherwise, there are two cases: either $v \in H$ and $v \in Q$, thus the algorithm simply decides hyperplane H is legal, or $v \in H$ and $v \notin Q$, in which case the algorithm again decides H is legal but also inserts v to Q.

The algorithm computes Q^H from Q, then iterates over the new hyperplanes to either compute new vertices or decide they are legal, until no increment is possible, which happens when all hyperplanes are legal. Alg. 1 ensures that each normal w to a hyperplane supporting a facet of Q is used only *once*, by storing all used w's in a set W. When a new normal w is created, the algorithm checks if $w \notin W$, then calls $\mathrm{Vtx}(\mathcal{A}, w, \pi)$ and updates $W \leftarrow W \cup w$. If $w \in W$ then the same or a parallel hyperplane has been checked in a previous step of the algorithm. It is straightforward that w can be safely ignored; Lem. 7 formalizes the latter case.

Lemma 7. *Let H' be a hyperplane supporting a facet constructed by Alg. 1, and $H \neq H'$ an illegal hyperplane at a previous step. If H', H are parallel then H' is legal.*

The next lemma formulates the termination criterion of our algorithm.

Lemma 8. *Let vertex v be computed by $Vtx(\mathcal{A}, w, \pi)$, where w is normal to a supporting hyperplane H of Q, then $v \notin H \Leftrightarrow H$ is not a supporting hyperplane of Π.*

Proof. Let $v = \pi(\rho_T)$, where T is a triangulation refining subdivision S in Vtx. It is clear that, since $v \in \partial\Pi$ extremal wrt w, if $v \notin H$ then H cannot be a supporting hyperplane of Π. Conversely, let $v \in H$. By the proof of Lem. 5, every other vertex $\pi(\rho'_T)$ on the face of $N(\mathcal{R})$ is extremal wrt w, hence lies on H, thus H is a supporting hyperplane of Π. □

We now bound the *complexity* of our algorithm. Beneath-and-beyond, given a k-dimensional polytope with l vertices, computes its H-representation and a triangulation in $O(k^5 l t^2)$, where t is the number of full-dimensional faces (cells) [22]. Let $|\Pi|, |\Pi^H|$ be the number of vertices and facets of Π.

Lemma 9. *Alg. 1 computes at most $|\Pi| + |\Pi^H|$ regular triangulations of $\mathcal{A}$.*

Proof. The steps of Alg. 1 increment Q. At every such step, and for each supporting hyperplane H of Q with normal w, we compute one vertex of Π, by Lem. 5. If H is illegal, this vertex is unique because H separates the set of (already computed) vertices of Q from the set of vertices of $\Pi \setminus Q$ which are extremal wrt w, hence, an appropriate translate of H also separates the corresponding sets of vertices of $\Sigma(\mathcal{A})$ (Fig. 3). This vertex is never computed again because it now belongs to Q. The number of regular triangulations of $\mathcal{A}$ yielding vertices is thus bounded by $|\Pi|$.

For a legal hyperplane of Q, we compute one vertex of Π that confirms its legality; the regular triangulation of $\mathcal{A}$ yielding this vertex is accounted for by the legal hyperplane. Since every normal to a hyperplane of Q is used only once in Alg. 1 (by the earlier discussion concerning the set W of all used normals), the statement follows. □

Algorithm 1: ComputeΠ $(A_0, \ldots, A_n, \pi)$

Input : essential $A_0, \ldots, A_n \subset \mathbb{Z}^n$ processed by lem. 3,
projection $\pi : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^m$,
H-, V-repres. Q^H, Q; triang. T_Q of $Q \subseteq \Pi$.
Output: H-, V-repres. Q^H, Q; triang. T_Q of $Q = \Pi$.

$\mathcal{A} \leftarrow \bigcup_0^n (A_i \times e_i)$ `// Cayley trick`
;
$\mathcal{H}_{illegal} \leftarrow \emptyset$;
foreach $H \in Q^H$ **do** $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \cup \{H\}$

while $\mathcal{H}_{illegal} \neq \emptyset$ **do**
  select $H \in \mathcal{H}_{illegal}$ and $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \setminus \{H\}$;
  w is the outer normal vector of H;
  $v \leftarrow \mathrm{Vtx}(\mathcal{A}, w, \pi)$;
  if $v \notin H \cap Q$ **then**
    $Q^H_{temp} \leftarrow \mathrm{conv}(Q \cup \{v\})$;
    foreach $(d-1)$-*face* $f \in T_Q$ *with* $f \subset \partial H$ **do**
      $T_Q \leftarrow T_Q \cup \{\text{faces of } conv(f, v)\}$;
    foreach $H' \in \{Q^H \setminus Q^H_{temp}\}$ **do**
      $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \setminus \{H'\}$`//`$H'$`separates`$\,Q, v$
      ;
    foreach $H' \in \{Q^H_{temp} \setminus Q^H\}$ **do**
      $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \cup \{H'\}$ `// new h/plane`
      ;
    $Q \leftarrow Q \cup \{v\}$;
    $Q^H \leftarrow Q^H_{temp}$;

return Q, Q^H, T_Q;

Let the size of a triangulation be the number of its cells. Let $s_{\mathcal{A}}$ denote the size of the largest triangulation of $\mathcal{A}$ computed by Vtx, and s_{Π} that of Π computed by Alg. 1. In Vtx, the computation of a regular triangulation reduces to a convex hull, computed in $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$; for ρ_T we compute Volume for all cells of T in $O(s_{\mathcal{A}} n^3)$. The overall complexity of Vtx becomes $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$. Alg. 1 calls, in every step, Vtx to find a point on $\partial\Pi$ and insert it to Q, or conclude that a hyperplane is legal. By Lem. 9 it executes Vtx as many as $|\Pi| + |\Pi^H|$ times, in $O((|\Pi| + |\Pi^H|)n^5|\mathcal{A}|s_{\mathcal{A}}^2)$, and computes the H-representation of Π in $O(m^5|\Pi|s_{\Pi}^2)$. Now we have, $|\mathcal{A}| \leq (2n+1)s_{\mathcal{A}}$ and as the input $|\mathcal{A}|, m, n$ grows large we can assume that $|\Pi| \gg |\mathcal{A}|$ and thus s_{Π} dominates $s_{\mathcal{A}}$. Moreover, $s_{\Pi}(m+1) \geq |\Pi^H|$. Now, let $\widetilde{O}(\cdot)$ imply that polylogarithmic factors are ignored.

Theorem 10. *The time complexity of Alg. 1 to compute* $\Pi \subset \mathbb{R}^m$ *is* $O(m^5|\Pi|s_{\Pi}^2 + (|\Pi| + |\Pi^H|)n^5|\mathcal{A}|s_{\mathcal{A}}^2)$, *which becomes* $\widetilde{O}(|\Pi|s_{\Pi}^2)$ *when* $|\Pi| \gg |\mathcal{A}|$.

This implies our algorithm is output sensitive. Its experimental performance confirms this property, see Sect. 5.

We have proven that oracle Vtx (within our algorithm) has two important properties:

1. Its output is a vertex of the target polytope (Lem. 5)
2. When the direction w is normal to an illegal facet, then the vertex computed by the oracle is computed once (Lem. 9)

The algorithm can easily be generalized to incrementally compute any polytope P if the oracle associated with the problem satisfies property (1); if it satisfies also property (2), then the computation can be done in $O(|P| + |P^H|)$ oracle calls, where $|P|$, $|P^H|$ denotes the number of vertices

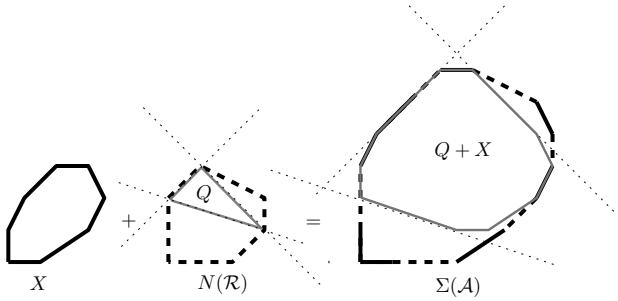

Figure 3: Lem. 9: each illegal hyperplane of Q with normal w, separates the already computed vertices of Π (here equal to $N(\mathcal{R})$) from new ones, extremal wrt w. X is a polytope s.t. $X+N(\mathcal{R}) = \Sigma(\mathcal{A})$.

and number of facets of P, respectively. For example, if the described oracle returns $\pi(\phi_T)$ instead of $\pi(\rho_T)$, it can be used to compute orthogonal projections of secondary polytopes.

The algorithm readily yields an approximate variant: for each supporting hyperplane H, we use its normal w to compute $v = \text{Vtx}(\mathcal{A}, w, \pi)$. Instead of computing a convex hull, now simply take the hyperplane parallel to H through v. The set of these hyperplanes defines a polytope $Q_o \supseteq \Pi$, i.e. an *outer approximation* of Π. Thus, we have an approximation algorithm by stopping Alg. 1 when $\text{vol}(Q)/\text{vol}(Q_o)$ achieves a user-defined threshold. Then, $\text{vol}(Q)/\text{vol}(\Pi)$ is bounded by the same threshold. This algorithm is up to 25 times faster than the deterministic version. Of course, $\text{vol}(Q)$ is available by our incremental convex hull algorithm. However, $\text{vol}(Q_o)$ is the critical step; we plan to examine algorithms that update (exactly or approximately) this volume.

When all hyperplanes of Q are checked, knowledge of legal hyperplanes accelerates subsequent computations of Q^H, although it does not affect its worst-case complexity. Specifically, it allows us to avoid checking legal facets against new vertices.

4 Hashing of Determinants

This section discusses methods to avoid duplication of computations by exploiting the nature of the determinants appearing in the inner loop of our algorithm. Our algorithm computes many regular triangulations, which are typically dominated by the computation of determinants. Similar techniques can be used to improve determinantal predicates in incremental convex hull computations [13].

Consider the $2n \times |\mathcal{A}|$ matrix with the points of $\mathcal{A}$ as columns. Define P as the extension of this matrix by adding lifting values $\widehat{w}$ as the last row. We use the Laplace (or cofactor) expansion along the last row for computing the determinant of the square submatrix formed by any $2n+1$ columns of P; wlog these are the first $2n+1$ columns $a_1, \ldots, a_{2n+1}$. Let $\langle a_1, \ldots, a_{2n+1} \rangle \setminus i$ be the vector $\langle a_1, \ldots, a_{2n+1} \rangle$ without its i-th element; $P_{\langle a_1, \ldots, a_{2n+1} \rangle \setminus i}$ is the $(2n) \times (2n)$ matrix obtained from the $2n$ first elements of the columns whose indices are in $\langle a_1, \ldots, a_{2n+1} \rangle \setminus i$.

Orientation is the sign of the determinant of $P^{hom}_{\langle a_1, \ldots, a_{2n+2} \rangle}$, constructed by columns $a_1, \ldots, a_{2n+2}$ when we add $\vec{1} \in \mathbb{R}^{2n+2}$ as last row. Computing a regular subdivision is a long sequence of such predicates with different a_i's. We expand along the next-to-last row which contains the lifting values and compute the determinants $|P_{\langle a_1, \ldots, a_{2n+2} \rangle \setminus \{i\}}|$ for $i \in \{1, \ldots, 2n+2\}$. Another predicate is Volume, used by Vtx. It equals the determinant of $P^{hom}_{\langle a_1, \ldots, a_{2n+1} \rangle}$, constructed by columns $a_1, \ldots, a_{2n+1}$ when we replace its last row by $\vec{1} \in \mathbb{R}^{2n+1}$.

Our contribution consists in maintaining a hash table with the computed minors, which will be reused at subsequent steps of the algorithm. We store all minors of sizes between 2 and $2n$. For Orientation, they are independent of w and once computed they are stored in the hash table. The main advantage of our scheme is that, for a new w, the only change in P are m (nonzero)

coordinates in the last row, hence computing the new determinants can be done by reusing hashed minors. This also saves time from matrix constructions.

Laplace expansion has complexity $O(n!)$, which decreases as long as minors do not need to be recomputed.

Lemma 11. *Using hashing of determinants, the complexity of the Orientation and Volume predicates is $O(n)$ and $O(1)$, respectively, if all minors have already been computed.*

Many determinant algorithms modify the input matrix; this makes necessary to create a new matrix and introduces a constant overhead on each minor computation. Computing with Laplace expansion, while hashing the minors of smaller size, performs better than state-of-the-art algorithms, in practice. Experiments in Sect. 5 show that our algorithm with hashed determinants outperforms the version without hash. For $m = 3$ and $m = 4$, we experimentally observed that the speedup factor is between 18 and 100; Fig. 4(b) illustrates the second case.

We looked for a hashing function that takes as input a vector of integers and returns an integer that minimizes collisions. We considered many different hash functions, including some variations of the well-known FNV hash [15]. We obtained the best results with the implementation of Boost Hash [20], which shows fewer collisions than the other tested functions.

The drawback is the amount of storage, which is in $O(n!)$. The hash table can be cleared at any moment to limit memory consumption, at the cost of dropping all previously computed minors. Finding a heuristics to clear the hash table according to the number of times each minor was used would decrease the memory consumption, while keeping running times low. In our experiments, we obtained a good tradeoff between efficiency and memory consumption by clearing the entire hash table when it contains 10^6 minors. Last column of Table 1 shows that the memory consumption of our algorithm is related to $|A|$ and $dim(\Pi)$.

It is possible to exploit the structure of the above $(2n) \times (2n)$ minor matrices. Let M be such a matrix, with columns corresponding to points of $A_0, \ldots, A_n$. Let $f : \{1, \ldots, 2n\} \to \{0, \ldots n\}$ return i, given the column index. After column permutations, we split M into 4 $n \times n$ submatrices A, B, D, I, where I is the identity and D is a permutation matrix. This follows from the fact that the bottom half of every column in M has at most one 1, and the last n rows of M contain at least one 1 each, unless $\det M = 0$, which is easily checked. Now, $\det M = \pm \det(B - AD)$, with AD constructed in $O(n)$. Hence, the computation of $(2n) \times (2n)$ minors is asymptotically equal to computing an $n \times n$ determinant. This only decreases the constant within the asymptotic bound. A simple implementation of this idea is not faster than Laplace expansion in the dimensions that we currently focus. However, this idea should be valuable in higher dimensions.

5 Implementation and Experiments

We implemented Alg. 1 in C++ to compute Π; our code can be obtained from `http://respol.sourceforge.net`. All timings are on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, running 64-bit Debian GNU/Linux. We rely on CGAL, using mainly a preliminary version of package `triangulation` under development, for both regular triangulations, as well as for the V- and H-representation of Π.

We first compare 4 state-of-the-art exact convex hull packages, `triangulation` [4] implementing [8], and `beneath-and-beyond (bb)` in `polymake` [17]; double description implemented in `cdd` [16], and `lrs` implementing reverse search [1], to compute Π actually extending the work in [2] for the new class of polytopes Π. The first was shown to be faster in computing Delaunay triangulations in ≤ 6 dimensions [4]. The last 3 are run through `polymake`, where we have ignored the time to load the data. We test all packages in an offline version. We first compute the V-representation of Π using our implementation and then we give this as an input to the convex hull packages that compute the H-representation of Π. Moreover, we test `triangulation` by inserting points in the order that Alg. 1 computes them, while improving the point location of these points since we know by the execution of Alg. 1 one facet to be removed (online version). The experiments show that `triangulation` and `bb` are faster than `lrs`, which outperforms `cdd`.

m	$\lvert\mathcal{A}\rvert$	# of Π vertices	time						`respol` memory
			`respol`	`tr/on`	`tr/off`	`bb`	`cdd`	`lrs`	
3	2490	318	85.03	0.07	0.10	0.07	1.20	0.10	37
4	27	830	15.92	0.71	1.08	0.50	26.85	3.12	46
4	37	2852	97.82	2.85	3.91	2.29	335.23	39.41	64
5	15	510	11.25	2.31	5.57	1.22	47.87	6.65	44
5	18	2584	102.46	13.31	34.25	9.58	2332.63	215.22	88
5	24	35768	4610.31	238.76	577.47	339.05	$>$ 1hr	$>$ 1hr	360
6	15	985	102.62	20.51	61.56	28.22	610.39	146.83	2868
6	19	23066	6556.42	1191.80	2754.30	$>$ 1hr	$>$ 1hr	$>$ 1hr	6693
7	12	249	18.12	7.55	23.95	4.99	6.09	11.95	114
7	17	500	302.61	267.01	614.34	603.12	10495.14	358.79	5258

Table 1: Total time (in seconds) and memory consumption (in megabytes) of our code (`respol`) and time comparison of online version of `triangulation (tr/on)` and offline versions of all convex hull packages for computing the H-representation of Π.

Furthermore, the online version of `triangulation` is 2.5 times faster than its offline counterpart due to faster point location (Table 1).

input	m	3	3	4	4	5	5
	$\lvert\mathcal{A}\rvert$	200	490	20	30	17	20
approxim. algorithm	# of Q vertices	15	11	63	121	$>$ 10hr	$>$ 10hr
	$\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$	0.96	0.95	0.93	0.94	$>$ 10hr	$>$ 10hr
	$\mathrm{vol}(Q_o)/\mathrm{vol}(\Pi)$	1.02	1.03	1.04	1.03	$>$ 10hr	$>$ 10hr
	time	0.15	0.22	0.37	1.42	$>$ 10hr	$>$ 10hr
uniformly random	$\lvert Q\rvert$	34	45	123	207	228	257
	random vectors	606	576	613	646	977	924
	$\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$	0.93	0.99	0.94	0.90	0.90	0.90
	$\mathrm{vol}(Q_o)/\mathrm{vol}(\Pi)$	1.05	1.01	1.02	1.02	1.03	$>$ 10hr
	time	5.61	12.78	1.1	4.73	8.41	16.9
random variant	# of Q vertices	26	21	102	380	341	544
	$\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$	0.92	0.90	0.80	0.92	0.80	0.81
	$\mathrm{vol}(Q_o)/\mathrm{vol}(\Pi)$	1.02	1.04	1.14	1.03	1.08	1.10
	time	0.16	0.31	1.54	23.66	59.87	211.50
exact alg.	# of Π vertices	98	133	416	1296	1674	5093
	time	2.03	5.87	3.72	25.97	51.54	239.96

Table 2: Typical results on experiments computing Q, Q_o^H using the approximation algorithm and the random vectors procedures (the uniform and its variant); we stop the approximation algorithm when $\frac{\mathrm{vol}(Q)}{\mathrm{vol}(Q_o)} > 0.9$; results on random procedures is the average values over 10 independent experiments; "$>$ 10hr" means computation of $\mathrm{vol}(Q_o)$ takes $>$ 10hr on `polymake`.

An advantage of `triangulation` is that it maintains a polytope whose boundary and interior are triangulated. This is useful when the oracle $\mathrm{Vtx}(\mathcal{A}, w, \pi)$ needs to refine the regular subdivision of $\mathcal{A}$ which is obtained by projecting the upper hull of the lifted pointset $\mathcal{A}^{\widehat{w}}$ (Sect. 3). In fact this refinement is attained by a placing triangulation, i.e., by computing the projection of the upper hull of the placing triangulation of $\mathcal{A}^{\widehat{w}}$. This is implemented in two steps:

Step 1. compute the placing triangulation of the last $\lvert\mathcal{A}\rvert - m$ points in the order they appear in $\mathcal{A}^{\widehat{w}}$ (they all have height zero),

Step 2. place the first m points of $\mathcal{A}^{\widehat{w}}$ in T_0 in the order they appear in $\mathcal{A}$.

Step 1 is performed only once at the beginning of the algorithm, whereas Step 2 is performed every

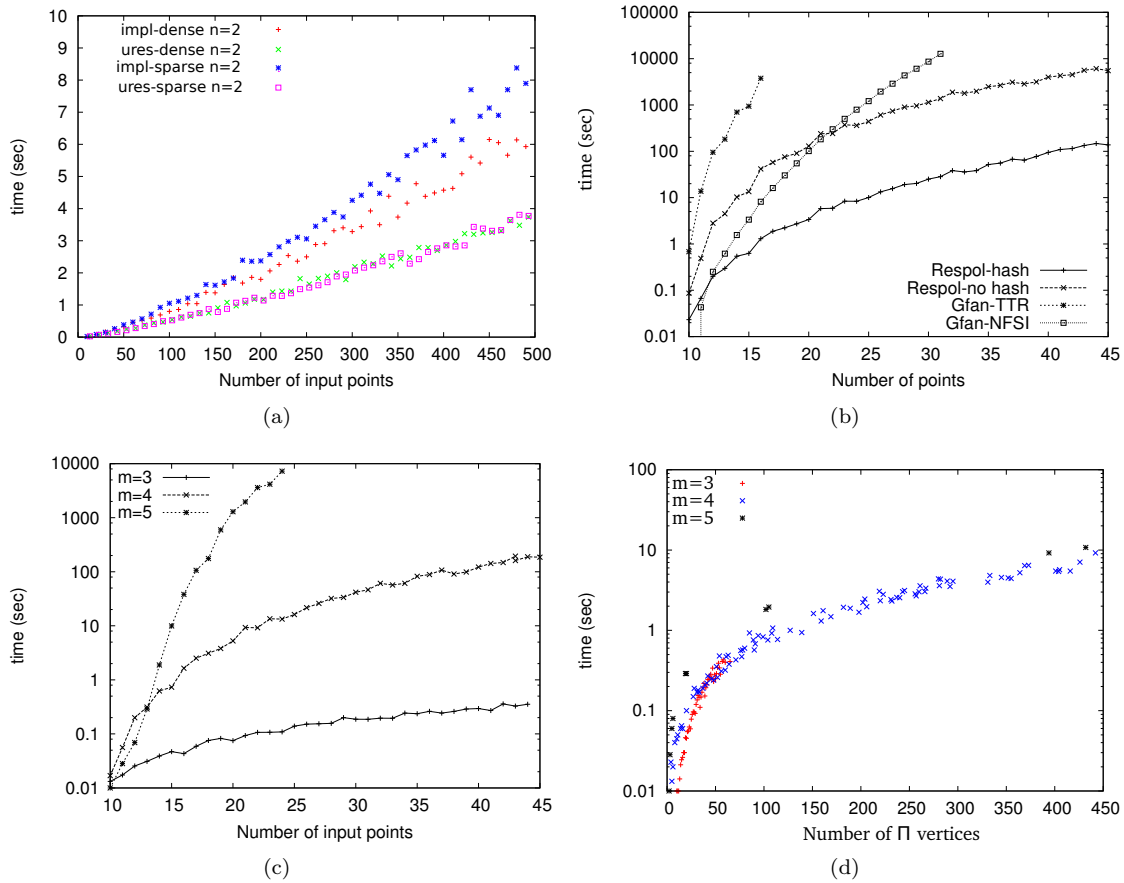

Figure 4: (a) Implicitization and u-resultants for $n = 2, m = 3$ (b) Comparison of `respol` (hashing and not hashing determinants) and `Gfan` (traversing tropical resultants and computing normal fan from stable intersection) for $m = 4$ (c) Performance of Alg 1 for $m = 3, 4, 5$ as a function of input (d) Performance of Alg 1 as a function of its output; y-axis in (b),(c),(d) are in logarithmic scale.

time we check a new w. The order of placing the points in Step 2 only matters if w is not generic; otherwise, w already produces a triangulation of the m points, so any placing order results in this triangulation.

We can formulate this 2-step construction using a single lifting. Let $c > 0$ be a sufficiently large constant, $a_i \in \mathcal{A}$, $q_i \in \mathbb{R}$, $q_i > c\, q_{i+1}$, for $i = 1, \ldots, |\mathcal{A}|$. Define lifting $h : \mathcal{A} \to \mathbb{R}^2$, where $h(a_i) = (w_i, q_i)$, for $i = 1, \ldots, m$, and $h(a_i) = (0, q_i)$, for $i = m+1, \ldots, |\mathcal{A}|$. Then, projecting the upper hull of $\mathcal{A}^h$ to $\mathbb{R}^{2n}$ yields the triangulation of $\mathcal{A}$ obtained by the 2-step construction.

This is the implemented method; although different from the perturbation in the proof of Lem. 5, it is more efficient because of the reuse of triangulation T_0 in Step 1 above. Moreover, our experiments show that it always validates the two conditions in Sect. 3.

Fixing the dimension of the triangulation at compile time results in $< 1\%$ speedup. We also tested a filtered kernel with a similar time speedup. On the other hand, `triangulation` is expected to implement incremental high-dimensional regular triangulations wrt to a lifting, faster than the above method [10]. Moreover, we use a modified version of `triangulation` in order to benefit from our hashing scheme. Therefore, all cells of the triangulated facets of Π have the same normal vector and we use a structure (`STL set`) to maintain the set of unique normal vectors, thus computing only one regular triangulation per triangulated facet of Π.

We perform an *experimental analysis* of our algorithm. We design experiments parameterized on: the total number of input points $|\mathcal{A}|$, the dimension n of pointsets A_i, and the dimension

of projection m. First, we examine our algorithm on random inputs for implicitization and u-resultants, where $m = n + 1$, while varying $|\mathcal{A}|, n$. We fix $\delta \in \mathbb{N}$ and select random points on the δ-simplex to generate dense inputs, and points on the $(\delta/2)$-cube to generate sparse inputs. For *implicitization* the projection coordinates correspond to point $a_{i1} = (0, \ldots, 0) \in A_i$. For $n = 2$ the problem corresponds to implicitizing surfaces: when $|\mathcal{A}| < 60$, we compute the polytopes in $<$ 1sec (Fig. 4(a)). When computing the *u-resultant* polytope, the projection coordinates correspond to $A_0 = \{(1, \ldots, 0), \ldots, (0, \ldots, 1)\}$. For $n = 2$, when $|\mathcal{A}| < 500$, we compute the polytopes in $<$ 1sec (Fig. 4(a)).

By using the *hashing determinants* scheme we gain a 18$\times$ speedup when $n = 2, m = 3$. For $m = 4$ we gain a larger speedup; we computed in $<$ 2min an instance where $|\mathcal{A}| = 37$ and would take $>$ 1hr to compute otherwise. Thus, when the dimension and $|\mathcal{A}|$ becomes larger, this method allows our algorithm to compute instances of the problem that would be intractable otherwise, as shown for $n = 3, m = 4$ (Fig. 4(b)).

We confirm experimentally the *output-sensitivity* of our algorithm. First, our algorithm always computes vertices of Π either to extend Π or to legalize a facet. We experimentally show that our algorithm has a subexponential behaviour wrt to both input and output (Fig. 4(c), 4(d)) and its output is subexponential wrt the input.

We explore the *limits* of our implementation. By bounding runtime to $<$ 2hr, we compute instances of 5-, 6-, 7-dimensional Π with 35K, 23K, 500 vertices, resp. (Table 1).

We also compare with the implementation of [21], which is based on `Gfan` library. They develop two algorithms to compute projections of $N(\mathcal{R})$. Assuming $\mathcal{R}$ defines a hypersurface, their methods computes a union of (possibly overlapping) cones, along with their multiplicities [21, Thm.2.9]. From this intermediate result they construct the normal cones to the resultant vertices. We compare with the best timings of `Gfan` methods using the examples and timings of [21]: our method is faster for $m < 7, |\mathcal{A}| < 16$ and competitive (up to 2 times slower) when $m = 5, |\mathcal{A}| = 20$ or $m = |\mathcal{A}| = 15$. We run extensive experiments for $n = 3$, considering implicitization, where $m = 4$ and our method, with and without using hashing, is much faster than any of the two algorithms based on `Gfan` (Fig. 4(b)). However, for $n = 4, m = 5$ the beta version of `Gfan` we run was not stable and always crashes when $|\mathcal{A}| > 13$.

We analyze the computation of inner and outer *approximations* Q and Q_o^H. We test the variant of Sect. 3 by stopping it when $\frac{\text{vol}(Q)}{\text{vol}(\Pi)} > 0.9$. In the experiments, the number of Q vertices is $< 15\%$ of the Π vertices, thus the speedup is up to 25 times faster than the exact algorithm and $\frac{\text{vol}(Q_o^H)}{\text{vol}(\Pi)} < 1.04$ (Table 2). The bottleneck of this computation is the computation of $\text{vol}(Q_o^H)$, which is in H-representation. We use `lrs` through `polymake` in every step to compute $\text{vol}(Q_o^H)$ because we are lacking of an implementation that given a polytope P in H-representation, its volume and a halfspace H, computes the volume of the intersection of P and H. Note that we don't count this computation time in total time.

Next, we study procedures that compute only the V-representation of Q. For this, we count how many *random vectors* uniformly distributed on the m-sphere are needed to obtain $\frac{\text{vol}(Q)}{vol(\Pi)} > 0.9$. This procedure runs up to 10 times faster than the exact algorithm. Its drawback is that it does not provide guarantees for $\frac{\text{vol}(Q)}{\text{vol}(\Pi)}$ because we do not know Π in advance. To overcome this, we test a variant procedure which starts with a sequence of random vectors and produces vectors that are affine combinations of the existing ones. It stops when the produced vectors do not give a new vertex for Q. Even if it computes Q s.t. $\frac{\text{vol}(Q)}{\text{vol}(\Pi)} > 0.9$ up to 10 times faster than the exact algorithm, on average it computes Q s.t. $\frac{\text{vol}(Q)}{\text{vol}(\Pi)} > 0.8$ in similar runtimes (Table 2).

6 Future work

As shown, `polymake`'s convex hull algorithm is competitive, thus one may use it for implementing our algorithm. On the other hand, `triangulation` is expected to include [10] fast enumeration

of all regular triangulations for a given (non generic) lifting, in which case Π may be extended by more than one (coplanar) vertices.

The resultant edges are associated to flips on mixed cells [29]. We studied them algorithmically [12] and may revisit them to generate all neighbors of a vertex of $N(\mathcal{R})$.

Any incremental convex hull algorithm has a worst-case super-polynomial *total time complexity* [5] in the number of input points and output facets. The question is whether there is a polynomial total time incremental algorithm for Π. For this we study the structure of $N(\mathcal{R})$ through experiments; it appears to be an exciting family of polytopes.

7 Acknowledgments

We thank O. Devillers and S. Hornus for discussions on `triangulation`, and A. Jensen and J. Yu for discussions and for sending us a beta version of their code.

References

[1] D. Avis. lrs: A revised implementation of the reverse search vertex enumeration algorithm. In *Polytopes: Combinatorics & Computation*, volume 29 of *Oberwolfach Seminars*, pages 177–198. 2000.

[2] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comput. Geom.: Theory & Appl.*, 7:265–301, 1997.

[3] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry.* Springer-Verlag, Berlin, 2003.

[4] J.-D. Boissonnat, O. Devillers, and S. Hornus. Incremental construction of the Delaunay triangulation and the Delaunay graph in medium dimension. In *Proc. SoCG*, pages 208–216, 2009.

[5] D. Bremner. Incremental convex hull algorithms are not output sensitive. In *Proc. 7th Intern. Symp. Algorithms and Comput.*, pages 26–35, London, UK, 1996. Springer.

[6] H. Brönnimann, I.Z. Emiris, V. Pan, and S. Pion. Sign determination in Residue Number Systems. *Theor. Comp. Science, Spec. Issue on Real Numbers & Computers*, 210(1):173–197, 1999.

[7] CGAL: Computational geometry algorithms library. http://www.cgal.org.

[8] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom.: Theory & Appl.*, 3:185–121, 1993.

[9] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry.* Number 185 in GTM. Springer, New York, 2nd edition, 2005.

[10] O. Devillers, 2011. Personal communication.

[11] J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Linbox: A generic library for exact linear algebra. In *Proc. Intern. Congress Math. Software*, pages 40–50, Beijing, 2002.

[12] I.Z. Emiris, V. Fisikopoulos, and C. Konaxis. Regular triangulations and resultant polytopes. In *Proc. Europ. Works. Comput. Geometry*, Dortmund, 2010.

[13] I.Z. Emiris, V. Fisikopoulos, and L. Peñaranda. Optimizing the computation of sequences of determinantal predicates. In *Proc. Europ. Workshop Computat. Geometry*, Assisi, Italy, 2012.

[14] I.Z. Emiris, T. Kalinka, and C. Konaxis. Implicitization using predicted support. In *Proc. Intern. Works. Symbolic-Numeric Computation*, 2011.

[15] G. Fowler, L.C. Noll, and P. Vo. Fowler-Noll-Vo hash. www.isthe.com/chongo/tech/comp/fnv, 1991.

[16] K. Fukuda. cdd and cdd+ Home Page. ETH Zürich. www.ifor.math.ethz.ch/∼fukuda/cdd_home/, 2008.

[17] E. Gawrilow and M. Joswig. Polymake: an approach to modular software design in computational geometry. In *Proc. Annual ACM Symp. Computational Geometry*, pages 222–231. ACM Press, 2001.

[18] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[19] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[20] D. James. Boost functional library. www.boost.org/ libs/functional/hash, 2008.

[21] A. Jensen and J. Yu. Computing tropical resultants. *arXiv:math.AG/1109.2368v1*, 2011.

[22] M. Joswig. Beneath-and-beyond revisited. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, Mathematics and Visualization. Springer, Berlin, 2003.

[23] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 13:91–130, 2005.

[24] M.M. Kapranov. Characterization of A-discriminantal hypersurfaces in terms of logarithmic Gauss map. *Math. Annalen*, 290:277–285, 1991.

[25] J.A. De Loera, J. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer, 2010.

[26] T. Michiels and R. Cools. Decomposing the secondary Cayley polytope. *Discr. Comput. Geometry*, 23:367–380, 2000.

[27] T. Michiels and J. Verschelde. Enumerating regular mixed-cell configurations. *Discr. Comput. Geometry*, 21(4):569–579, 1999.

[28] J. Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. In *Proc. Intern. Congress Math. Software*, pages 330–340, 2002.

[29] B. Sturmfels. On the Newton polytope of the resultant. *J. Algebraic Combin.*, 3:207–236, 1994.

[30] B. Sturmfels and J. Yu. Tropical implicitization and mixed fiber polytopes. In *Software for Algebraic Geometry*, volume 148 of *IMA Volumes in Math. & its Applic.*, pages 111–131. Springer, New York, 2008.

[31] G.M. Ziegler. *Lectures on Polytopes*. Springer, 1995.