Project number IST-25582

**CGL**

Computational Geometric Learning

# An oracle-based, output-sensitive algorithm for projections of resultant polytopes

**STREP**

**Information Society Technologies**

| | |
|---|---|
| Period covered: | November 1, 2011 – October 31, 2012 |
| Date of preparation: | October 15, 2012 |
| Date of revision: | October 15, 2012 |
| Start date of project: | November 1, 2010 |
| Duration: | 3 years |
| Project coordinator name: | Joachim Giesen (FSU) |
| Project coordinator organisation: | Friedrich-Schiller-Universität Jena<br>Jena, Germany |

# An oracle-based, output-sensitive algorithm for projections of resultant polytopes

Ioannis Z. Emiris[*] Vissarion Fisikopoulos[*] Christos Konaxis[**]
Luis Peñaranda[§]

October 15, 2012

**Abstract**

We compute the Newton polytope of the resultant, known as resultant polytope, or its orthogonal projection along a given direction. The resultant is fundamental in algebraic elimination, optimization, and geometric modeling. The algorithm exactly computes V- and H-representations of the polytope using an oracle working in $\mathbb{R}^n$ and producing resultant vertices in a given direction, thus avoiding walking on the polytope whose dimension is $\alpha - 2n - 1$, where the input consists of $\alpha$ points in $\mathbb{Z}^{2n}, \alpha > 2n + 2$ but typically $\alpha \gg 2n$. Our approach is output-sensitive as it makes one oracle call per vertex and facet. It extends to any polytope whose oracle-based definition is advantageous, including the secondary and discriminant polytopes.

Our publicly available implementation uses the experimental CGAL package `triangulation`. It computes 5-, 6- and 7-dimensional polytopes with 35K, 23K and 500 vertices, respectively, within 2hr, and the Newton polytope of most surface equations encountered in geometric modeling in < 1sec, whereas the corresponding secondary polytopes are intractable. It is faster than tropical geometry software up to dimension 5 or 6. One variant computes inner and outer approximations with, respectively, 90% and 105% of the true volume, up to 25 times faster. Our approach can yield approximate methods by exploiting oracles on the polar polytope.

**Keywords:** General Dimension, Convex Hull, Secondary Polytope, Resultant, CGAL Implementation, Oracle

## 1 Introduction

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

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

where $e_0, \dots, e_n$ form an affine basis of $\mathbb{R}^n$: $e_0$ is the zero vector, $e_i = (0, \dots, 0, 1, 0, \dots, 0), i = 1, \dots, n$. Clearly, $|\mathcal{A}| = |A_0| + \cdots + |A_n|$, where $|\cdot|$ denotes cardinality. By Cayley's trick (Proposition 2) the regular tight mixed subdivisions of the Minkowski sum $A_0 + \cdots + A_n$ are

[*]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece. {emiris,vissarion}@di.uoa.gr

[**]Current affiliation: Archimedes Center for Modeling, Analysis & Computation (ACMAC), University of Crete, Heraklio, Greece, ckonaxis@acmac.uoc.gr

[§]Current affiliation: IMPA – Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, Brazil, luisp@impa.br

$f(x_1,x_2) = 8x_2 + x_1x_2 - 24x_2^2 - 16x_1^2 + 220x_1^2x_2 - 34x_1x_2^2 - 84x_1^3x_2 + 6x_1^2x_2^2 - 8x_1x_2^3 + 8x_1^3x_2^2 + 8x_1^3 + 18x_2^3$

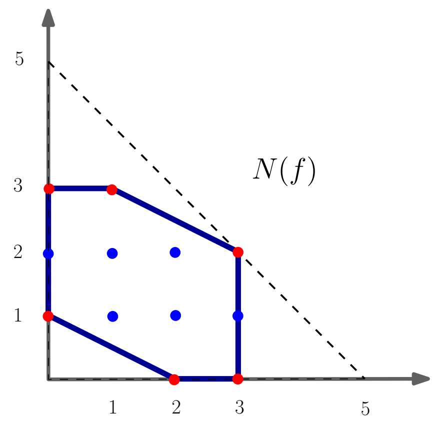


Figure 1: The Newton polytope of a polynomial of degree 5 in two variables. Every monomial corresponds to an integral point on the plane. The dashed triangle is the corresponding polytope of the dense polynomial of degree 5.

in bijection with the regular triangulations of $\mathcal{A}$, which are in bijection with the vertices of the *secondary polytope* $\Sigma(\mathcal{A})$ (see Section 2).

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 precise information (see Figures 1 and 3). Given $n+1$ polynomials in $n$ variables, with fixed supports $A_i$ and symbolic coefficients, their *sparse (or toric) resultant* $\mathcal{R}$ is a polynomial in these coefficients which vanishes exactly when the polynomials have a common root (Definition 1). The resultant is the most fundamental tool in elimination theory, it is instrumental in system solving and optimization, and is crucial in geometric modeling, most notably for changing the representation of parametric hypersurfaces to implicit.

The Newton polytope of the resultant $N(\mathcal{R})$, or *resultant polytope*, is the object of our study; it is of dimension $|\mathcal{A}| - 2n - 1$ (Proposition 4). We further consider the case when some of the input coefficients are not symbolic, hence we seek an orthogonal projection of the resultant polytope. The lattice points in $N(\mathcal{R})$ yield a superset of the support of $\mathcal{R}$; this reduces implicitization [EKKB13, SY08] and computation of $\mathcal{R}$ to sparse interpolation (Section 2). The number $|\mathcal{A}|$ of coefficients of the $n+1$ polynomials is $> 2n+2$ for the resultant problem to be nontrivial and typically $|\mathcal{A}| \gg n$; in general $|\mathcal{A}| = O(n^d d^n)$, 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 $\Sigma(\mathcal{A})$, which is also of dimension $|\mathcal{A}| - 2n - 1$. We consider an equivalence relation defined on the $\Sigma(\mathcal{A})$ 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 typically has much more vertices than $N(\mathcal{R})$. This is known in the literature as an *optimization* oracle since it optimizes inner product with a given vector over the (unknown) polytope.

**Example 1.** [The bicubic surface] *A standard benchmark in geometric modeling is the implicitization of the bicubic surface, with* $n = 2$, *defined by* 3 *polynomials in two parameters. The input polynomials have supports* $A_i \subset \mathbb{Z}^2$, $i = 0,1,2$, *with cardinalities* $7, 6, 14$, *respectively; the total degrees are* $3, 3, 6$, *respectively. The Cayley set* $\mathcal{A} \subset \mathbb{Z}^4$, *constructed as in Equation 1, has* $7 + 6 + 14 = 27$ *points. It is depicted in the following matrix, with coordinates as columns, where the supports from different polynomials and the Cayley coordinates are distinguished. By*

*Proposition 4 it follows that $N(\mathcal{R})$ has dimension $|\mathcal{A}| - 4 - 1 = 22$; it lies in $\mathbb{R}^{27}$.*

$$\left[\begin{array}{ccccccc|cccccc|cccccccccccccc} 0&0&1&0&2&0&3&0&0&1&2&0&3&0&0&1&0&1&2&1&2&1&2&3&2&3&3\\ 0&1&0&2&0&3&0&0&1&0&0&3&0&0&1&0&2&1&0&2&1&3&2&1&3&2&3\\ \hline 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\ 0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}0\\0\end{array}}\right\}\textit{support}\\ \left.\vphantom{\begin{array}{c}0\\0\end{array}}\right\}\textit{Cayley}\end{array}$$

*Implicitization requires eliminating the two parameters to obtain a constraint equation over the symbolic coefficients of the polynomials. Most of the coefficients are specialized except for* 3 *variables, hence the sought for implicit equation of the surface is trivariate and the projection of $N(\mathcal{R})$ lies in $\mathbb{R}^3$.*

*TOPCOM [Ram02] needs more than a day and* 9*GB of RAM to compute* 1,806,467 *regular triangulations of $\mathcal{A}$, corresponding to* 29 *of the vertices of $N(\mathcal{R})$, and crashes before computing the entire $N(\mathcal{R})$. Our algorithm yields the projected vertices $\{(0,0,1),(0,1,0),(1,0,0),(0,0,9),(0,18,0),(18,0,0)\}$ of the* 3*-dimensional projection of $N(\mathcal{R})$, which is the Newton polytope of the implicit equation, in* 30*msec. Given this polytope, the implicit equation of the bicubic surface is interpolated in 42 seconds [EKKB12]. It is a polynomial of degree* 18 *containing* 715 *terms which corresponds exactly to the lattice points contained in the predicted polytope.*

Our main contribution is twofold. First, we design an oracle-based 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 primarily required by the algorithm and may also be relevant 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; in this sense, the algorithms is output sensitive. The overall cost is asymptotically dominated by computing as many regular triangulations of $\mathcal{A}$ (Theorem 11). 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 $\Sigma(\mathcal{A})$, the Newton polytope of the discriminant and, more generally, any polytope that can be efficiently described by a vertex oracle or its orthogonal projection. In particular, it suffices to replace our oracle by the oracle in [Rin12] to obtain a method for computing the discriminant polytope [DEF12].

Second, we describe an efficient, publicly available implementation based on CGAL [CGA] and its 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 [JY11]. Our code, in the critical step of computing the convex hull of the resultant polytope, uses `triangulation`. On our instances, the latter, 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 and matrix structure 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. A variant of our 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. This may lead to an approximation algorithm.

**Previous work.** Sparse (or toric) elimination theory was introduced in [GKZ94]. 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 Section 6 of [Stu94] is proved that $N(\mathcal{R})$ is 1-dimensional if and only if $|A_i| = 2$, 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 (Figure 2(b)). Following [Stu94, 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 are investigating with our code [DEF12]. The maximal (in terms of number of vertices) such polytope we have computed has f-vector $(22, 66, 66, 22)$ (Figure 2(c)). Furthermore, Table 2 presents some typical f-vectors of $4, 5, 6$ dimensional projections of resultant polytopes.

A direct approach for computing the vertices of $N(\mathcal{R})$ might consider all vertices of $\Sigma(\mathcal{A})$ since the vertices of the former are equivalence classes over the vertices of the latter, studied algorithmically in [EFK10]. The computation of secondary polytopes has been efficiently implemented in TOPCOM [Ram02], which has been the reference software for computing regular or all triangulations. It builds a search tree with flips as edges over the vertices of $\Sigma(A)$ but is limited by space usage. To address this, reverse search was proposed [IMTI02], but the implementation cannot compete with TOPCOM. The approach based on computing $\Sigma(\mathcal{A})$ is not efficient for computing $N(\mathcal{R})$. For instance, in implicitizing parametric surfaces with up to 100 terms, which includes all common instances in geometric modeling, we compute the Newton polytope of the equations in less than 1sec, whereas $\Sigma(\mathcal{A})$ is intractable (see e.g. Example 1).

In [MC00] they describe all Minkowski summands of $\Sigma(A)$. In [MV99] is defined an equivalence class over $\Sigma(A)$ vertices having the same mixed cells. The classes map in a many-to-one fashion to resultant vertices; our algorithm exploits a stronger equivalence relationship.

Tropical geometry is a polyhedral analogue of algebraic geometry and can be viewed as generalizing sparse elimination theory. It gives alternative ways of recovering resultant polytopes [JY11] and Newton polytopes of implicit equations [SY08]. See Section 4 for comparisons with our software. It also provides vertex oracles for the Newton polytope of the discriminant polynomial [Rin12].

As is typical in computational geometry, the practical bottleneck is in computing determinantal predicates. This is efficiently addressed in [EFKP12b, FP12] and summed up in the technical report [EFP12].

**Milestone on Representation.** There are two standard representations of a $d$-dimensional polytope, namely as the convex hull of a pointset (V-representation) or the intersection of halfspaces (H-representation). The transformation from one to another is the convex hull problem (V-rep to H-rep) or, dually, the vertex enumeration problem (H-rep to V-rep); they are both known to be hard, also in practice [ABS97a]. A related representation is the triangulation of the polytope, namely a simplicial complex that offers a partition of the polytope to simplices. Storing the entire face lattice of the triangulation is very space-consuming; storing only the faces of dimension 0,1,$d-2, d-1, d$ is adopted by the experimental package `triangulation` of CGAL. Another important representation is the edge-skeleton, which we seriously considered for resultant polytopes since neither the V- nor the H-representations are given.

However, we decided to use a representation based on oracles. We constructed an efficient optimization oracle, which outputs the extremal vertex of the polytope in a given direction. These kind of oracles are well studied in combinatorial optimization [GLS88], including oracles applied to the polar dual. Our oracle provides a *compact* representation for the resultant, but also secondary and discriminant, polytopes. The method should also lead to a polynomial-time volume approximation by exploiting duality. Our current algorithm computes the V- and H-representations of the resultant polytope as well as its triangulation, but current work focuses on investigating whether we can avoid to compute all 3 representations, or the edge skeleton be computed faster.

The material presented in this report has been submitted as [EFKP12a]. This report extends technical report [EFK11] and presents an improved algorithm with better complexity analysis as well as the description of an approximation algorithm, more experimental results, and a more comprehensive comparison to existing work and presentation of the theory of resultants.

The roadmap of this report is as follows: Section 2 describes the combinatorics of resultants, and the following section presents our algorithm. Section 4 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 [LRS10, Zie95] to describe resultants [GKZ94, CLO05]. We shall denote by $\mathrm{vol}(\cdot) \in \mathbb{N}$ the normalized Euclidean volume, $(\mathbb{R}^m)^\times$ the linear $m$-dimensional functionals, $\mathrm{Aff}(\cdot)$ the affine hull, and $\mathrm{CH}(\cdot)$ the convex hull.

Let $\mathcal{A} \subset \mathbb{Z}^d$ 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 $\sigma$ of $T$ having $a$ as a vertex; $\Sigma(\mathcal{A})$ is the convex hull of $\phi_T$ 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.** [[GKZ94]] *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 a regular triangulation $T$ (resp. a regular polyhedral subdivision of $\mathcal{A}$) if and only if it lies in the normal cone of vertex $\phi_T$ (resp. of the corresponding face) of $\Sigma(\mathcal{A})$. The dimension of $\Sigma(\mathcal{A})$ is $|\mathcal{A}| - d - 1$.*

Let $A_0, \dots, A_n$ be subsets of $\mathbb{Z}^n$, $P_0, \dots, P_n \subset \mathbb{R}^n$ 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 Minkowski 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 $v_i \in P_i$. In the sequel we shall call a Minkowski cell, simply cell.

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 $w := (w_0 \dots, w_n)$ 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 $A_0, \dots, A_n$ and the affine basis $\{e_0, \dots, e_n\}$ of $\mathbb{R}^n$, we define the Cayley pointset $\mathcal{A} \subset \mathbb{Z}^{2n}$ as in equation (1).

**Proposition 2.** [Cayley trick, [GKZ94]] *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 $A_0, \dots, A_n \subset \mathbb{Z}^n$ 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 greater than or equal to $j$. It is straightforward to check this property algorithmically and, if it does not hold, to find an essential subset. In the sequel, the input $A_0, \dots, A_n \subset \mathbb{Z}^n$ is supposed to be essential.

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

The resultant offers a solvability condition from which $x$ has been eliminated, hence is also known as eliminant. For $n = 1$, it 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 $F(x_1, \dots, x_n)$ is given by the resultant of $F, \partial F/\partial x_1, \dots, \partial F/\partial x_n$.

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 it is of smaller dimension (Proposition 4). The monomials corresponding to vertices of $N(\mathcal{R})$ are the extreme resultant monomials.

**Proposition 3.** [[GKZ94, Stu94]] *For a sufficiently generic lifting function $w \in (\mathbb{R}^{|\mathcal{A}|})^\times$, the $w$-extreme monomial of $\mathcal{R}$, whose exponent vector maximizes inner product with $w$, equals*

$$\pm \prod_{i=0}^{n} \prod_{\sigma} c_{i,v_i}^{\mathrm{vol}(\sigma)}, \tag{3}$$

*where $\sigma$ ranges over all $v_i$-mixed cells of the regular tight mixed subdivision $S$ of $P$ induced by $w$, and $c_{i,v_i}$ is the coefficient of monomial $x^{v_i}$ in $f_i$.*

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. For simplicity we shall denote by $\sigma$, both a cell of $S$ and its corresponding simplex in $T$. Then,

$$\rho_T(a) = \sum_{\substack{a-\text{mixed} \\ \sigma \in T : a \in \sigma}} \mathrm{vol}(\sigma) \ \in \mathbb{N}, \qquad a \in \mathcal{A}, \tag{4}$$

where simplex $\sigma$ is $a$-mixed if and only if 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 simplices $\sigma \in T$, and each of these volumes is equal to the *mixed volume*[CLO05] of a set of *lattice* polytopes, the Minkowski summands of the corresponding $\sigma \in S$. In particular, assuming that $\sigma \in S$ is $i$-mixed, it can be written as $\sigma = \sigma_0 + \cdots + \sigma_n$, $\sigma_j \subseteq A_j$, $j = 0, \dots, n$, and $\mathrm{vol}(\sigma) = MV(\sigma_0, \dots, \sigma_{i-1}, \sigma_{i+1}, \dots, \sigma_n)$, where $MV$ denotes the mixed volume function which is integer valued for lattice polytopes [CLO05]. Now, $N(\mathcal{R})$ is the convex hull of all $\rho_T$ vectors [GKZ94, Stu94].

Proposition 3 establishes a many-to-one surjection from regular triangulations of $\mathcal{A}$ to regular tight mixed subdivisions of $P$, or, equivalently, 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 $N(\mathcal{R})$. The above discussion is illustrated in Figure 2(a),(b).

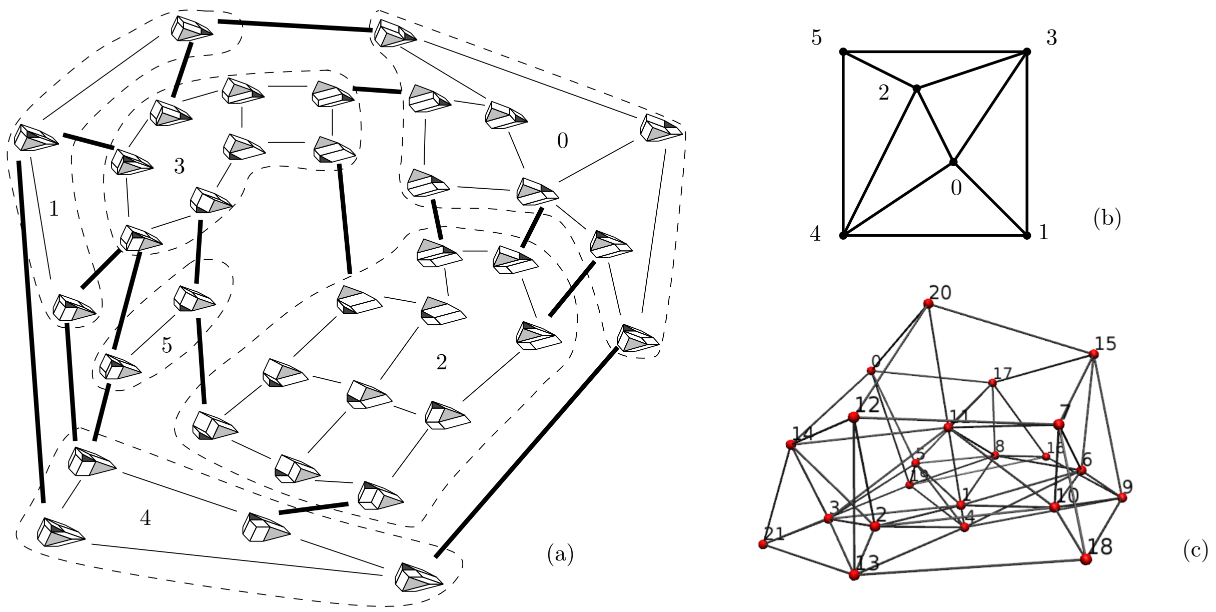


Figure 2: (a) The secondary polytope $\Sigma(\mathcal{A})$ of two triangles (dark, light grey) and one segment $A_0 = \{(0,0),(1,2),(4,1)\}$, $A_1 = \{(0,1),(1,0)\}$, $A_2 = \{(0,0),(0,1),(2,0)\}$, where $\mathcal{A}$ is defined as in Equation 1; vertices correspond to mixed subdivisions of the Minkowski 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 $\Sigma(\mathcal{A})$. Bold edges of $\Sigma(\mathcal{A})$, 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`.

**Proposition 4.** [[GKZ94]] *$N(\mathcal{R})$ is a Minkowski summand of $\Sigma(\mathcal{A})$, and both $\Sigma(\mathcal{A})$ and $N(\mathcal{R})$ have 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 $i = n+1, \ldots, 2n+1$, see Prop. 7.1.11 of [GKZ94]. 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 compute some orthogonal projection of $N(\mathcal{R})$, denoted $\mathit{\Pi}$, in $\mathbb{R}^m$:

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

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 $m = |\mathcal{A}|$, $\pi$ 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 5.** [[JY11] Lemma 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.*

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

**Example 2.** *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 in [Kap91], 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 [BPR03]. Given $f_1, \dots, f_n \in \mathbb{C}[x_1, \dots, x_n]$, 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 that the roots of $f_1 = \cdots = f_n = 0$ are isolated, denoted $r_i = (r_{i1}, \dots, r_{in})$. 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) $N(\mathcal{R}_u)$.

**Example 3.** *Let $f_1 = x_1^2 + x_2^2 - 4$, $f_2 = x_1 - x_2 + 2$, and $f_0 = u_0 + u_1x_1 + u_2x_2$. Our algorithm computes a polygon with vertices $\{(2, 0, 0), (0, 2, 0), (0, 0, 2)\}$, which contains $N(\mathcal{R}_u) = CH(\{(2, 0, 0), (1, 1, 0), (1, 0, 1), (0, 1, 1)\})$. The coefficient specialization is not generic, hence $N(\mathcal{R}_u)$ is strictly contained in the computed polygon. Proceeding as in Example 2, $\mathcal{R}_u = 2u_0^2 + 4u_0u_1 - 4u_0u_2 - 8u_1u_2$, which factors as $2(u_0 + 2u_1)(u_0 - 2u_2)$.*

The last application comes from geometric modeling, where $y_i = f_i(x)$, $i = 0, \dots, n$, $x = (x_1, \dots, x_n) \in \Omega \subset \mathbb{R}^n$, defines a parametric hypersurface. Many applications require the equivalent implicit representation $F(y_1, \dots, y_n) = 0$. 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 [EKKB12, EKK11]. In particular, we compute the polytope of surface equations within 1sec, assuming $\leq 100$ terms in parametric polynomials, which includes all common instances in geometric modeling.

**Example 4.** *Let us see how the above computation can serve in implicitization. Consider the surface given by the polynomial parameterization*

$$(y_1, y_2, y_3) = (x_1x_2, x_1x_2^2, x_1^2).$$

*For polynomials $f_0 := c_{00} - c_{01}x_1x_2$, $f_1 := c_{10} - c_{11}x_1x_2^2$, $f_2 := c_{20} - c_{21}x_1^2$ with supports $A_0 = \{(0, 0), (1, 2)\}, A_1 = \{(0, 0), (1, 2)\}$ and $A_2 = \{(0, 0), (2, 0)\}$. The resultant polytope is a segment in $\mathbb{R}^6$ with endpoints $(4, 0, 0, 2, 0, 1)$, $(0, 4, 2, 0, 1, 0)$ and, actually, $\mathcal{R} = -c_{00}^4c_{11}^2c_{21} + c_{01}^4c_{10}^2c_{20}$. The supports and the two mixed subdivisions corresponding to the vertices of $N(\mathcal{R})$ are illustrated in Figure 3. Specializing the symbolic coefficients of the polynomials as:*

$$(c_{00}, c_{01}, c_{10}, c_{11}, c_{20}, c_{21}) \mapsto (y_1, -1, y_2, -1, y_3, -1)$$

*yields the vertices of the implicit polytope: $(4, 0, 0), (0, 2, 1)$, which our algorithm can compute directly. The implicit equation of the surface turns out to be $-y_1^4 + y_2^2y_3$.*

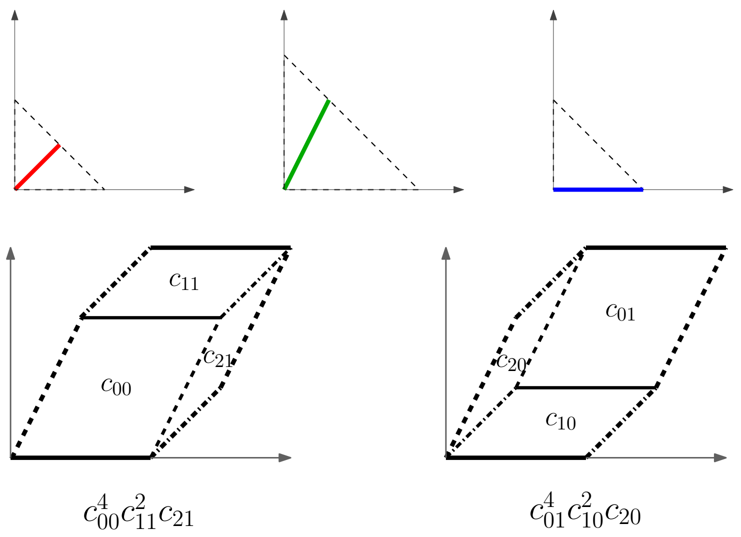


Figure 3: The supports $A_0, A_1, A_2$ of Example 4, their Newton polytopes (segments) and the two mixed subdivisions of their Minkowski sum.

## 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 $\Pi$ of $N(\mathcal{R})$ which results to algorithms for computing $\Pi$ avoiding 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 $\Pi = \pi(N(\mathcal{R}))$, 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$(\cdot)$, is regular and corresponds to some secondary vertex $\phi_T$ which maximizes the inner product with $\widehat{w}$.

**Lemma 6.** *All points computed by* VTX$(\cdot)$ *are vertices of* $\Pi$*.*

*Proof.* Let $v = \pi(\rho_T) = \mathrm{VTX}(\mathcal{A}, w, \pi)$. We first prove that $v$ lies on $\partial\Pi$. Point $\rho_T$ of $N(\mathcal{R})$ is a Minkowski summand of vertex $\phi_T$ of $\Sigma(\mathcal{A})$ extremal with respect to $\widehat{w}$, hence $\rho_T$ is extremal with respect to $\widehat{w}$. Since $\widehat{w}$ is perpendicular to projection $\pi$, $\rho_T$ projects to a point in $\partial\Pi$. The same argument implies that every vertex $\phi'_T$, where $T'$ is a triangulation refining the subdivision produced by $\widehat{w}$, corresponds to a resultant vertex $\rho_{T'}$ such that $\pi(\rho_{T'})$ lies on the same face of $\Pi$ with $\rho_T$, hence also lies on $\partial\Pi$.

Now we prove that $v$ is a vertex of $\Pi$ and not lying inside a face. Let $w$ be such that the face $f$ of $N(\mathcal{R})$ extremal with respect to $\widehat{w}$ contains a vertex $\rho_T$ 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 $\Pi$. 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 with respect to $w$, but projecting to a vertex of $\pi(f)$. This is equivalent to computing a triangulation refining the regular subdivision of $\mathcal{A}$ induced by $\widehat{w}$, whose normal cone's projection is of full dimension, i.e. $m$-dimensional. The perturbation can be implemented in $\mathrm{VTX}(\cdot)$, without affecting any other parts of the algorithm, either by case analysis or by a method of symbolic perturbation. In practice, our implementation does avoid degenerate cases. $\square$

The *initialization algorithm* computes an inner approximation of $\Pi$ in both V- and H-representations (denoted $Q$, $Q^H$, respectively), and triangulated. First, it calls $\mathrm{VTX}(\mathcal{A}, w, \pi)$ for $w \in W \subset (\mathbb{R}^m)^\times$; the 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 $H$. Since every new vertex lies outside the affine hull of the current polytope $Q$, all polytopes produced are simplices. We stop when these points do no longer increase $\dim(Q)$.

**Lemma 7.** *The initialization algorithm computes $Q \subseteq \Pi$ such that $\dim(Q) = \dim(\Pi)$.*

*Proof.* Suppose that the initialization algorithm computes a polytope $Q' \subset \Pi$ such that $\dim(Q') < m$. Then there exists vertex $v \in \Pi$, $v \notin \mathrm{Aff}(Q')$ and vector $w \in (\mathbb{R}^m)^\times$ perpendicular to $\mathrm{Aff}(Q')$, such that $w$ belongs to the normal cone of $v$ in $\Pi$ and $\dim(\mathrm{Aff}(Q' \cup v)) > \dim Q'$. This is a contradiction, since such a $w$ would have been computed as $\mathrm{VTX}(\mathcal{A}, w, \pi)$ or $\mathrm{VTX}(\mathcal{A}, -w, \pi)$, where $w$ is normal to the hyperplane $H$ containing $Q'$. $\square$

Incremental Algorithm 1 computes both V- and H-representations of $\Pi$ and a triangulation of $\Pi$, given an inner approximation $Q, Q^H$ of $\Pi$ computed at the initialization. A hyperplane $H$ is called *legal* if it is a supporting hyperplane to a facet of $\Pi$, otherwise it is called *illegal*. At every step of Algorithm 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 $\Pi$ by inserting it to $Q$, i.e. $Q \subset \mathrm{CH}(Q \cup v) \subseteq \Pi$. 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. Algorithm 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; Lemma 8 formalizes the latter case.

**Lemma 8.** *Let $H'$ be a hyperplane supporting a facet constructed by Algorithm 1, and $H \neq H'$ an illegal hyperplane at a previous step. If $H', H$ are parallel then $H'$ is legal.*

*Proof.* Let $w, w'$ be the outer normal vectors of the facets supported by $H, H'$ respectively. If $H, H'$ are parallel then $v = \mathrm{VTX}(\mathcal{A}, w, \pi)$ maximizes the inner product with $w'$ in $Q$ which implies that hyperplane $H'$ is legal. $\square$

The next lemma formulates the termination criterion of our algorithm.

**Lemma 9.** *Let $v = \mathrm{VTX}(\mathcal{A}, w, \pi)$, where $w$ is normal to a supporting hyperplane $H$ of $Q$, then $v \notin H$ if and only if $H$ is not a supporting hyperplane of $\Pi$.*

**Algorithm 1**: Compute$\mathit{\Pi}$ $(A_0, \ldots, A_n, \pi)$

```
Input  : essential $A_0, \ldots, A_n \subset \mathbb{Z}^n$ processed by Lemma 5,
         projection $\pi : \mathbb{R}^{|\mathcal{A}|} \rightarrow \mathbb{R}^m$,
         H-, V-repres. $Q^H, Q$; triang. $T_Q$ of $Q \subseteq \mathit{\Pi}$.
Output: H-, V-repres. $Q^H, Q$; triang. $T_Q$ of $Q = \mathit{\Pi}$.

$\mathcal{A} \leftarrow \bigcup_0^n (A_i \times e_i)$   // 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
            $T_Q \leftarrow T_Q \cup \{$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 hyperplane
            ;
        $Q \leftarrow Q \cup \{v\}$;
        $Q^H \leftarrow Q^H_{temp}$;
return $Q, Q^H, T_Q$;
```

*Proof.* Let $v = \pi(\rho_T)$, where $T$ is a triangulation refining subdivision $S$ in VTX$(\cdot)$. It is clear that, since $v \in \partial\mathit{\Pi}$ extremal with respect to $w$, if $v \notin H$ then $H$ cannot be a supporting hyperplane of $\mathit{\Pi}$. Conversely, let $v \in H$. By the proof of Lemma 6, every other vertex $\pi(\rho'_T)$ on the face of $N(\mathcal{R})$ is extremal with respect to $w$, hence lies on $H$, thus $H$ is a supporting hyperplane of $\mathit{\Pi}$. □

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) [Jos03]. Let $|\mathit{\Pi}|, |\mathit{\Pi}^H|$ be the number of vertices and facets of $\mathit{\Pi}$.

**Lemma 10.** *Algorithm 1 computes at most $|\mathit{\Pi}| + |\mathit{\Pi}^H|$ regular triangulations of $\mathcal{A}$.*

*Proof.* The steps of Algorithm 1 increment $Q$. At every such step, and for each supporting hyperplane $H$ of $Q$ with normal $w$, we compute one vertex of $\mathit{\Pi}$, by Lemma 6. 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 $\mathit{\Pi} \setminus Q$ which are extremal with respect to $w$, hence, an appropriate translate of $H$ also separates the corresponding sets of vertices of $\Sigma(\mathcal{A})$ (Figure 4). 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 $|\mathit{\Pi}|$.

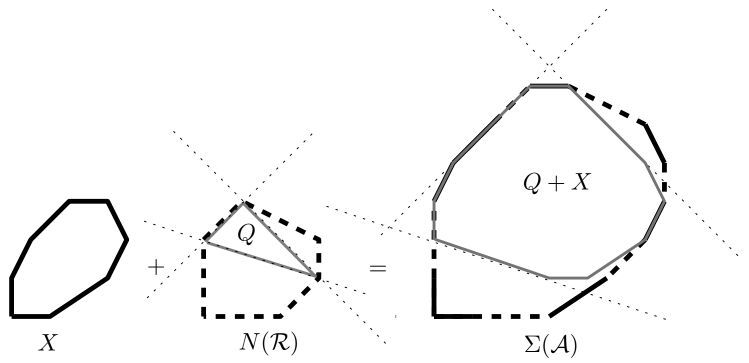


Figure 4: Lemma 10: each illegal hyperplane of $Q$ with normal $w$, separates the already computed vertices of $\Pi$ (here equal to $N(\mathcal{R})$) from new ones, extremal with respect to $w$. $X$ is a polytope such that $X + N(\mathcal{R}) = \Sigma(\mathcal{A})$.

For a legal hyperplane of $Q$, we compute one vertex of $\Pi$ 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 Algorithm 1 (by the earlier discussion concerning the set $W$ of all used normals), the statement follows. $\square$

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$(\cdot)$, and $s_{\Pi}$ that of $\Pi$ computed by Algorithm 1. In VTX$(\cdot)$, the computation of a regular triangulation reduces to a convex hull, computed in $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$; for $\rho_T$ we compute Volume for all cells of $T$ in $O(s_{\mathcal{A}}n^3)$. The overall complexity of VTX$(\cdot)$ becomes $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$. Algorithm 1 calls, in every step, VTX$(\cdot)$ to find a point on $\partial\Pi$ and insert it to $Q$, or to conclude that a hyperplane is legal. By Lemma 10 it executes VTX$(\cdot)$ 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 $\Pi$ 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 $s_{\Pi}$ dominates $s_{\mathcal{A}}$. Moreover, $s_{\Pi}(m+1) \geq |\Pi^H|$. Now, let $\widetilde{O}(\cdot)$ imply that polylogarithmic factors are ignored.

**Theorem 11.** *The time complexity of Algorithm 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 Section 4.

We have proved that oracle VTX$(\cdot)$ (within our algorithm) has two important properties:

1. Its output is a vertex of the target polytope (Lemma 6)
2. When the direction $w$ is normal to an illegal facet, then the vertex computed by the oracle is computed once (Lemma 10)

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 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 $v =$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 $\Pi$. Thus, we have an approximation algorithm by stopping Algorithm 1 when $\mathrm{vol}(Q)/\mathrm{vol}(Q_o)$ achieves a user-defined threshold. Then, $\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$ is bounded by the same threshold. This algorithm is up to 25 times faster than the exact version. Of course, $\mathrm{vol}(Q)$ is available by our incremental convex hull algorithm. However, $\mathrm{vol}(Q_o)$ 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 Implementation and Experiments

We implemented Algorithm 1 in C++ to compute $\Pi$; our code can be obtained from

`http://respol.sourceforge.net`.

All timings shown in this section were obtained on an Intel Core i5-2400 3.1GHz, with 6MB L2 cache and 8GB RAM, running 64-bit Debian GNU/Linux.

Our implementation, `respol`, relies on CGAL, using mainly a preliminary version of package `triangulation` [BDH09] under development, for both regular triangulations, as well as for the V- and H-representation of $\Pi$. For hashing determinants, we employed the implementation described in [EFP12]. This technique can induce a growth in memory consumption. We measured experimentally the memory employed by our implementation in some instances. Last column of Table 1 shows that the memory consumption of our algorithm is related to $|A|$ and $\dim(\Pi)$.

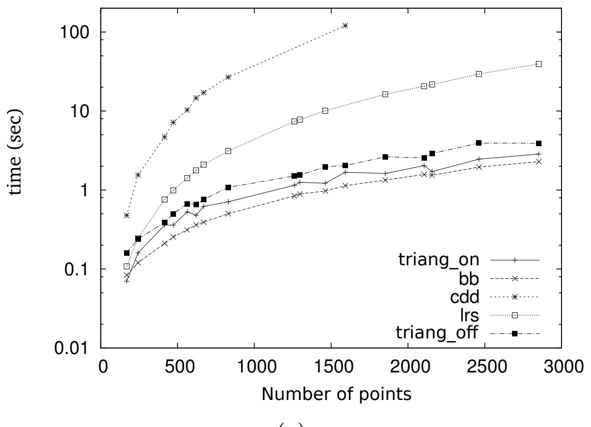
We start our experiments by comparing four state-of-the-art exact convex hull packages: `triangulation` implementing [CMS93] and `beneath-and-beyond (bb)` in `polymake` [GJ01]; double description implemented in `cdd` [Fuk08]; and `lrs` implementing reverse search [Avi00]. We compute $\Pi$, actually extending the work in [ABS97b] for the new class of polytopes $\Pi$. The first was shown to be faster in computing Delaunay triangulations in $\leq 6$ dimensions [BDH09]. The last three 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 $\Pi$ using our implementation and then we give this as an input to the convex hull packages that compute the H-representation of $\Pi$. Moreover, we test `triangulation` by inserting points in the order that Algorithm 1 computes them, while improving the point location of these points since we know by the execution of Algorithm 1 one facet to be removed (online version). The experiments show that `triangulation` and `bb` are faster than `lrs`, which outperforms `cdd`. Furthermore, the online version of `triangulation` is 2.5 times faster than its offline counterpart due to faster point location (Table 1, Figure 5).

A *placing triangulation* of a set of points is a triangulation produced by the Beneath-and-Beyond convex hull algorithm for some ordering of the points. That is, the algorithm places the points in the triangulation with respect to the ordering. Each point, that is going to be placed, is connected to all visible faces of the current triangulation resulting to the construction of new cells. An advantage of `triangulation` is that it maintains a placing triangulation of a polytope in $\mathbb{R}^d$ by storing the $0, 1, d-1, d$ dimensional cells of the triangulation. 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 $\mathcal{A}^{\widehat{w}}$ (Section 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 $|\mathcal{A}| - m$ points in the order they appear in $\mathcal{A}^{\widehat{w}}$ (they all have height zero),

| $m$ | $\lvert\mathcal{A}\rvert$ | # of $\Pi$ | time (seconds) | | | | | | `respol` |
|---|---|---|---|---|---|---|---|---|---|
| | | vertices | `respol` | `tr/on` | `tr/off` | `bb` | `cdd` | `lrs` | Mb |
| 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 and memory consumption 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 $\Pi$.



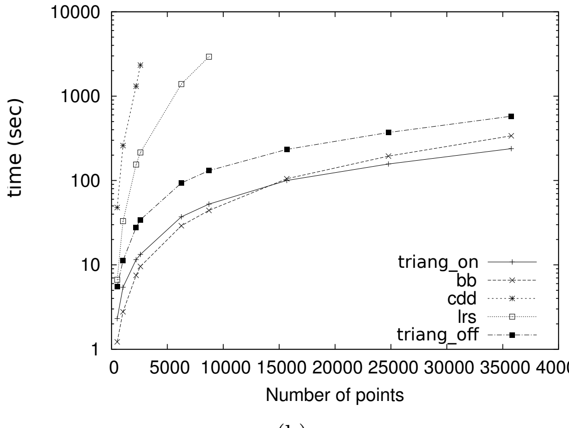


Figure 5: Comparison of convex hull packages for 4-dimensional (a) and 5-dimensional (b) $\Pi$. `triang_on`/`triang_off` are the online/offline versions of `triangulation` package (y-axis is in logarithmic scale).

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 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 Lemma 6, 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 Section 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 with respect to a lifting, faster than the above method [Dev11]. Moreover, we use a modified version of

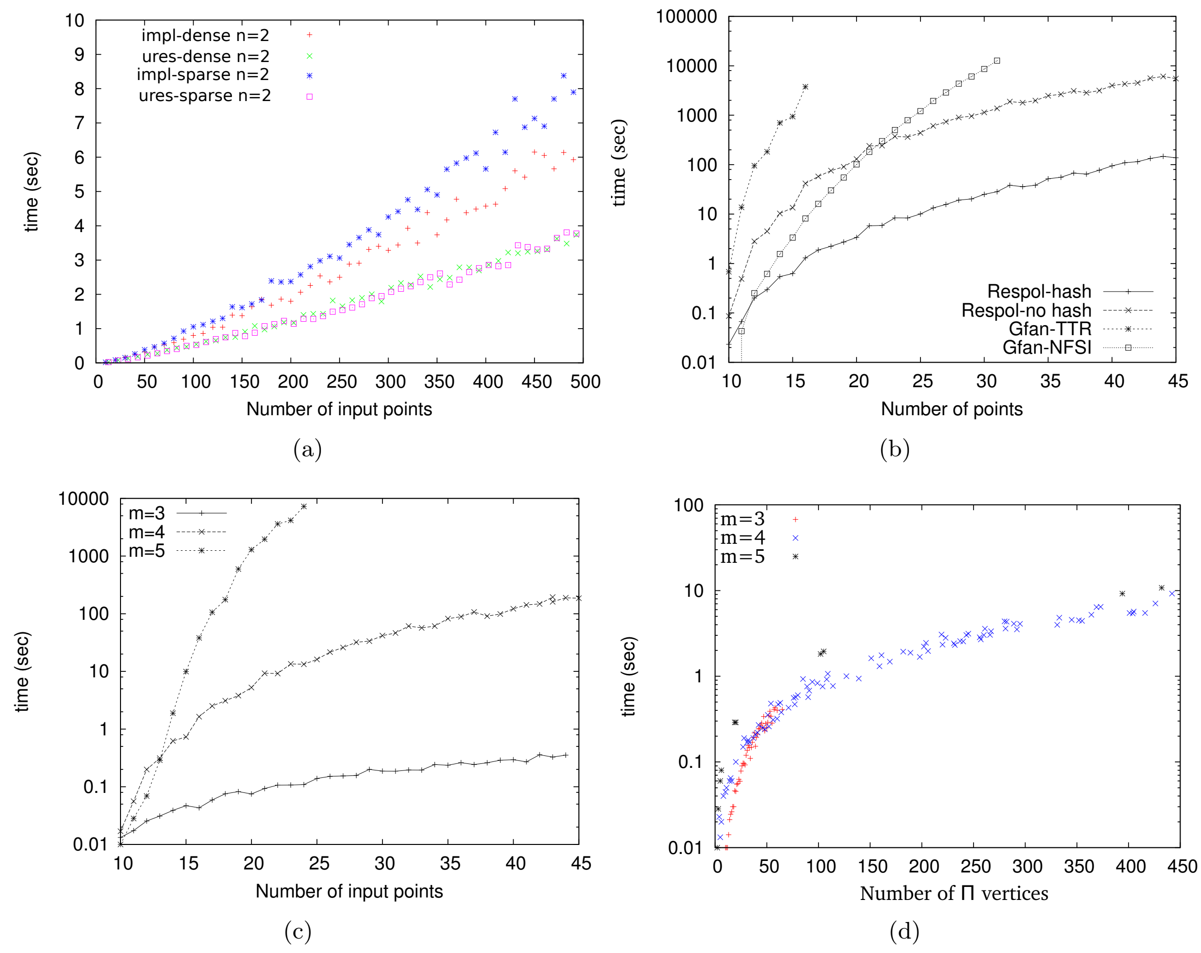


Figure 6: (a) Implicitization and $u$-resultants for $n = 2, m = 3$; (b) Comparison of `respol` (hashing and not hashing determinants) and `Gfan` (traversing tropical resultants and computing normal fan from stable intersection) for $m = 4$; (c) Performance of Alg. 1 for $m = 3, 4, 5$ as a function of input; (d) Performance of Alg. 1 as a function of its output; y-axes in (b), (c), (d) are in logarithmic scale.

`triangulation` in order to benefit from our hashing scheme. Therefore, all cells of the triangulated facets of $\Pi$ 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 $\Pi$.

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 $\delta$-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 $< 1$sec (Figure 6(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 $< 1$sec (Figure 6(a)).

| # cells in triangulation | | | | time (sec) | | | | f-vector of $\Pi$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mu$ | $\sigma$ | min | max | $\mu$ | $\sigma$ | min | max | | | | | | |
| 4781 | 154 | 4560 | 5087 | 0.35 | 0.01 | 0.34 | 0.38 | | | 449 | 1405 | 1438 | 482 |
| 16966 | 407 | 16223 | 17598 | 1.51 | 0.03 | 1.45 | 1.56 | | | 1412 | 4498 | 4705 | 1619 |
| 18229 | 935 | 16668 | 20058 | 1.92 | 0.10 | 1.77 | 2.11 | | 432 | 1974 | 3121 | 2082 | 505 |
| 563838 | 6325 | 548206 | 578873 | 99 | 1.62 | 93.84 | 103.07 | | 9678 | 43569 | 71004 | 50170 | 13059 |
| 289847 | 15788 | 264473 | 318976 | 69 | 4.88 | 61.67 | 77.31 | 1308 | 7576 | 16137 | 16324 | 7959 | 1504 |
| 400552 | 14424 | 374149 | 426476 | 96.5 | 4.91 | 88.86 | 107.12 | 1680 | 9740 | 21022 | 21719 | 10890 | 2133 |

Table 2: Typical f-vectors of projections of resultant polytopes and the size of their triangulations. We perform 20 runs with random insertion order of vertices for each polytope and report the minimum, maximum, average value $\mu$ and the standard deviation $\sigma$ for the number of cells and the runtime.

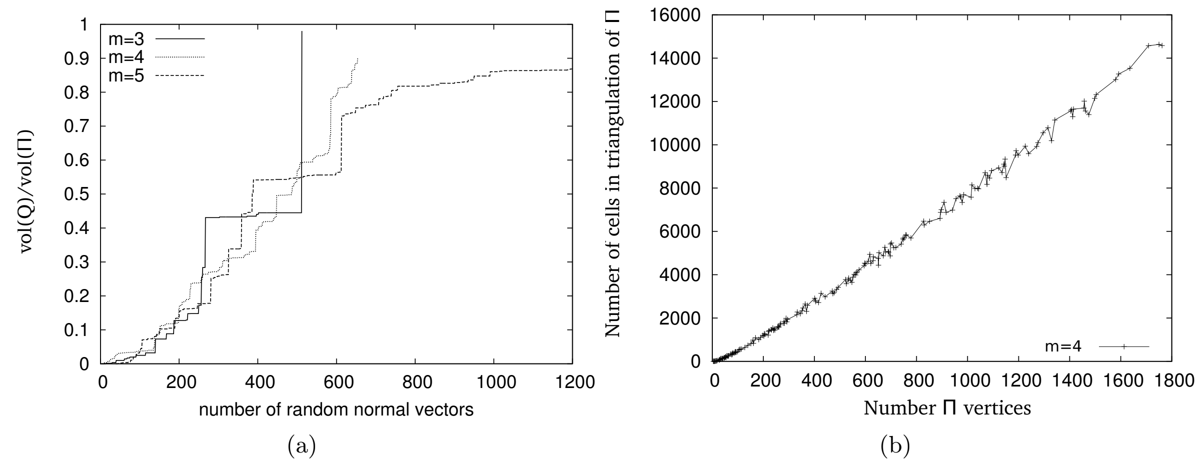


Figure 7: (a) vol($Q$)/vol($\Pi$) as a function of the number of random normal vectors used to compute $Q$; (b) The size of the triangulation of $\Pi$ as a function of the output of Alg. 1.

By using the *hashing determinants* scheme we gain a 18× 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 $n = 3$, $m = 4$ (Figure 6(b)).

We confirm experimentally the *output-sensitivity* of our algorithm. First, our algorithm always computes vertices of $\Pi$ either to extend $\Pi$ or to legalize a facet. We experimentally show that our algorithm has a subexponential behaviour with respect to both input and output (Figure 6(c), 6(d)) and its output is subexponential with respect to the input.

As the complexity analysis (Theorem 11) indicates, the runtime of the algorithm depends on the size of the constructed placing triangulation of $\Pi$. The size of placing triangulation depends on the ordering of the inserted points. We perform experiments on the effect of the inserting order to the size of the triangulation as well as the running time of the computation of the triangulation (Table 2). These sizes as well as the runtimes vary in a very narrow range. Thus, the insertion order is not crucial in both the runtime and the space of our algorithm. Further experiments in 4-dimensional $N(R)$ show that the size of the input bounds polynomially the size of the triangulation of the output (Figure 7(b)) which explains the efficiency of our algorithm in this dimension.

| input | m | 3 | 3 | 4 | 4 | 5 | 5 |
|---|---|---|---|---|---|---|---|
| | $\lvert\mathcal{A}\rvert$ | 200 | 490 | 20 | 30 | 17 | 20 |
| approximation | # 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 |
| algorithm | $\mathrm{vol}(Q_o)/\mathrm{vol}(\Pi)$ | 1.02 | 1.03 | 1.04 | 1.03 | $>$ 10hr | $>$ 10hr |
| | time (sec) | 0.15 | 0.22 | 0.37 | 1.42 | $>$ 10hr | $>$ 10hr |
| uniformly | $\lvert Q\rvert$ | 34 | 45 | 123 | 207 | 228 | 257 |
| | random vectors | 606 | 576 | 613 | 646 | 977 | 924 |
| random | $\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$ | 0.93 | 0.99 | 0.94 | 0.90 | 0.90 | 0.90 |
| | time (sec) | 5.61 | 12.78 | 1.1 | 4.73 | 8.41 | 16.9 |
| exact | # of $\Pi$ vertices | 98 | 133 | 416 | 1296 | 1674 | 5093 |
| algorithm | time (sec) | 2.03 | 5.87 | 3.72 | 25.97 | 51.54 | 239.96 |

Table 3: Results on experiments computing $Q, Q_o^H$ using the approximation algorithm and the random vectors procedure; we stop the approximation algorithm when $\mathrm{vol}(Q)/\mathrm{vol}(Q_o) > 0.9$; the results with random vectors are the average values over 10 independent experiments; "$>$ 10hr" indicates computation of $\mathrm{vol}(Q_o)$ was interrupted after 10hr.

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

We also compare with the implementation of [JY11], 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 compute a union of (possibly overlapping) cones, along with their multiplicities, see Theorem 2.9 of [JY11]. 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 [JY11]. Our method is faster in examples (d), (e), (g), (h) where $m < 7$, is competitive (up to 2 times slower) in (a) where $m = |\mathcal{A}| = 12$ and (i) where $m = 5, |\mathcal{A}| = 20$ and slower in (b), (c), (f) where $m \geq 12$. The bottleneck of our implementation, that makes it slower when the dimension of the projection $m$ is high, is the incremental convex hull construction in $\mathbb{R}^m$. Moreover, since our implementation considers that $N(\mathcal{R})$ lies in $\mathbb{R}^{|\mathcal{A}|}$ instead of $\mathbb{R}^{|\mathcal{A}|-2n-1}$, (see also the discussion on the homogeneities of $\mathcal{R}$ in Section 2), it cannot take advantage of the fact that $\dim(N(\mathcal{R}))$ could be less than $m$ when $|\mathcal{A}| - 2n - 1 < m < |\mathcal{A}|$. This is the case in examples (b), (c) and (f), and performs a convex hull computation in dimension $m$. On the other hand, 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` (Figure 6(b)). However, for $n = 4$, $m = 5$ the beta version of `Gfan` used in our experiments was not stable and always crashed when $|\mathcal{A}| > 13$.

We analyze the computation of inner and outer *approximations* $Q$ and $Q_o^H$. We test the variant of Section 3 by stopping it when $\mathrm{vol}(Q)/\mathrm{vol}(Q_o^H) > 0.9$. In the experiments, the number of $Q$ vertices is $<$ 15% of the $\Pi$ vertices, thus there is a speedup of up to 25 times over the exact algorithm at the largest instances. The approximation of the volume is very satisfactory: $\mathrm{vol}(Q_o^H)/\mathrm{vol}(\Pi) < 1.04$ and $\mathrm{vol}(Q)/\mathrm{vol}(\Pi) > 0.93$ for the tested instances (Table 3). The bottleneck here is the computation of $\mathrm{vol}(Q_o^H)$, where $Q_o^H$ is given in H-representation: the runtime explodes for $m \geq 5$. We use `polymake` in every step to compute $\mathrm{vol}(Q_o^H)$ 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 do not include this computation time in the reported time. Our current work considers ways to extend these observations to a polynomial time approximation algorithm for the volume and the polytope itself when the latter is given by an optimization oracle, as is the case here.

Next, we study procedures that compute only the V-representation of $Q$. For this, we count how many *random vectors* uniformly distributed on the $m$-dimensional sphere are needed to obtain $\mathrm{vol}(Q)/\mathrm{vol}(\Pi) > 0.9$. This procedure runs up to 10 times faster than the exact algorithm (Table 3). Figure 7(a) illustrates the convergence of $\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$ to the threshold value 0.9 in typical $3, 4, 5$-dimensional examples. The basic drawback of this method is that it does not provide guarantees for $\mathrm{vol}(Q)/\mathrm{vol}(\Pi)$ because we do not have sufficient *a priori* information on $\Pi$. These experiments also illustrate the extent in which the normal vectors required to deterministically construct $\Pi$ are uniformly distributed over the sphere.

## 5 Future work

One algorithm that should be experimentally evaluated is the following. We perform a search over the vertices of $\Sigma(A)$, that is, we build a search tree with flips as edges. We keep a set with the extreme vertices with respect to a given projection. Each computed vertex that is not extreme in the above set is discarded and no flips are executed on it, i.e. the search tree is pruned in this vertex. The search procedure could be the algorithm of TOPCOM or the one presented in [MV99] which builds a search tree in some equivalence classes of $\Sigma(A)$. The main advantage of this algorithm is that it does not involve a convex hull computation. On the other hand, it is not output-sensitive with respect to the number of vertices of the resultant polytope; its complexity depends on the number of vertices on the *silhouette* of $\Sigma(A)$, with respect to a given projection and those that are connected by an edge with them.

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 fast enumeration of all regular triangulations for a given (non generic) lifting [Dev11], in which case $\Pi$ may be extended by more than one (coplanar) vertices.

The resultant edges are associated to flips on mixed cells [Stu94]. We studied them algorithmically [EFK10] and may revisit them to generate all neighbors of a vertex of $N(\mathcal{R})$.

Our proposed algorithm uses an incremental convex hull algorithm and it is known that any such algorithm has a worst-case super-polynomial *total time complexity* [Bre96] in the number of input points and output facets. The basic open question that this paper raises is whether there is a polynomial total time algorithm for $\Pi$ or even for the set of $\Pi$ vertices.

**Acknowledgments.** C. Konaxis' research leading to these results has also received funding from the European Union's Seventh Framework Programme (FP7-REGPOT-2009-1) under grant agreement n$^o$ 245749. 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

[ABS97a] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comp. Geom.: Theory & Applic.*, 7:265–301, 1997.

[ABS97b] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comput. Geom.: Theory & Appl.*, 7:265–301, 1997.

[Avi00] 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. Birkhäuser, 2000.

[BDH09] 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.

[BPR03] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry.* Springer-Verlag, Berlin, 2003.

[Bre96] 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.

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

[CLO05] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry.* Number 185 in GTM. Springer, New York, 2nd edition, 2005.

[CMS93] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom.: Theory & Appl.*, 3:185–121, 1993.

[DEF12] A. Dickenstein, I.Z. Emiris, and V. Fisikopoulos. Newton polytopes of resultants and discriminants. Manuscript, 2012.

[Dev11] O. Devillers, 2011. Personal communication.

[EFK10] I.Z. Emiris, V. Fisikopoulos, and C. Konaxis. Regular triangulations and resultant polytopes. In *Proc. Europ. Workshop Computat. Geometry*, Dortmund, Germany, 2010.

[EFK11] I.Z. Emiris, V. Fisikopoulos, and C. Konaxis. An output-sensitive algorithm for computing projections of resultant polytopes. Technical Report CGL-TR-08, NKUA, 2011.

[EFKP12a] Ioannis Emiris, Vissarion Fisikopoulos, Christos Konaxis, and Luis Peñaranda. An oracle-based, output-sensitive algorithm for projections of resultant polytopes. 2012. Submitted for journal publication.

[EFKP12b] Ioannis Emiris, Vissarion Fisikopoulos, Christos Konaxis, and Luis Peñaranda. An output-sensitive algorithm for computing projections of resultant polytopes. In *Proc. Annual ACM Symp. on Computational Geometry*, Chapel Hill, North Carolina, June 2012.

[EFP12] I.Z. Emiris, V. Fisikopoulos, and L. Peñaranda. High dimensional predicates: Algorithms and software. Technical Report CGL-TR-27, NKUA, 2012.

[EKK11] I.Z. Emiris, T. Kalinka, and C. Konaxis. Implicitization using predicted support. In *Proc. Inter. Works. Symbolic-Numeric Computation*, San Jose, Calif., 2011.

[EKKB12] I.Z. Emiris, T. Kalinka, C. Konaxis, and T. Luu Ba. Sparse implicitization by interpolation: Characterizing non-exactness and an application to computing discriminants. In *Proc. ACM Symp. Solid & Phys. Modeling*, Dijon, France, 2012.

[EKKB13] I.Z. Emiris, T. Kalinka, C. Konaxis, and T. Luu Ba. Implicitization using predicted support. *Theor. Comp. Science*, 2013. To appear.

[FP12] V. Fisikopoulos and L. Peñaranda. Faster geometric algorithms via dynamic determinant computation. In *Proc. 20th Europ. Symp. Algorithms*, pages 443–454, 2012.

[Fuk08] K. Fukuda. cdd and cdd+ Home Page. ETH Zürich. www.ifor.math.ethz.ch/∼fukuda/cdd_home/, 2008.

[GJ01] 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.

[GKZ94] I.M. Gelfand, M.M. Kapranov, and A.V. Zelevinsky. *Discriminants, Resultants and Multidimensional Determinants*. Birkhäuser, Boston, 1994.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, New York, 1988.

[IMTI02] H. Imai, T. Masada, F. Takeuchi, and K. Imai. Enumerating triangulations in general dimensions. *Intern. J. Comput. Geom. Appl.*, 12(6):455–480, 2002.

[Jos03] M. Joswig. Beneath-and-beyond revisited. In M. Joswig and N. Takayama, editors, *Algebra, Geometry, and Software Systems*, Mathematics and Visualization. Springer, Berlin, 2003.

[JY11] A. Jensen and J. Yu. Computing tropical resultants. *arXiv:math.AG/1109.2368v1*, 2011.

[Kap91] M.M. Kapranov. Characterization of A-discriminantal hypersurfaces in terms of logarithmic Gauss map. *Math. Annalen*, 290:277–285, 1991.

[LRS10] 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.

[MC00] T. Michiels and R. Cools. Decomposing the secondary Cayley polytope. *Discr. Comput. Geometry*, 23:367–380, 2000.

[MV99] T. Michiels and J. Verschelde. Enumerating regular mixed-cell configurations. *Discr. Comput. Geometry*, 21(4):569–579, 1999.

[Ram02] J. Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. In *Proc. Intern. Congress Math. Software*, pages 330–340, 2002.

[Rin12] F. Rincón. Computing tropical linear spaces. In *J. Symbolic Computation (to appear)*, 2012.

[Stu94] B. Sturmfels. On the Newton polytope of the resultant. *J. Algebraic Combin.*, 3:207–236, 1994.

[SY08] 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.

[Zie95] G.M. Ziegler. *Lectures on Polytopes*. Springer, 1995.