Exact and approximate algorithms for resultant polytopes

Ioannis Z. Emiris* Vissarion Fisikopoulos* Christos Konaxis†

Abstract

We develop an incremental algorithm to compute the Newton polytope of the resultant, aka resultant polytope, or its projection along a given direction. Our algorithm exactly computes vertex- and halfspace-representations of the resultant polytope using an oracle producing resultant vertices in a given direction. It is output-sensitive as it uses one oracle call per vertex. 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. Compared to tropical geometry software, ours is faster up to dimensions 5 or 6, and competitive in higher dimensions. Compared to `lrs`, `cdd`, and `polymake`, the computation of convex hull is fastest along with `polymake`. The resultant is fundamental in algebraic elimination and in implicitizing parametric hypersurfaces: we compute the Newton polytope of surface equations in < 1sec, when there are < 100 terms in the parametric polynomials, which includes all common instances in geometric modeling. Our method computes instances of 5, 6 or 7-dimensional polytopes with 35K, 23K or 500 vertices, respectively, in < 2hr.

Keywords. general dimension, convex hull, regular triangulation, polynomial resultant, CGAL implementation, experimental complexity

1 Introduction

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

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

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, $|\mathcal{A}| = \sum_i |A_i|$, where $|\cdot|$ denotes cardinality. By Cayley's trick ([13, sec.5]) the regular tight mixed subdivisions of the Minkowski sum $A_0 + \cdots + A_n$ are in bijection with the regular triangulations of $\mathcal{A}$; the latter correspond to the vertices of the *secondary polytope* $\Sigma(\mathcal{A})$.

The *Newton polytope* of a polynomial is the convex hull of the set of exponent vectors of monomials with nonzero coefficient. Given $n+1$ polynomials in n variables, with fixed exponent sets A_i, $i = 0, \dots, n$, 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. 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. Our algorithms compute the Newton polytope of the resultant $N(\mathcal{R})$, or *resultant polytope*, in particular when some of the coefficients are not symbolic, in which case we seek a projection of the resultant polytope.

We exploit an equivalence relation defined on the secondary vertices. The class representatives correspond bijectively to the resultant vertices. This information defines an oracle producing a resultant vertex in a given direction, and avoids computing $\Sigma(\mathcal{A})$, which has much more vertices than $N(\mathcal{R})$. Although there exist efficient software computing $\Sigma(\mathcal{A})$ [12], 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, whereas $\Sigma(\mathcal{A})$ is intractable.

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

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

By reindexing, this is the subspace of the first m coordinates. It is possible that none of the coefficients is specialized, hence $m = |\mathcal{A}|$, π is trivial, and $\Pi = N(\mathcal{R})$. Assuming the specialized coefficients take sufficiently generic values, Π is the Newton polytope of the corresponding specialization of $\mathcal{R}$.

Let us review previous work. Sparse (or toric) elimination theory was introduced in [9], where $N(\mathcal{R})$ is described via Cayley's trick. In [13, sec.6] is proven that $N(\mathcal{R})$ is 1-dimensional iff $|A_i| = 2$, for all i, the

*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece. {emiris,vissarion}@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 FP7 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@acmac.uoc.gr. Enjoys support from the FP7-REGPOT-2009-1 project "Archimedes Center for Modeling, Analysis and Computation"

only planar polytope is the triangle, whereas the only 3-dimensional ones are the tetrahedron, the square-based pyramid, and the polytope of two univariate trinomials. Following [13, Thm.6.2], the 4-dimensional polytopes include the 4-simplex, some $N(\mathcal{R})$ obtained by pairs of polynomials, and those of 3 trinomials, which we can investigate with our code: for example, we computed one with f-vector $(19, 57, 57, 19)$. Tropical geometry is a polyhedral analogue of algebraic geometry which gives alternative ways of recovering resultant polytopes [10].

For a more detailed presentation of the background of this paper see [5].

2 Algorithms and complexity

This section presents our exact and approximate algorithms for computing an orthogonal projection Π of $N(\mathcal{R})$ without computing $N(\mathcal{R})$, and analyzes their asymptotic complexity.

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

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 Extreme$\Pi(\mathcal{A}, w, \pi)$ 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 Extreme$\Pi(\mathcal{A}, w, \pi)$ and Extreme$\Pi(\mathcal{A}, -w, \pi)$, 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 1 *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 $v =$ Extreme$\Pi(\mathcal{A}, w, \pi)$ 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 CH$(\cdot)$ denotes convex hull. This happens when

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

```
Input : $A_0, \dots, A_n \subset \mathbb{Z}^n$, $\pi : \mathbb{R}^{|\mathcal{A}|} \to \mathbb{R}^m$,
        H-, V-rep. $Q^H, Q$, triang. $T_Q$ of $Q \subseteq \Pi$
Output: H-, V-rep. $Q^H, Q$, triang. $T_Q$ of $Q = \Pi$

$\mathcal{A} \leftarrow \bigcup_0^n (A_i \times e_i)$; $\mathcal{H}_{illegal} \leftarrow \emptyset$
foreach $H \in Q^H$ do $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \cup \{H\}$
while $\mathcal{H}_{illegal} \neq \emptyset$ do
    select $H \in \mathcal{H}_{illegal}$; $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \setminus \{H\}$
    $w$ is the outer normal vector of $H$
    $v \leftarrow$ Extreme$\Pi(\mathcal{A}, w, \pi)$
    if $v \notin H \cap Q$ then
        $Q^H_{temp} \leftarrow$ conv$(Q \cup \{v\})$
        foreach $(d-1)$-face $f \in T_Q$, $f \subset \partial H$ do
            $T_Q \leftarrow T_Q \cup \{$faces of $conv(f, v)\}$
        foreach $H' \in \{Q^H \setminus Q^H_{temp}\}$ do
            $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \setminus \{H'\}$
        foreach $H' \in \{Q^H_{temp} \setminus Q^H\}$ do
            $\mathcal{H}_{illegal} \leftarrow \mathcal{H}_{illegal} \cup \{H'\}$
        $Q \leftarrow Q \cup \{v\}$;  $Q^H \leftarrow Q^H_{temp}$
return $Q, Q^H, T_Q$
```

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 insert v to Q.

Let us examine degenerate cases that may appear during execution of Alg. 1. Let w be a normal to a supporting hyperplane H of a facet of Q s.t. 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 Extreme$\Pi(\mathcal{A}, w, \pi)$ returns $\pi(\rho_T)$, this is a point on $\partial\Pi$ but not a vertex of Π. Of course, in subsequent steps of the algorithm, the vertices of $\pi(f)$ will be computed, but this jeopardizes the output-sensitivity of the algorithm. 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 $\pi(f)$. The perturbation can be implemented in ExtremeΠ, without affecting any other parts of the algorithm. In practice, our implementation does avoid degenerate cases.

Lemma 2 *Let vertex v be computed by Extreme$\Pi(\mathcal{A}, w, \pi)$, where w is normal to a supporting hyperplane H of Q, then $v \notin H \Leftrightarrow H$ is not a supporting hyperplane of Π.*

We now bound the *complexity* of our algorithm. Beneath-beyond, given a k-dimensional polytope with

l vertices, computes its H-representation and a triangulation in $O(k^5lt^2)$, where t is the number of full-dimensional faces (cells) [11]. Let $|\Pi|, |\Pi^H|$ be the number of vertices and facets of Π.

Lemma 3 *Alg. 1 computes at most* $|\Pi| + |\Pi^H|$ *regular triangulations of* $\mathcal{A}$.

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 ExtremeΠ, and s_Π that of Π computed by Alg. 1. In ExtremeΠ, the computation of a regular triangulation reduces to a convex hull, computed in $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$; to compute ρ_T we need to compute the volume of all cells in T; this is done in $O(s_{\mathcal{A}}n^3)$. The overall complexity of ExtremeΠ becomes $O(n^5|\mathcal{A}|s_{\mathcal{A}}^2)$. Alg. 1 calls, in every step, ExtremeΠ to find a point on $\partial\Pi$ and insert it to Q, or conclude that a hyperplane is legal. By Lem. 3 it executes ExtremeΠ 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 4 *The time complexity of Alg. 1 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. The performance of our algorithm confirms this property, see experimental evidence in Sect. 3.

Our algorithm readily yields an approximate variant: for each supporting hyperplane, we use its normal w to compute v =Extreme$\Pi(\mathcal{A}, w, \pi)$. 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 vol(Q)/vol(Q_o) achieves a user-defined threshold. Then, vol(Q)/vol(Π) is bounded by the same threshold. Of course, vol(Q) is available by our incremental convex hull algorithm. However, vol(Q_o) is the critical step; we plan to examine algorithms that update (exactly or approximately) this volume.

3 Implementation and Experiments

We implement Alg. 1 in C++ to compute Π; our code is available in `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, working in general dimension, for both regular triangulations as well as for the vertex and halfspace-representation of Π.

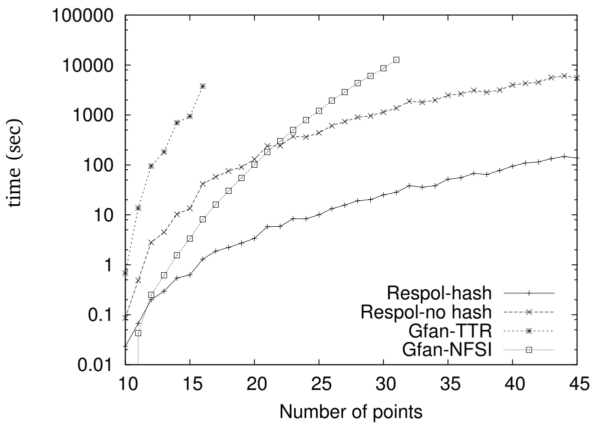

Figure 1: Comparison of `respol` (hashing and not hashing determinants) and 2 algorithms of `Gfan` for $m = 4$; y-axis in logarithmic scale.

We first compare 4 state-of-the-art exact convex hull packages, `triangulation` [3], `polymake`'s `beneath-beyond (bb)` [8]; `cdd` [7], and `lrs` [1], to compute the H-rep. from the vertices of Π (offline version) actually extending the work in [2] for the new class of polytopes Π. We also test `triangulation` by inserting points in the order that Alg. 1 computes them (online version). The experiments show that `triangulation`, `bb` are faster than `lrs`, `cdd` and `triangulation` online is 2.5 times faster than offline (Table 1). Moreover, `triangulation` maintains a polytope with its boundary and its interior triangulated which is useful when we compute regular triangulations wrt to a lifting (ExtremeΠ).

We perform an experimental analysis of our algorithm confirming its *output-sensitivity*: its behaviour is subexponential wrt to both input and output and its output is subexponential wrt the input. Moreover, the size of the input bounds polynomially the size of the triangulation of the output.

The resultant is fundamental in *elimination*, and in *implicitizing* parametric hypersurfaces: we compute the polytope of surface equations in < 1sec, assuming < 100 terms in parametric polynomials, which includes all common instances in geometric modeling, whereas the corresponding $\Sigma(\mathcal{A})$ are intractable. By using the *hashing determinants* scheme [6] we gain a speedup of 18, 100 times when $m = 3, 4$ respectively. For $m = 4$ 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 us to compute instances of the problem that would be intractable otherwise. We explore the *limits* of our implementation. By bounding runtime to < 2hr, we compute instances of 5-, 6-, and 7-dimensional Π with 35K, 23K and 500 vertices, resp. (Table 1). We also compare with the implementation of [10], based on `Gfan` library. Our code

$dim(\Pi)$	$\|\mathcal{A}\|$	# of Π vertices	time					
			`respol`	`triang/on`	`triang/off`	`bb`	`cdd`	`lrs`
3	2490	318	85.03	0.07	0.10	0.067	1.20	0.097
4	37	2852	97.82	2.85	3.91	2.29	335.23	39.41
5	24	35768	4610.31	238.76	577.47	339.05	> 1hr	> 1hr
6	19	23066	6556.42	1191.8	2754.3	> 1hr	> 1hr	> 1hr
7	17	500	302.61	267.01	614.34	603.12	10495.14	358.79

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

is faster up to dimensions 5, 6, and competitive in higher dimensions.

We analyze the computation of inner and outer *approximations* Q and Q_o^H. We test the variant of Sect. 2 by stopping it when $\frac{\mathrm{vol}(Q)}{\mathrm{vol}(\Pi)} > 0.9$. 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 $\frac{\mathrm{vol}(Q_o^H)}{\mathrm{vol}(\Pi)} < 1.04$. Next, we study procedures that compute only V-rep. of Q by counting the *random vectors* uniformly distributed on the m-sphere needed to obtain $\frac{\mathrm{vol}(Q)}{vol(\Pi)} > 0.9$. This procedure runs up to 10 times faster than the exact algorithm but does not provides guarantees for $\frac{\mathrm{vol}(Q)}{\mathrm{vol}(\Pi)}$.

4 Future work

The resultant edges are associated to certain flips on mixed cells [13]. We have studied them algorithmically [4] and may revisit them in conjunction with the current approach: for every computed vertex of $N(\mathcal{R})$ apply all such flips to generate its neighbors.

A theoretical question is whether there is a *polynomial total time* incremental algorithm for Π. For this we wish to study the structure of $N(\mathcal{R})$; [9, 13], whereas our code sheds light to this issue by computing examples.

Acknowledgment. We thank L. Peñaranda for his useful help on implementation and experiments. We also thank O. Devillers and S. Hornus for discussions on `triangulation`, and A. Jensen and J. Yu for discussions on their approach 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 and Computation*, volume 29 of *Oberwolfach Seminars*, pp. 177–198. Birkhäuser-Verlag, 2000.

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

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

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

[5] I.Z. Emiris, V. Fisikopoulos, C. Konaxis, and L. Peñaranda. An output-sensitive algorithm for computing projections of resultant polytopes. arXiv:1108.5985v2 [cs.SC], 2011.

[6] I.Z. Emiris, V. Fisikopoulos, and L. Peñaranda. Optimizing the computation of sequences of determinantal predicates. Technical Report CGL-TR-14, NKUA, 2011.

[7] K. Fukuda. cdd and cdd+ home page. ETH Zürich. http://www.ifor.math.ethz.ch/~fukuda/cdd_home, 2008.

[8] E. Gawrilow and M. Joswig. Polymake: an approach to modular software design in computational geometry. In *Proc. Annual ACM Symp. Comp. Geom.*, pp. 222–231. ACM Press, 2001.

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

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

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

[12] J. Rambau. TOPCOM: Triangulations of point configurations and oriented matroids. In A.M. Cohen, X-S. Gao, and N. Takayama, eds., *Math. Software: ICMS*, pp. 330–340. World Scientific, 2002.

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