
ELSEVIER

Computational Geometry 22 (2002) 143–166

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Enumerating a subset of the integer points inside a Minkowski sum

Ioannis Z. Emiris

INRIA, B.P. 93, Sophia-Antipolis 06902, France

Communicated by S. Fortune; received 1 July 2000; accepted 30 November 2000

Abstract

Sparse elimination exploits the structure of algebraic equations in order to obtain tighter bounds on the number of roots and better complexity in numerically approximating them. The model of sparsity is of combinatorial nature, thus leading to certain problems in general-dimensional convex geometry. This work addresses one such problem, namely the computation of a certain subset of integer points in the interior of integer convex polytopes. These polytopes are Minkowski sums, but avoiding their explicit construction is precisely one of the main features of the algorithm. Complexity bounds for our algorithm are derived under certain hypotheses, in terms of output-size and the sparsity parameters. A public domain implementation is described and its performance studied. Linear optimization lies at the inner loop of the algorithm, hence we analyze the structure of the linear programs and compare different implementations.

Keywords: Minkowski sum; Integral polytope; Integer point; Linear programming; Newton polytope

1. Introduction

A major computational problem in algebraic geometry concerns the numerical approximation of all common roots of a system of polynomial equations. Methods based on *resultant matrices* can exploit the sparse structure of the input polynomials, are robust to input perturbations, and have lower worst-case complexity than Gröbner bases, which is the best-established and most general method today [15]. Exploiting structure is achieved in a strong and predictable way by the theory of *sparse elimination*; see, e.g., [15,17,19] and the next two sections. This theory has generalized several results of classical variable elimination theory. The model of sparsity is a combinatorial one, and raises several problems in general-dimensional convex geometry. One bottleneck is due to the computational question examined in this paper:

E-mail address: Ioannis.Emiris@inria.fr (I.Z. Emiris).
URL address: http://www-sop.inria.fr/galaad/emiris (I.Z. Emiris).

0925-7721/01/$ – see front matter
PII: S0925-7721(01)00049-9

Problem 1. Consider the Minkowski sum of n convex polytopes lying in $\mathbb{R}^n$ with integer coordinate vertices, and suppose a direction is specified. One problem is to compute the set of *integer points* in this Minkowski sum, having positive distance from the polytope boundary along the given direction. Usually, a positive lower bound on the distance or an upper bound on the set's cardinality may be supplied. In this case, we seek the subset of points satisfying all bounds provided.

We have implemented a simple and efficient algorithm, which avoids constructing the Minkowski sum and implements certain branch-and-bound heuristics for searching the integer lattice by exploiting properties of the distance function. Asymptotic complexity bounds are derived both in terms of the sparsity parameters of the problem and as a function of output size, provided certain conditions are met, as specified below. Roughly, these conditions prohibit long and thin polytopes whose volume is significantly lower than the number of interior integer points. Our algorithm improves significantly upon an early version [19], mainly by using properties of convexity and projection to lower dimensions. A public domain implementation is presented and applied to different input instances.

The next section mentions related work. Section 3 outlines sparse elimination theory and defines the problem at hand. Section 4 presents our algorithm, whose building block is a linear programming subroutine, and the various heuristics used to prune the search space. Special attention to the structure of the optimization problems is paid in Section 5. The asymptotic complexity analysis is found in Section 6. Section 7 describes the implementation and illustrates its performance. Experimental results serve to confirm the asymptotic bounds. Moreover, we compare the simplex code in our program with the linear programming functionalities of packages `cdd+` and `lrs`. We conclude with directions of further research.

A preliminary account of this work has appeared as [18].

2. Related work

Most existing work on Minkowski sums of convex polytopes limits itself to low dimensions [32,34, 46], or to special cases like zonotopes [21]. Among the former, we note the result in [32] that settles the 3-dimensional case by showing that Minkowski addition has complexity bounded by the sum of input and output sizes. As for zonotopes, they are the hardest inputs on which Minkowski addition may be applied, hence their interest.

A general treatment is given by Gritzmann and Sturmfels [28], who consider both arithmetic and bit computational models, and measure input size in terms of two independent parameters, namely space dimension and number of summand polytopes. It is then shown that neither a facet nor a vertex representation of the Minkowski sum can be computed in polynomial time, regardless of the model used. However, if the summand and sum polytopes are all represented by their vertices and, in addition, one of the two input parameters is fixed, then Minkowski addition has polynomial complexity in at least one of the two models. Interestingly, the basis of the corresponding algorithms is linear programming.

In our case, constructing explicitly the Minkowski sum is to be avoided; we shall opt for a method that directly enumerates the points sought, thus reminiscent of [9]. The algorithm achieves complexity proportional to the number of output points under certain assumptions; for each point, the cost is polynomial in the number of vertices of the input polytopes and the dimension. On the other hand,

computing explicitly the Minkowski sum would require a convex hull operation on a number of points exponential in the dimension, irrespective of the output size.

The principal branch-and-bound heuristic enumerates integer points in *projections* of the Minkowski sum, then lifts the interesting subset to higher dimensions. Again, no explicit computation of the projected polytope is required. Our heuristics reject a small fraction of interior integer points in polyhedral projections to small dimensions; in fact, the smaller the dimension, the fewer candidates are eliminated. In this case, it may be interesting to transform interior point enumeration to the computation of 0/1-vertices of a polytope in some high dimension [23]. If the interior point coordinates are expressed by integer unknowns, say x, these can be written as $x = \sum_{i=0}^{k} x_i 2^i$, where the x_i are binary variables; any available information may be used to bound the binary size k. The vertex enumeration problem could be efficiently solved by available software, preferably after expressing the projected polytope as the intersection of half-spaces rather than as a point convex hull.

Let us assume for a moment that the Minkowski sum had been computed by some arbitrary-dimension convex hull software. Then we may concentrate on the point enumeration problem per se. An efficient implementation which we have tested to enumerate *all* integer points is `porta` [13]. The practical complexity of the method is discussed in Section 7. Counting the number of integer points in an integer polytope, irrespective of its representation, is #P-complete when the dimension is not fixed; the same holds even for zonotopes. Otherwise, there are polynomial-time algorithms for vertex-represented as well as hyperplane-represented polytopes. Volume computation in general dimension is also #P-complete [27,29]. Applications of point enumeration are related to the probabilistic estimation of volume as well as several problems in optimization, most notably integer programming, and polyhedral combinatorics. See [27,29] and the references therein.

In bounding complexity, we are concerned with the asymptotic behaviour of the number of integer points in the interior of a convex polytope with integer vertices. Ehrart [16] established an asymptotic bound by the volume of the polytope. Alternatively, for any full-dimensional polytope $Q^{(n)} \subset \mathbb{R}^n$, $\mathrm{vol}(Q^{(n)})$ can be approximated by $\#(Q^{(n)} \cap \mathbb{Z}^n)$ by an adequate sampling with known error estimates; see [27, Section 3] and its references. We may also use simple inequalities, such as $\#(Q^{(n)} \cap \mathbb{Z}^n) \leqslant \mathrm{vol}(Q^{(n)} + C)$, where C is the Dirichlet cell of 0, i.e., C contains all points in $\mathbb{R}^n$ closer to the origin than to any other lattice point [29]. For polytopes that are not too "thin", the second part of this inequality is very close to, and asymptotically dominated by, the polytope's volume. Another line of work, particularly useful with Minkowski sums, concerns formulas of inclusion-exclusion type or those that reduce the problem to counting points in lower-dimensional faces; see [4,7].

Point enumeration is of course important in sparse elimination theory because of the bijection between integer points and monomials, where we assume that all variables are nonzero. The principal computational object of interest is *the sparse resultant* and its matrix, whose determinant is a nontrivial multiple of the resultant and which reduces system solving to an eigenproblem or univariate factorization. The next section details combinatorial methods for exploiting the monomial structure of the input polynomials; see also [15,19,33,45]. In general, the resultant (or eliminant) provides an efficient approach in terms of asymptotic complexity for solving 0-dimensional polynomial systems and has complexity simply exponential in the dimension and polynomial in the number of roots. Similar bounds can be achieved by *straight-line programs*, which use an algebraic model of sparseness; see, e.g., [26]. *Fewnomials* [36] provide an alternative model of sparseness, but have yet to lead to a comprehensive theory. *Gröbner* (or standard) *bases* may exploit sparseness only implicitly and have worst-case complexity doubly exponential in the number of variables, even when the input polynomials have

bounded degree [40]. Of course, they provide a complete arsenal for studying and performing arithmetic between polynomial ideals.

Linear programming shall provide the main building block and will be crucial in all respects, including algorithm design, asymptotic complexity analysis as well as practical efficiency of the program. For a general introduction and a comprehensive list of references see [8,30]. Complexity bounds will rely on polynomial-time interior-point methods, namely the one in [47]. This is best suited to our problems. The worst-case bit complexity per linear program is in $O((V^2 + CV + C^{1/2})VL^2 \log L)$, where L represents the bit size of the input coefficients and V, C stand for the number of variables and constraints, respectively, where C includes the constraints on the variables.

Different public domain software is available for linear programming, see, e.g., [41]. Our requirement for freely available software has excluded powerful packages, such as the state-of-the-art solver from CPLEX Optimization Inc. Another characteristic is that we are not interested in exact solutions because the inputs are integers and the output can be approximate. We have implemented the simplex method based on code from [42], but have also experimented with the reverse search and double description implementations in `lrs` and `cdd+` respectively [2,22] to solve linear programs.

Important work has been done for linear programming *queries* in fixed dimension, where the constraints do not change between successive optimizations of different objective functions [12,39,44]. This is very relevant here, except that dimension is an input parameter. Possibly relevant work concerns incremental approaches, such as those that could be derived from [1,14,25]. In the latter work, the idea is to randomly choose a subset (ε-net) of constraints so that, with high probability, the solution to the subproblem satisfies a large number of constraints.

3. Sparse elimination theory

This section sketches the theory of sparse variable elimination and the main combinatorial concepts required. Further information can be found in [15,17,19]. Sparse elimination allows us to consider *Laurent* polynomials $f \in K[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$, where K is an arbitrary field of coefficients.

Definition 2. The *support* of polynomial $f \in K[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$ is the set of exponent vectors in $\mathbb{Z}^n$, corresponding to terms in f whose coefficient is nonzero. The *Newton polytope* of f is the convex hull of its support in $\mathbb{R}^n$.

The Newton polytope refines the notion of total degree in classical elimination theory (see Fig. 1). For point sets A, B in $\mathbb{R}^n$, the *Minkowski sum* $A + B = \{a + b \mid a \in A, b \in B\}$ is convex if A and B are convex. Minkowski addition is associative and commutative.

Definition 3. Given convex polytopes $A_1, \ldots, A_n \subset \mathbb{R}^n$, their *mixed volume* $MV(A_1, \ldots, A_n) \in \mathbb{R}_{\geqslant 0}$ is the unique symmetric function, multilinear with respect to Minkowski addition and scalar multiplication, such that $MV(A_1, \ldots, A_1) = n!\,\mathrm{vol}(A_1)$, where $\mathrm{vol}(\cdot)$ denotes euclidean volume. Equivalently, it is the coefficient of $\lambda_1 \cdots \lambda_n$ in $\mathrm{vol}(\lambda_1 A_1 + \cdots + \lambda_n A_n)$.

This definition differs from the one in [31] by the factor $n!$. The following theorem provides the stepping stone in applying this notion to computational algebra and effective algebraic geometry.

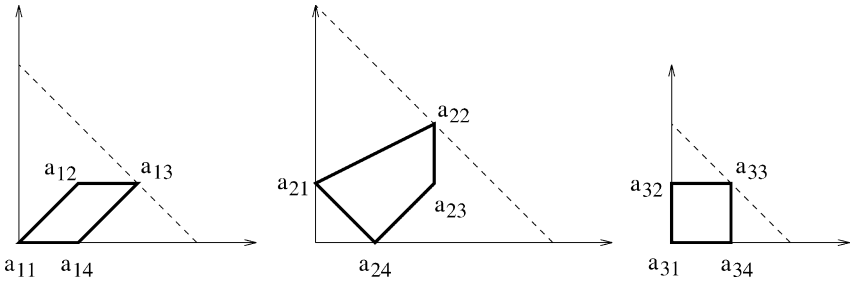

Fig. 1. Newton polytopes of the given polynomials and of the dense polynomials of the same total degree.

Theorem 4. *Consider n polynomials $f_i \in K[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$ with Newton polytopes Q_i. The number of common isolated roots in $(\overline{K}^*)^n$ does not exceed $MV(Q_1, \ldots, Q_n)$, where $\overline{K}$ is the algebraic closure of K and $\overline{K}^* = \overline{K} \setminus \{0\}$.*

This was first stated by Bernstein [6], but relies critically on work by Khovanskii and Kushnirenko [35, 37], hence its naming as the BKK bound. It provides a sharper bound than the classical Bézout bound which is the product of total degrees. Several extensions have been established, including a study of the cases where equality holds [45], as well as the case of $\overline{K}^n$, covered by a generalization of the mixed volume to the *stable* mixed volume [33].

Mixed volume computation provides important information for numerically approximating the common zeros of the given system, either by a sparse homotopy, or by resultant-based methods, for we obtain a monomial basis of the quotient ring [15]. The solution of well-constrained algebraic systems can be reduced either to an eigenproblem or to univariate polynomial factorization, both reductions relying on the resultant. The resultant is a new polynomial in the coefficients of the input equations, which characterizes the existence of common roots in the input system. Resultants are most conveniently expressed as (divisors of) matrix determinants. There are several types of such matrices whose study was initiated by such luminaries as Euler, Bézout and Cayley; today, this is a very active field both of algorithmic research as well as software development. This paper focuses on sparse elimination theory, which has introduced *sparse resultant matrices* in order to express the sparse, or toric, resultant. Since they depend on the corresponding Newton polytopes, they are also known as Newton matrices.

To solve a well-constrained system of n polynomials in n variables, it turns out that we must consider an overconstrained system $f_0, \ldots, f_n$ in $K[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$, obtained either by adding a polynomial of our choice to the given system, or by "hiding" a variable in the coefficient field (and incrementing n). For instance, well-known sparse resultant matrices include the matrix of coefficients, when all $n+1$ polynomials are linear, and the Sylvester matrix, when $n = 1$.

Let $MV_{-i} = MV(f_0, \ldots, f_{i-1}, f_{i+1}, \ldots, f_n)$, which stands for the mixed volume of the respective Newton polytopes. It is assumed that the affine lattice generated by the input supports is n-dimensional; otherwise a smaller system may be considered [15,33]. This lattice can be identified with $\mathbb{Z}^n$, modulo a change of variables, implemented by means of Smith's normal form. Then the sparse resultant's degree in the coefficients of f_i is MV_{-i}, for $i = 0, \ldots, n$, and the total degree is $\sum_{i=0}^{n} MV_{-i}$, which gives the *optimal* dimension of any sparse resultant matrix.

A sparse resultant matrix has rows indexed by sets $B_0, \ldots, B_n \subset \mathbb{Z}^n$, with cardinality $\#B_i \geqslant MV_{-i}$. Let $Q = Q_0 + \cdots + Q_n \subset \mathbb{R}^n$ be the Minkowski sum of all Newton polytopes. Let

$$Q_{-i} = \sum_{j \neq i} Q_j, \quad T_i \subset Q_{-i} \cap \mathbb{Z}^n, \qquad i = 0, \ldots, n,$$

where the T_i are defined to be the computed point sets. We shall use $Q^{(n)}$ to denote some arbitrary Q_{-i}. In all sparse resultant algorithms $B_i \subset T_i$, but the precise way of selecting the T_i and the B_i makes these algorithms differ. The smaller the T_i the better, and this depends on the prior knowledge available in order to bound the point search. The smallest matrices are, in general, constructed by the *incremental algorithm* of [19], which tries out successively larger matrices, specified by successively larger point sets B_i.

Definition 5 [19]. Given a convex polytope $A \subset \mathbb{R}^n$ and a nonzero vector $v \in (\mathbb{R}^*)^n$, we define the *v-distance* $\delta_v(p)$ of any point $p \in A \cap \mathbb{Z}^n$ to be the maximum nonnegative real such that $p + \delta_v(p)v \in A$. If $|v|_2 = 1$, then $\delta_v(p)$ is the euclidean distance of p from the boundary of A in the direction v. Furthermore, $\delta_{\bar{v}}(\bar{p})$ will stand for the $\bar{v}$-distance of point $\bar{p} \in \mathbb{Z}^k$ inside some polytope in $\mathbb{R}^k$, where $\bar{v}$ is the projection of v to $(\mathbb{R}^*)^k$.

One can think of each set B_i including successively larger subsets of T_i upon demand, so that the minimum v-distance in B_i is maximized. Set B_i is initialized with MV_{-i} points, then incremented until a valid matrix is obtained with an additional goal, that $\min\{\delta_v(p)\colon\ p \in B_i\}$ be the same for all i. It has been proven that this process always terminates. In the current implementation, the T_i are computed once, under the hypothesis that they contain enough points for the matrix to be constructed. The algorithm may try different vectors v to reduce the matrix dimension, unless an optimal v is given deterministically.

Another merit of the incremental algorithm is that it produces optimal matrices for all cases where this is provably possible. Partition the n variables into r disjoint subsets, where n_k is the group's number of variables and $\sum_{i=1}^{r} n_k = n$. Polynomials which can be separately homogenized in every one of the r groups are called *multihomogeneous*. We focus on polynomials with identical supports, therefore with the same partition of variables and the same degree d_k in the kth variable subset; such a system is said to be of *type* $(n_1, \ldots, n_r; d_1, \ldots, d_r)$. All multihomogeneous systems which have $n_k = 1$ or $d_k = 1$ for every $k = 1, \ldots, r$ admit an optimal sparse resultant matrix, readily constructed by the incremental algorithm; see below for examples. In this case, a deterministic choice for v is possible [19]. For systems resembling this structure we use v obtained by randomly perturbing the vector specified as above. In all of these cases, we either know cardinalities $\#B_i$ or a good bound on each.

Example 6. A completely dense multihomogeneous polynomial of type $(2, 1; 2, 1)$ with variable sets $\{x_1, x_2\}, \{y_1\}$ is the following:

$$x_0^2 y_0 + x_0^2 y_1 + x_1^2 y_0 + x_1^2 y_1 + x_2^2 y_0 + x_2^2 y_1 + x_0 x_1 y_0 + x_0 x_1 y_1 + x_1 x_2 y_0 + x_1 x_2 y_1 + x_2 x_0 y_0 + x_2 x_0 y_1,$$

where x_0 and y_0 are the respective homogenization variables for the two subsets. A dense polynomial of type $(2, 1; 1, 1)$ with the same variable sets is:

$$x_0 y_0 + x_0 y_1 + x_1 y_0 + x_1 y_1 + x_2 y_0 + x_2 y_1.$$

Example 7. Consider a 3-polynomial system with 2 variables and Newton polytopes as shown in Fig. 1:

$$f_0 = c_{01} + c_{02}x_1x_2 + c_{03}x_1^2x_2 + c_{04}x_1,$$
$$f_1 = c_{11}x_2 + c_{12}x_1^2x_2^2 + c_{13}x_1^2x_2 + c_{14}x_1,$$
$$f_2 = c_{21} + c_{22}x_2 + c_{23}x_1x_2 + c_{24}x_1.$$

For each subsystem of two polynomials we compute $MV_{-0} = 4$, $MV_{-1} = 3$, $MV_{-2} = 4$ thus the total degree of the sparse resultant is 11. On the other hand, the classical resultant has total degree given by the sum of three twofold Bézout bounds, each being the product of two total degrees, namely $8+6+12 = 26$. The sparse resultant matrices constructed by the algorithms in [10,11] have respectively dimension 15 and 14, whereas the incremental algorithm of [19], with $v = (2, 11/10)$, yields a matrix of dimension 12.

The corresponding twofold Minkowski sums have the following interior point sets, including the respective v-distances:

$$(Q_{-0} \cap \mathbb{Z}^2) = \{(0,1;0.150), (1,0;0.100), (1,1;0.100), (1,2;0.091), (2,1;0.050), (2,2;0.050), (0,2;0), (0,3;0), (2,0;0), (2,3;0), (3,1;0), (3,2;0), (3,3;0)\},$$

$$(Q_{-1} \cap \mathbb{Z}^2) = \{(0,0;0.150), (1,0;0.100), (0,1;0.091), (1,1;0.091), (2,1;0.050), (1,2;0), (2,2;0), (2,0;0), (3,2;0), (3,1;0)\},$$

$$(Q_{-2} \cap \mathbb{Z}^2) = \{(0,1;0.182), (1,1;0.150), (1,0;0.111), (2,1;0.100), (2,2;0.091), (3,2;0.050), (1,2;0), (2,0;0), (3,1;0), (3,3;0), (4,2;0), (4,3;0)\}.$$

The incremental algorithm starts with point set (here are included the respective v-distances) $B_1 = \{(0,0;0.150),\ (1,0;0.100),\ (0,1;0.091)\}$. The algorithm has to increment the matrix so updates B_1 to $B_1 \cup \{(1,1;0.091)\}$. The exact sets $T_i \subset Q_{-i} \cap \mathbb{Z}^2$ depend on the particular algorithm and the amount of prior knowledge that can supplied in order to bound the point search; below are shown the T_i obtained when the sole requirement is a positive v-distance.

$$T_0 = \{(0,1;0.150), (1,0;0.100), (1,1;0.100), (1,2;0.091), (2,1;0.050), (2,2;0.050)\},$$
$$T_1 = \{(0,0;0.150), (1,0;0.100), (0,1;0.091), (1,1;0.091), (2,1;0.050)\},$$
$$T_2 = \{(0,1;0.182), (1,1;0.150), (1,0;0.111), (2,1;0.100), (2,2;0.091), (3,2;0.050)\}.$$

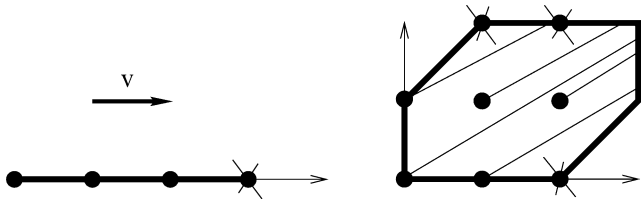

Fig. 2. To the left is the 1-dimensional projection of Q_{-1} and projection $\bar{v}$ of $v = (2, 11/10)$. Points with $\delta_{\bar{v}} = 0$ are crossed out; the algorithm recurses on the other 3 points. In two dimensions (on the right), the v-ray to the boundary is shown for points with $\delta_v(p) > 0$.

Fig. 2 anticipates our algorithm and shows how points are enumerated by lifting their projection each time to one dimension higher.

The incremental algorithm yields the following 13×12 matrix, from which it returns any maximal nonsingular minor as sparse resultant matrix. Columns are indexed, in this order, by points $(0,1)$, $(0,2)$, $(1,0)$, $(1,1)$, $(1,2)$, $(2,0)$, $(2,1)$, $(2,2)$, $(2,3)$, $(3,1)$, $(3,2)$, $(3,3) \in Q \cap \mathbb{Z}^2$. For each row is shown the monomial multiple of an input polynomial that fills in that row.

$$\begin{bmatrix}
c_{01} & 0 & 0 & c_{04} & c_{02} & 0 & 0 & c_{03} & 0 & 0 & 0 & 0 \\
0 & 0 & c_{01} & 0 & 0 & c_{04} & c_{02} & 0 & 0 & c_{03} & 0 & 0 \\
0 & 0 & 0 & c_{01} & 0 & 0 & c_{04} & c_{02} & 0 & 0 & c_{03} & 0 \\
0 & 0 & 0 & 0 & c_{01} & 0 & 0 & c_{04} & c_{02} & 0 & 0 & c_{03} \\
c_{11} & 0 & c_{14} & 0 & 0 & 0 & c_{13} & c_{12} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & c_{11} & 0 & c_{14} & 0 & 0 & 0 & c_{13} & c_{12} & 0 \\
0 & c_{11} & 0 & c_{14} & 0 & 0 & 0 & c_{13} & c_{12} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & c_{11} & 0 & c_{14} & 0 & 0 & 0 & c_{13} & c_{12} \\
c_{21} & c_{22} & 0 & c_{24} & c_{23} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & c_{21} & c_{22} & 0 & c_{24} & c_{23} & 0 & 0 & 0 & 0 \\
0 & 0 & c_{21} & c_{22} & 0 & c_{24} & c_{23} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & c_{21} & c_{22} & 0 & c_{24} & c_{23} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{21} & c_{22} & 0 & c_{24} & c_{23}
\end{bmatrix}
\begin{matrix}
x_2 f_0 \\ x_1 f_0 \\ x_1 x_2 f_0 \\ x_1 x_2^2 f_0 \\ f_1 \\ x_1 f_1 \\ x_2 f_1 \\ x_1 x_2 f_1 \\ x_2 f_2 \\ x_1 x_2 f_2 \\ x_1 f_2 \\ x_1^2 x_2 f_2 \\ x_1^2 x_2^2 f_2
\end{matrix}$$

A stand-alone `C` implementation of the incremental algorithm, together with a general polynomial system solver is publicly available; see Section 7. To illustrate the importance, from a complexity point of view, of computing the T_i we refer to a robot calibration problem, revisited in Section 7, with $n = 6$. In this problem, $Q_0 \neq Q_1 = Q_2 = \cdots = Q_6$ so only two Newton polytopes and 6-fold mixed volumes need be computed; this takes 23 seconds. $Q_{-1} = Q_{-2} = \cdots = Q_{-6}$ implies that only T_0, T_1 are needed, and their computation takes 46 seconds with Algorithm 2 before the very last improvement is applied; refer to Table 2. Constructing the matrix takes another 62 seconds, whereas solving by linear algebra operations takes between 11 and 39 seconds, depending on matrix compression and arithmetic precision. Another example from computer vision concerns a camera motion computation, given 5 point correspondences in 2 images. This problem has been accurately solved with the incremental matrix, with about half the running time spent in computing the point sets; see [19] and the references therein.

4. Point computation

In this section we concentrate on a single Minkowski sum of n polytopes, say $Q^{(n)} = Q_1 + \cdots + Q_n$. The straightforward approach of [19] shall enumerate integer lattice points $T \subset Q^{(n)} \cap \mathbb{Z}^n$ and compute their respective v-distances. Then, we present our branch-and-bound methods designed to yield an efficient algorithm in practice.

The plain version of the *Mayan pyramid* algorithm in [19] solved the plain version of Problem 1, i.e., without any bound on the output set's cardinality and only restricting the v-distance to be positive. It computes, at its kth step, the range of values of the kth coordinate for any point in T whose first $k-1$ coordinates are fixed, hence its name after the Mayan pyramids.

Algorithm 1 (Mayan Pyramid).
Input: Vertex sets of polytopes $Q_1, \ldots, Q_n \in \mathbb{R}^n$ and $v \in (\mathbb{R}^*)^n$.
Output: Points $T \subset Q^{(n)} \cap \mathbb{Z}^n$ together with their v-distances, such that the v-distances are all positive.
Steps:
(1) $T \leftarrow \emptyset$, $k \leftarrow 1$ and initialize $\bar{p}$ to the empty vector.
(2) Compute $mn, mx \in \mathbb{Z}$ which are, respectively, the minimum and maximum kth coordinates of any integer point in the projection of $Q^{(n)}$ to $\mathbb{R}^k$ whose first $k-1$ coordinates are specified by $\bar{p}$.
(3) If $k < n$, for each $s \in [mn, mx]$ do: let $\bar{p} \leftarrow (\bar{p}, s)$, $k \leftarrow k+1$. If $mn < s < mx$ then recurse at step (2). Otherwise, if $\delta_{\bar{v}}(\bar{p}) > 0$ then recurse at step (2).
(4) If $k = n$, then for each $s \in [mn, mx]$ do: set $p \leftarrow (\bar{p}, s)$ and compute $\delta_v(p)$. If $\delta_v(p) > 0$, then store p along with $\delta_v(p)$ into T.
(5) Sort all $p \in T$ according to $\delta_v(p)$.

It is possible to remove the recursion. Linear programming is used to compute mn, mx at step (1) above:

$$(\bar{p}, s) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \lambda_{ij} \bar{v}_{ij}, \tag{1}$$

with $\sum_{j=1}^{m_i} \lambda_{ij} = 1$, for $i = 1, \ldots, n$, $\forall \lambda_{ij} \geqslant 0$, where m_i is the cardinality of the vertex set of Q_i, the v_{ij} are its vertices, $\bar{v}_{ij}$ is the projection of v_{ij} to its first k coordinates, and $\bar{p} \in \mathbb{Z}^{k-1}$. The linear program's variables are the λ_{ij} and s. For the same constraints, mn is the ceiling of the minimum value of s, while mx is the floor of the maximum s. Notice that *no* integer programming is required. Of course, variable s can be avoided since it appears only in one constraint; this can be solved for s and can be used as the objective function to be minimized or maximized.

In $\mathbb{R}^k$, computing $\bar{v}$-distances is accomplished by *maximizing* $\sigma \in \mathbb{R} \geqslant 0$ subject to:

$$\bar{p} + \sigma \bar{v} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \lambda_{ij} \bar{v}_{ij}, \tag{2}$$

and $\sum_{j=1}^{m_i} \lambda_{ij} = 1$, $\lambda_{ij} \geqslant 0$, for $i = 1, \ldots, n$, where $\bar{p}$, $\bar{v}$ and $\bar{v}_{ij}$ are, respectively, projections of p, v, and v_{ij} to $\mathbb{Z}^k$, $(\mathbb{R}^*)^k$, and $\mathbb{Z}^k$, for $k = 1, \ldots, n$.

Algorithm 1 uses a restricted version of Proposition 10 in step (3) to avoid recursing at coordinate values mn, mx when $\delta_{\bar{v}}(\bar{p}) = 0$. In the rest of the section we derive certain properties of $\delta_v(\,)$ and describe the current version of the Mayan pyramid algorithm. The principal advantages of our approach, to be used in improving the algorithm, are:

- There is no need to construct explicitly the Minkowski sum $Q^{(n)}$.
- The algorithm starts by considering projections of the polytope, which allows for bounding the search space.
- It considers each point in some projection of $Q^{(n)}$ a constant number of times and for most points only once.
- It allows us to limit the number of extra points that shall not be output, when a cardinality bound is provided.
- It offers control over the direction of search in order to prune the search space.

Typically, a set T contains many more points than eventually needed in the matrix construction. To take advantage of a bound on the output cardinality and/or a bound on the v-distance, we wish to exploit

any properties of the latter. Observe that v-distance is not monotone, unlike inner product. However, it is *piecewise linear,* with "pieces" defined by the boundary facets. Let us consider the projection of $Q^{(n)}$ and v to lower-dimensional subspaces in order to restrict the search; the points and vector are respectively denoted by $\bar{p} \in \mathbb{Z}^k$ and $\bar{v}$. The basic property is that, for any $k \leqslant n$, $\delta_{\bar{v}}(\bar{p}) = 0$ if and only if $\bar{p}$ lies on a face of the projected polytope whose exterior normal has non-negative inner product with $\bar{v}$.

Proposition 8 (Concavity). *In one dimension, $\delta_{\bar{v}}(\cdot)$ is* monotone*, where $\bar{v} \in \mathbb{R}$. In dimension $k > 1$, $\delta_{\bar{v}}(\cdot)$ is* concave *within every one-dimensional subset with the first $k-1$ coordinates fixed, i.e., $\delta_{\bar{v}}(\bar{p}, s)$ is a concave function of s with fixed $\bar{p} \in \mathbb{Z}^{k-1}$ and $\bar{v} \in (\mathbb{R}^*)^k$.*

Proof. For $k = 1$, the first claim is obvious. Consider any two points in the 1-dimensional subset $p_i = (\bar{p}, s_i)$ with respective $\bar{v}$-distances denoted by σ_i. This implies that $q_i = p_i + \sigma_i \bar{v}$ are points in the (boundary of) projected polytope. For any two $\lambda_i \in [0, 1]$, such that $\lambda_1 + \lambda_2 = 1$, we have $(\lambda_1 p_1 + \lambda_2 p_2) + (\lambda_1 \sigma_1 + \lambda_2 \sigma_2)\bar{v} = \lambda_1 q_1 + \lambda_2 q_2$. Since the latter is a polytope point, $\delta_{\bar{v}}(\lambda_1 p_1 + \lambda_2 p_2) \geqslant \lambda_1 \sigma_1 + \lambda_2 \sigma_2$, which is tantamount to concavity. □

The proposition is illustrated in Fig. 2. It is applied in steps (3), (4) and (5) in Algorithm 2.

Corollary 9 (Zeros). *Let s be in the range of all possible $(k+1)$-st coordinates for points $(\bar{p}, s)$ with $\bar{p} \in \mathbb{Z}^k$ fixed; let $(\bar{v}, v_{k+1})$ be the projection of v in $(\mathbb{R}^*)^{k+1}$. Suppose there exist at least two distinct values of s. If $\delta_{(\bar{v}, v_{k+1})}(\bar{p}, s) = 0$ for all s then $\delta_{\bar{v}}(\bar{p}) = 0$. Otherwise, $\delta_{(\bar{v}, v_{k+1})}(\bar{p}, s) = 0$ is possible only at the two extremal values of s.*

Proof. The set of all points $(\bar{p}, s)$ in the projection of $Q^{(n)}$ forms a segment in $\mathbb{R}^{k+1}$. Under the fist hypothesis, all $(\bar{p}, s)$ lie on the same polytope face in $\mathbb{R}^{k+1}$. The exterior normal to this face is $(N, 0)$, where $N \in \mathbb{R}^k$, and has non-negative inner product $\langle (N, 0), (\bar{v}, v_{k+1}) \rangle$. Hence, the projection of this face to $\mathbb{R}^k$ is a face of the projected polytope with normal N, and contains $\bar{p}$. Since $\langle N, \bar{v} \rangle \geqslant 0$ the first statement is proven. Otherwise, let $[s_1, s_2]$ be the shortest segment such that $\delta_{(\bar{v}, v_{k+1})}(\bar{p}, s)$ vanishes only at its boundaries, and suppose s_1 is not extremal among the values of s for which $(\bar{p}, s)$ is in the projection of $Q^{(n)}$. By concavity, $\delta(\bar{p}, s) = 0$ for all s between an extremal point and s_1. Since we are in a convex polytope, all points $(\bar{p}, s)$ must lie on the boundary in the direction of $(\bar{v}, v_{k+1})$, thus contradicting the hypothesis, that s_1 bounds the shortest segment. Hence s_1 must be extremal, and similarly for s_2. □

This indicates to test positivity of $\bar{v}$-distance only at extremal points of 1-dimensional subsets. So when the bound on v- and $\bar{v}$-distances is trivial (very close or equal to zero) steps (4), (5) below can be improved.

Proposition 10 (Monotonicity). *Let $p \in Q^{(n)} \cap \mathbb{Z}^n$ and $\bar{v}$, $\bar{p}$ be the k-dimensional projections of vector v and point p, respectively. Then $\delta_v(p) \leqslant \delta_{\bar{v}}(\bar{p})$.*

Proof. Since $\delta_v(p)$ is maximum such that $p + \delta_v(p)v = \sum_{i,j} \lambda_{ij} v_{ij} \in Q^{(n)}$, it follows (by taking only k of the n constraints) that $\bar{p} + \delta_v(p)\bar{v} = \sum_{i,j} \lambda_{ij} \bar{v}_{ij}$ in the projection of $Q^{(n)}$ to the space of the first k coordinates. This describes a feasible solution to the linear program of maximizing $\delta_{\bar{v}}(\bar{p})$ in $\mathbb{R}^k$ and the claim follows. □

Now the improved algorithm can be stated. Just as before, linear optimization is used with $k < n$ coordinates to compute $\bar{v}$-distances. By Proposition 10, recursion is applied only for point projections whose $\bar{v}$-distance is strictly positive and larger than any given bound on v-distance, denoted β. Variable pr stores the last $\bar{v}$-distance of points when the last coordinate is varying and k coordinates are fixed. By exploiting concavity, established in Proposition 8, the $\bar{v}$-distance cannot increase once it becomes smaller than the previous value of $\bar{v}$-distance pr, supposing the latter is defined.

Algorithm 2 (Improved Mayan Pyramid).
Input: The vertices of Q_i, $v \in (\mathbb{R}^*)^n$, and bound $\beta > 0$ on v-distance; possibly, also a bound c on $\#T$.
Output: $T \subset Q^{(n)} \cap \mathbb{Z}^n$ and $\delta_v(p)$ for all $p \in T$, under the condition that $\delta_v(p) \geqslant \beta$.
Steps:
(1) $T \leftarrow \emptyset$, $k \leftarrow 1$ and initialize $\bar{p}$ to the empty vector.
(2) Compute mn, mx as the minimum and maximum kth coordinates of any integer point in the projection of $Q^{(n)}$ in $\mathbb{R}^k$ whose first $k-1$ coordinates are specified by $\bar{p}$.
(3) If $k = 1$ and $\bar{v} = v_1 > 0$ (resp. $v_1 < 0$) then for $s = mn, \ldots, mx$ ($s = mx, \ldots, mn$) and while $\delta_{\bar{v}}(s) \geqslant \beta$ do: $\bar{p} \leftarrow (s)$, $k \leftarrow 2$, then recurse at step (2).
(4) If $2 \leqslant k < n$, for $s = mn, \ldots, mx$, $\bar{p} \leftarrow (\bar{p}, s)$. If $\delta_{\bar{v}}(\bar{p}) \geqslant \beta$ then $k \leftarrow k + 1$ and recurse at step (2). Otherwise: if $s > mn$ and pr is defined and $\delta_{\bar{v}}(\bar{p}) < pr$, then terminate the iteration on s; if $s = mn$ or $\delta_{\bar{v}}(\bar{p}) \geqslant pr$, then set $pr \leftarrow \delta_{\bar{v}}(\bar{p})$ and continue the iteration on s.
(5) If $k = n$, for $s = mn, \ldots, mx$, set $p \leftarrow (\bar{p}, s)$. The iteration on s is terminated when $\delta_v(p) < \beta$ and at least one of the following two conditions is satisfied: $s > mn$ and pr is defined and $\delta_v(p) < pr$, or c is given and $\#T > \lambda c$ for some constant λ determined by fine-tuning. While the iteration continues, set $T \leftarrow T \cup \{p\}$ and $pr \leftarrow \delta_v(p)$.
(6) Sort all $p \in T$ according to $\delta_v(p)$ (keep only the c first points if c is given).

In the previous section, it was explained that in reality we need to compute $n+1$ sets $T_0, \ldots, T_n$. The minimum v-distance in every T_i will be roughly the same, so this yields a bound β on v-distance once T_0 has been computed. After computing each T_i, the program updates β. We derive successively better bounds even during the computation of T_0, by updating β whenever there are at least as many points with larger v-distance as a constant multiple λ of the upper cardinality bound c for T_0. To avoid an overly restrictive β, λ can be lowered if many points are found with smaller v-distance. Precisely how many, is a question of fine tuning discussed in Section 7. Similarly in step (5), points are not eliminated unless a certain number is already obtained.

By the discussion above, $\bar{v}$-distances, for any $k < n$, are needed only for comparison against the maximum available bound. Linear program (2) can then be simplified to a *feasibility* question of whether it is possible to find σ at least as large as the bound. The practical implications of this are discussed in Section 7. The linear optimization problems to find one of mn, mx for $k = n$ can be replaced by the computation of $\delta_v(\cdot)$. In the case of cyclic-7 (defined in Section 7) this eliminates about $1/8$ of the total number of optimization problems. This shortcut has not been fully implemented yet.

Step (5) of the improved algorithm can avoid all tests with $s = mx$ (or mn, depending on the direction of v) when $k = n$, which makes for about one third of all range computations. In that case, step (2) needs to calculate only one of mn, mx. Let us elaborate on this idea: for "box-like" polytopes close to hyper-rectangles, using the smallest *enclosing rectangle* accelerates the computation at those slices where all $\bar{v}$-distance checks are performed, i.e., for large k. On the other hand, the method fails on the cyclic n-roots

systems. Having optimized for *mn*, we add integer values and check every point until one has unbounded $\bar{v}$-distance. This will save computing *mx*, and is counter-efficient only for thin polytopes, close to the diagonal. This heuristic has significantly accelerated special classes of polytopes. More work is to be carried out before integrating this technique in the code, in order to automatically decide the polytopes for which this trick is acceptable.

Lastly, a related technique is described, which has not been tested experimentally. It relies on generalizing the definition of v to allow its projection $\bar{v}$ to be parallel to one of the axes, say the positive direction of the x_1-axis. This can be achieved by a (relatively cheap) rotation of the frame of reference.

Lemma 11. *Assume that q belongs to the k-dimensional projection of the polytope with $x_1 = a$, where $1 \leqslant k < n$ and $a \in \mathbb{Z}$. Then $\delta_{\bar{v}}(a+1, q) < \delta_{\bar{v}}(a, q)$, if (a, q) lies in the appropriate $(k+1)$-dimensional projection.*

Proof. The claim considers two points on the same line parallel to the x_1-axis and relies on the hypothesis that $\bar{v}$ is parallel to the x_1-axis. □

A useful consequence is that $\delta_{\bar{v}}(a+1, q) < \max\{\delta_{\bar{v}}(a, p)\}$ over all $p \in \mathbb{Z}^k$ in the k-dimensional projection with $x_1 = a$. This lets us compare $\bar{v}$-distances during a sweep of the polytope's projection in the direction of $\bar{v}$ and terminate this sweep when the above maximum falls below bound β.

5. Linear programming

This section studies the special structure of our linear programs in order to derive properties that lead to improvements in both asymptotic and practical complexity; see [8] for details and proofs of the transformations. Some of these observations have not been tested experimentally because their efficiency is strongly contingent upon the use of specially adapted code.

Observe that the equations $\lambda_{i1} + \cdots + \lambda_{im_i} = 1$ can be solved for λ_{i1}, thus eliminating one variable per summand polytope. This implies that the constraint becomes $\lambda_{i21} + \cdots + \lambda_{im_i} \leqslant 1$, hence the feasible polytope is (close to) the simple polytope defined as a product of simplices. More specifically, let us denote the product of convex polytopes A_1, A_2 by $A_1 \times A_2 = \{(a_1, a_2)\colon\ a_i \in A_i\}$. First note that the product of two simple polytopes is itself simple.

Proposition 12 [3, Lemma 1]. *The number of vertices and the total number of faces of any dimension in $A_1 \times A_2$ equals the* product *of the respective cardinalities in A_1 and A_2. The dimension and facet cardinality of $A_1 \times A_2$ equals the* sum *of the respective quantities of A_1, A_2. The same additive rule holds for the dimension of products of faces from A_1 and A_2. Moreover, all faces of the product polytope are obtained like this.*

The main goal in what follows is to exploit relationships between the various optimization problems in order to avoid some steps of the simplex algorithm. For instance, use an available basic feasible solution to avoid the first phase of the algorithm. One straightforward case is the linear program defined by (1). It is essentially used two times, with different objective functions. Hence, any (basic) feasible solution of the minimization problem is a (basic) feasible solution of the maximization problem.

When the solution of the minimization or maximization problem happens to be an integer, it defines a point $(\bar{p}, s)$ whose last coordinate equals the optimal value of s in (1). Therefore, the optimal values of the λ_{ij} provide a basic feasible solution for the maximization question of type (2) for computing the $\bar{v}$-distance of $(\bar{p}, s)$. Namely, these λ_{ij} correspond to the solution with $\sigma = 0$, provided that no bound $\beta > 0$ is imposed on σ. To see this, just observe that with $\sigma = 0$ the constraints of (1) and (2) are identical, with point $\bar{p}$ increasing in dimension by 1 between the two formulations.

To fully exploit the closely-knit interdependence of the successive linear programs of type (2), it is useful to consider their dual formulation. The primal problem, after solving for $\lambda_{i1},\ i = 1, \ldots, n$, as proposed above, gives the following; recall that $\bar{p}, \bar{v}_{ij} \in \mathbb{Z}^k$, $\bar{v} \in (\mathbb{R}^*)^k$, for $k \leqslant n$.

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=2}^{m_i} \lambda_{ij}(\bar{v}_{ijt} - \bar{v}_{ij1}) - \sigma\bar{v}_t = \bar{p}_t - \sum_{i=1}^{n} \bar{v}_{ij1}, && t = 1, \ldots, k,\\
&\sum_{j=2}^{m_i} \lambda_{ij} \leqslant 1, && i = 1, \ldots, n,\\
&\lambda_{ij} \geqslant 0, && i = 1, \ldots, n,\ j = 2, \ldots, m_i,\\
&\sigma \geqslant 0,\\
&\min -\sigma.
\end{aligned}
\tag{2'}
$$

The equivalent dual problem is derived line by line from the primal; the $y_1, \ldots, y_n$ and $z_1, \ldots, z_k$ are the new variables.

$$
\begin{aligned}
&y_t \text{ free}, && t = 1, \ldots, k,\\
&z_i \leqslant 0, && i = 1, \ldots, n,\\
&z_i + \sum_{t=1}^{k} y_t(\bar{v}_{ijt} - \bar{v}_{ij1}) \leqslant 0, && i = 1, \ldots, n,\ j = 2, \ldots, m_i,\\
&-\sum_{t=1}^{k} y_t\bar{v}_t \leqslant -1,\\
&\max \sum_{i=1}^{n} z_i + \sum_{t=1}^{k} y_t\left(\bar{p}_t - \sum_{i=1}^{n} \bar{v}_{ij1}\right).
\end{aligned}
$$

Computing $\bar{v}$-distances for different points $\bar{p} \in \mathbb{Z}^k$ implies changing the right-hand side of the equality constraints in the primal. In the dual, though, this changes only the maximization function. Hence, any (basic) feasible solution for one dual program remains (basic) feasible in subsequent $\bar{v}$-distance computations with the dual. This implies that, for given k, we have to execute the first phase of the (dual) simplex algorithm, namely the computation of a basic feasible solution, only once. Alternatively, the optimal solution of a primal problem on one point yields a basic feasible solution of the dual for any other point.

Consider now the computation of the $\bar{v}$-distance of all points in $\mathbb{Z}^{k+1}$ that have their first k coordinates equal to $\bar{p} \in \mathbb{Z}^k$, once the $\bar{v}$-distance of this latter point has been computed. Incrementing k means adding a new constraint to the primal program (2′). Equivalently, it creates a new free variable y_{k+1} in the dual, which appears in all constraints as well as the maximization function. Just as before, (basic) feasible solutions of the dual remain so when k is incremented, and the optimal solution of the primal gives a basic feasible solution of the dual for an augmented k.

An analogous primal-dual transformation is possible for program (1). In either case, working with the dual presents the advantage of reducing the number of variables from $O(nm)$ to $O(n)$, whereas the number of constraints is in both cases in $O(nm)$.

6. Asymptotic complexity

This section analyzes the worst-case asymptotic complexity of Algorithm 2 in terms of the sparsity parameters. Most bounds apply to Algorithm 1 too, since the improvements concern primarily the practical complexity and are hard to capture in an asymptotic analysis.

A direct application of the Aleksandrov–Fenchel inequality proves the following; see [17,38]:

Proposition 13. $(MV(Q_1,\ldots,Q_n))^n \geqslant (n!)^n \operatorname{vol}(Q_1)\cdots\operatorname{vol}(Q_n)$, *for any convex polytopes* $Q_i \in \mathbb{R}^n$.

Consider a family of $n+1$ polytopes Q_i and denote by Q_μ the polytope of minimum volume. The *system's scaling factor* s is the minimum real such that $s \geqslant 1$ and $Q_i + t_i \subset sQ_\mu$, for $i = 0,\ldots,n$ and some translation vectors $t_i \in \mathbb{R}^n$. We denote by e the basis of the natural logarithm.

Corollary 14. *Given a family of polytopes* $Q_i \subset \mathbb{R}^n$ *such that* $\operatorname{vol}(Q_i) > 0$ *for* $i = 0,\ldots,n$, *define* Q_μ *and the system's scaling factor* s *as above. Then* $\operatorname{vol}(Q_{-i}) = O(e^n s^n)\, MV(Q_0,\ldots,Q_{i-1},Q_{i+1},\ldots,Q_n)$.

This bound generalizes the case of identical polytopes in which $s = 1$. Moreover, it is asymptotically quite tight, except when $s \to \infty$ (hence the requirement of positive volumes).

To state our asymptotic bounds the following hypothesis is needed on the inputs, which is always satisfied in the examples we have considered.

Hypothesis 15. For any $k \in \{1,\ldots,n\}$ and $\bar{p}$ in the orthogonal projection of $Q^{(n)}$ into $\mathbb{Z}^{k-1}$, we assume that the number of points $\bar{p}$ for which $mn > mx$ at step (2) of Algorithm 2 is bounded above by a constant multiple of the total number of points $\bar{p}$ examined for that k.

This intuitively prohibits long and thin polytopes that contain a large number of integer points in their projections with respect to their n-dimensional euclidean volume. An immediate consequence is that the number of integer points inside any one of the Minkowski sum's orthogonal projections is asymptotically bounded by the sum's volume. The hypothesis can be explicitly checked during execution of the algorithm, and the number of times it fails can be reported in the end. Failure of the hypothesis is equivalent to $mn > mx$ for some integer point $\bar{p}$ in an orthogonal projection of the Minkowski sum.

Now let L be the maximum bit-size of any Newton polytope vertex coordinate. Consider any subsystem of n polytopes with $Q^{(n)}$ as their Minkowski sum and MV their mixed volume.

Theorem 16. *Let* m *bound the number of vertices per Newton polytope, for a system of* $n+1$ *polynomials. The bit complexity of the Mayan Pyramid algorithm to compute an integer point set* $T = Q^{(n)} \cap \mathbb{Z}^n$ *and the respective* v*-distances is* $O((\#T)n^{3.5}m^{1.5}L^2 \log L)$, *for any* $i = 0,\ldots,n$. *If* s *is the system's scaling factor, then the bit complexity is* $O(e^n s^n MV n^{3.5} m^{1.5} L^2 \log L)$.

Proof. In the dual formulations of the linear programming problems, the number of variables and constraints is $O(m)$ and $O(nm)$ respectively, and the bit size of the coefficients is L, so the bit complexity becomes $O(n^{2.5}m^{1.5}L^2 \log L)$ [47].

By Ehrart's famous result [16] which bounds asymptotically the cardinality of the integer point set by the euclidean volume (or alternative results from Section 2), in conjunction with Hypothesis 15, we can bound the number of integer points in every orthogonal projection by $O(\mathrm{vol}(Q^{(n)}))$. Thus the number of linear programs is bounded by $O(\mathrm{vol}(Q^{(n)})n)$, because at every dimension there is at most two optimization problems solved per point in the polytope's projection. Finally, apply Corollary 14. □

The number of integer points in a projection could be eventually refined by better bounding the volume of the orthogonal projection. For hypercubes, for instance, as well as similar polytopes, every time we project, the number of points is divided roughly by the edge length.

To simplify complexity bounds, we may ignore polylogarithmic factors in the arguments, which is denoted by $O^*(\cdot)$.

Corollary 17. *Let* $\deg R$ *denote the total degree of the input polynomials' sparse resultant. Suppose that the system's scaling factor* $s = O(1)$ *and the maximum bit size of any vertex is bounded by a polynomial in* n. *Then the total bit complexity of the Mayan pyramid algorithm for computing some sets* $T_0, \ldots, T_n$, $T_i \subset Q_{-i} \cap \mathbb{Z}^n$, *is* $O^*(e^n m^{1.5} \deg R)$.

Proof. Applying the previous theorem and summing over $i = 0, \ldots, n$, the total bit complexity is $\sum_{i=0}^{n} O^*(e^n MV_{-i} m^{1.5})$. Then use the fact that $\deg R = \sum_{i=0}^{n} MV_{-i}$, where $\deg R$ is the total degree of the sparse resultant. □

To model the situation when the Improved Mayan Pyramid algorithm is used, we may assume that the total number of points computed in any set T_i is a linear multiple in n of the number of points MV_{-i} needed for a matrix of minimum size. Then, the bit complexity for the ith point set would be $O(MV_{-i} n^{4.5} m^{1.5} L^2 \log L)$. The assumption on the output-sensitive behaviour of the Improved Mayan Pyramid algorithm is verified experimentally at Table 2. This proves the following asymptotic bound in terms of the main sparsity parameter, namely the total degree of the sparse resultant $\deg R$, which is the sum of all n-fold mixed volumes.

Proposition 18. *Let us suppose that Hypothesis 15 holds and that Algorithm 2 has to enumerate only* $O(nMV_{-i})$ *integer points in every set* T_i. *Then the total bit complexity of this algorithm is* $O^*(n^{4.5} m^{1.5} L^2 \deg R)$.

The point enumeration complexity dominates the computation of the *Newton polytope vertices*: for each polytope, we may test every support point by a linear program with at most m variables, $O(n + m)$ constraints. Assuming $n = O(m)$, the complexity is $O(m^3 L^2 \log L)$, where L is as above [47]. Hence the total bit complexity is in $O(nm^3 L^2 \log L)$.

Computing all n-fold *mixed volumes* is in $O^*(m^{2n+2})$, but in practice this cost is also largely dominated. Assuming that a constant number of vectors v is used and that the matrix dimension is a constant multiple of the optimal dimension, which is always the case in practice, then *matrix construction* has bit complexity $O^*(e^{3n}(\deg R)^3)$ [19]. These results provide a theoretic explanation of the bottleneck observed at point computation.

Let us estimate the complexity if Minkowski sum $Q^{(n)}$ were explicitly computed, represented by its vertices or by its facets. In the former case, the number of $Q^{(n)}$ vertices is in $O(m^{2(n-1)}n^{n-1})$ [28, Corollary 2.1.11], and this is the best possible bound. So, the complexity of computing a vertex representation of the Minkowski sum is exponential in n, irrespective of the output size; see also [28, Remark 2.1.3]. The number of facets in the Minkowski sum is in $O(g^n)$ [28, Theorem 2.1.10], where g denotes the number of nonparallel edges of the initial polytopes. Therefore, the complexity is again exponential in n, but will be higher than in the case of a vertex representation for most inputs. Hence, computing explicitly the Minkowski sum makes sense only if (almost) the entire point sets $Q_{-i} \cap \mathbb{Z}^n$ are needed and the vertex cardinality of $Q^{(n)}$ is small. See the next section for practical complexity.

7. Implementation and experiments

The implementation is in `C` and can be retrieved freely from

http://www-sop.inria.fr/galaad/logiciels/emiris/soft_geo.html.

This is research open-source software, using our implementation of the simplex method for linear programming based on the code from [42], and hence of limited efficiency. The main file of interest is `points.c`. Compilation is using the option `-DONLY_POINTS`, so that the sparse resultant matrix is not constructed. Input and output formats, as well as command-line options are explained in the `README` file. In the current version, point coordinates should be non-negative. The most important options include

- `-mc c`, with integer c providing a bound on the point set cardinalities,
- `-ms k`, with $k \in \{0, \ldots, n+1\}$ indicating the number of point sets already computed and stored in the appropriate file, $k = 0$ implying that all sets must be computed,
- `-mu b`, with rational b providing a bound on the v-distances.

We report on execution times on a SunUltra 1/170 workstation running SunOS 5.6, with a 167 MHz CPU and 64 MB of main memory. All running times are reported to the nearest integer number of user CPU seconds by `C` library function `getrusage`.

Let us list here the three main tuning parameters that may be adjusted before compilation; they are introduced in Algorithm 2 or the discussion that follows it.

- A choice of k for which $\delta_{\tilde{v}}$ is computed (in order to be compared against the available bound $\beta > 0$). The associated cost is analyzed in Table 6; in the present examples it pays off to apply this test for all k.
- A possible choice of when to update bound β: hasty updates eliminate too many points, whereas delayed updates neutralize this pruning test. This update is not included in the description of Algorithm 2 but has been implemented.
- A choice of λ in step (5) of Algorithm 2 such that λc indicates the number of points that must be collected in some T_i before the algorithm rejects any candidate (projection) point, where c denotes the target cardinality of T_i. λ should be sufficiently larger than 1 (typically between 2 and 4) so that it does not reject projected points which may eventually yield a large v-distance.

We concentrate on 3 classes of systems: multihomogeneous systems where all polytopes are identical (identified by their type) as described in Section 3, the standard algebraic benchmark of cyclic n-roots, and a system encountered in robot calibration with $n = 6$ and $Q_0 \neq Q_1 = \cdots = Q_6$, discussed at the end of Section 3. The cyclic N-roots family is now defined, which is encountered in Fourier analysis. It is

Table 1
CPU secs for computing the facet structure of the Minkowski sum and all interior points by `porta`

System	n	#facets	$\#T$	Time for $Q^{(n)}$	Time for T	Total
cyclic-6	4	29	215	1	0	1
cyclic-7	5	99	1400	93	1	94
(2, 1, 1; 1, 2, 2)	4	7	1215	2	0	2

defined by $f_k = \sum_{i=1}^{N} \prod_{j=i}^{i+k-1} x_j$, where $x_{N+t} = x_t$, for $k = 1, \ldots, N-1$, and $f_N = x_1 x_2 \cdots x_N - 1$. This is a square system of dimension N. We use an equivalent formulation that decreases the dimension by setting $y_i = x_i / x_N$ and $y_N = 1$ in the first $N-1$ equations, then employ $x_N^{-n} = y_1 \cdots y_{N-1}$ to solve for the Nth variable. The new system contains $N-1$ polynomials in $N-1$ variables:

$$f_k = \sum_{i=1}^{N} \prod_{j=i}^{i+k-1} y_j, \quad y_N = 1 \quad \text{and} \quad y_{N+t} = y_t, \quad k = 1, \ldots, N-1.$$

Here we consider the overconstrained system resulting from "hiding" y_1 in the coefficient field. Setting $n = N-2$ yields a system of $n+1$ equations in n variables. This is a particularly sparse system since every Newton polytope has zero n-dimensional volume.

We compare with the approach that first computes a facet representation of Minkowski sum $Q^{(n)}$ (from all *distinct* sums of n vertices) and then computes *all* integer points in $T = Q^{(n)} \cap \mathbb{Z}^n$. For this, we used `porta, v.1.3.2` [13] which offers both functionalities, though it requires a facet representation of $Q^{(n)}$ in order to compute the integer points in its closed interior. Of course, the facet representation of the convex hull can be obtained by any arbitrary-dimension convex hull software, including `cdd`, `lrs`, `porta` or `qhull`. Running times in Table 1 ignore the time to compute the points defining $Q^{(n)}$, the v-distances, and the sorting phase. The table reports the number of facets of $Q^{(n)}$, $\#T$, and the times for computing $Q^{(n)}$ and T respectively, on 3 instances. As expected, computing the convex hull is very expensive and clearly dominates point enumeration; it also makes the overall timings much larger as compared to those of our algorithm, as reported in the last two columns of Table 2. This is, clearly, not a comment about `porta` but only about the method that requires the explicit construction of the Minkowski sum. One thing that has not been exploited is that the point sums defining $Q^{(n)}$ form a "dense" subset of T, often of the same cardinality (which is the case in the examples of Table 1).

To compare with the algorithm of [19], we use an improved version of that code which implements Algorithm 1. Timings in CPU seconds are reported in column "Alg. 1" of Table 2. Our current implementation uses all pruning techniques incorporated in Algorithm 2 and yields the CPU timings in the corresponding column. All computed $T_i \subset Q_{-i} \cap \mathbb{Z}^n$ have a prescribed bound on the number of points (denoted c above) and reported in the fourth column of the same table. The sixth column ($\#B_i$) shows the cardinalities of the point subsets actually used in the matrix. The first input set is drawn from the cyclic N-roots family. For cyclic-8, the matrix takes rather long to construct, hence the cardinalities of B_i are not reported. For the multihomogeneous systems, which comprise the last group of examples and where there is a single distinct set $T_0 = B_0$, both programs exploit the fact that the cardinality of T_0 is known (and equal to the n-fold mixed volume).

Table 2
CPU seconds of original algorithm and two versions of the current algorithm

System	n	MV_{-i}	Compute	$\#T_i$	$\#B_i$	Alg. 1	Alg. 2	Semi
cyclic-6	4	12,14	$T_0, \ldots, T_4$	30	18–30	1	1	1
cyclic-7	5	43–47	$T_0, \ldots, T_5$	250	164–198	10	6	6
cyclic-8	6	128–152	$T_0, \ldots, T_6$	500	–	> 180	60	60
robotics	6	10,20	T_0, T_1	60	10,50	65	46	3
(2, 1, 1; 1, 2, 2)	4	48	T_0	48	48	0	0	0
(1, 1, 1, 1; 2, 2, 1, 1)	4	96	T_0	96	96	3	1	0
(1, 1, 1, 1; 3, 3, 1, 1)	4	216	T_0	216	216	5	3	0
(1, 1, 1, 1; 3, 3, 2, 1)	4	432	T_0	432	432	11	6	1
(1, 1, 1, 1; 3, 3, 3, 1)	4	648	T_0	648	648	16	12	2

Table 3
CPU seconds on the cyclic-N family ($N = n + 2$)

System	n	MV_{-i}	$\deg R$	[sec]
cyclic-4	2	2	6	0.017
cyclic-5	3	5	20	0.047
cyclic-6	4	12,14	66	0.517
cyclic-7	5	43–47	270	5.988
cyclic-8	6	128–152	948	59.967

A class of special interest are *semi-mixed* systems, with each Q_{-i} defined by k summand polytopes $S_1, \ldots, S_k$, where $k < n$ is the number of *distinct* Q_j, $j \neq i$. Of course, if Q_j is repeated r times in the Minkowski sum, then the corresponding $S_j = rQ_j$, and replaces all r Newton polytopes equal to Q_j. This drastically reduces the number of variables in the linear programs at a small cost, because the program has already computed the classification into distinct Newton polytopes. The running times resulting from exploiting identical polytopes (i.e., semi-mixed polytope sets) are reported in the last column of Table 2.

Table 3 looks into more detail in the running times of the cyclic family, each taken as the average of at least 3 runs. This will serve to test the assumptions used in deriving the output-sensitive asymptotic bounds in terms of the sparsity parameters in the previous section, in particular Proposition 18. The table includes the parameters needed for this proposition's bound, which becomes $O(n^6 \deg R)$ for this family, since m and n never differ by more than 1 and the input point coordinates have constant size. The logarithm of base 10 of function $n^6 \deg R$ is plotted in Fig. 3 with a solid line. The dashed line is the same logarithm of the running times from Table 3 multiplied by 10 so that the ranges of the two functions are close to each other. We see that in this small sample the behaviour of the practical complexity is indeed exponential in n and grows at a rate bounded by that of the asymptotic bound we have derived.

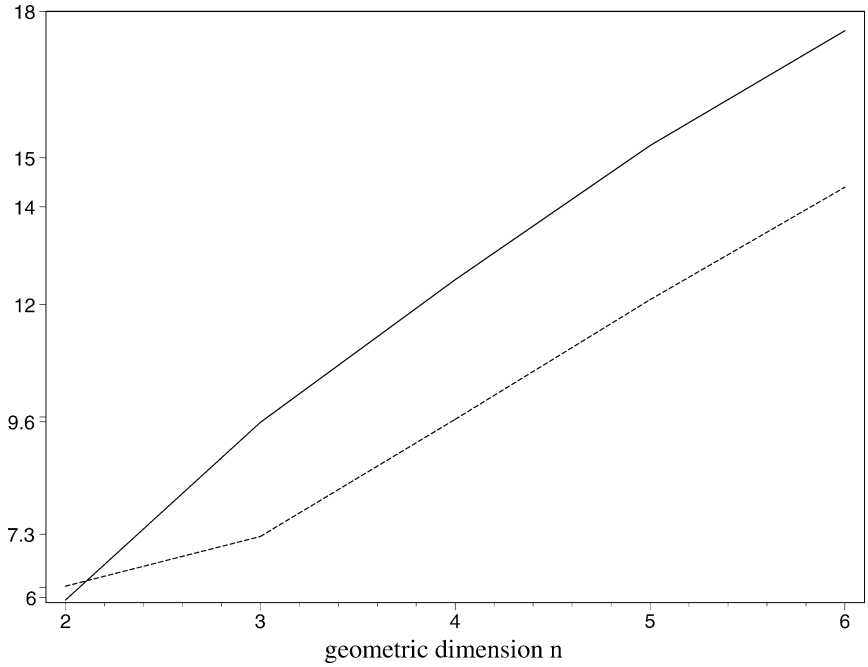

Fig. 3. Logarithms of the asymptotic bound function of Proposition 18 (solid line) and the running times for the cyclic-N family, $N = n + 2$ (dashed line), from Table 3.

Table 4
CPU seconds on multihomogeneous systems with fixed $n = 4$ and m polytope vertices

System	m	MV_{-i}	deg R	[sec]
(2, 1, 1; 1, 2, 2)	27	48	240	0.4
(1, 1, 1, 1; 2, 2, 1, 1)	36	96	480	1
(1, 1, 1, 1; 3, 3, 1, 1)	64	216	1080	3
(1, 1, 1, 1; 3, 3, 2, 1)	96	432	2160	6
(1, 1, 1, 1; 3, 3, 3, 1)	128	648	3240	12

Table 4 considers the running times of the multihomogeneous systems examined above, as an additional test of the assumptions used in deriving the asymptotic bounds. The bound of Proposition 18 becomes $O(m^{1.5} \deg R)$ for this family, since $n = 4$ and L is constant; the function $m^{1.5} \deg R$ divided by a constant power of 10 is plotted in Fig. 4 with a dashed line. The solid line represents the running times from Table 4, multiplied by 10 to make the two ranges approach each other. Despite the small size of our sample, the growth rate of the practical complexity remains below that of our asymptotic bound function.

Testing for feasibility instead of optimizing linear program (2), as explained after Algorithm 2, does not reduce significantly the running time even if, for certain inputs like the cyclic systems, this

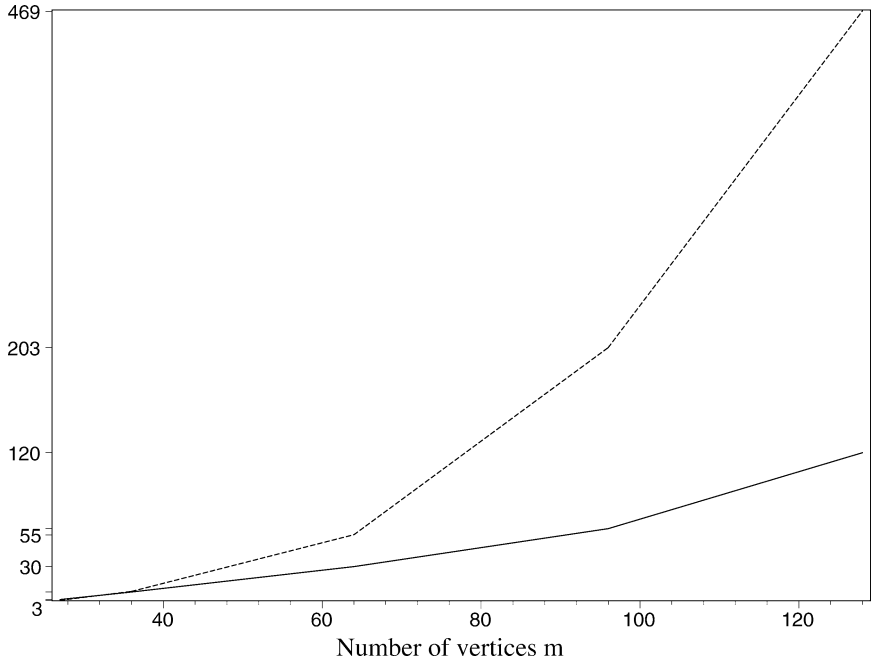

Fig. 4. The asymptotic bound function of Proposition 18 (dashed line) and the running times for the multihomogeneous systems (solid line) from Table 4.

Table 5
Improved performance by bounding v-distance on system $(1, 1, 1, 1; 2, 2, 1, 1)$

β	#candidates	total#LPs	[sec]
$\varepsilon \to 0^+$	1141	3018	3
dynamic	466	1314	1

represents about $1/3$ of the linear optimization problems. In terms of worst-case complexity, testing for feasibility runs in a constant fraction of the time needed for optimizing, especially in what concerns our implementation of the simplex method. This agrees with theory, which states that finding the feasible solution is roughly half the overall complexity of simplex-based methods. Specialized code could be used to decide this question faster; one possibility is to use `cdd+` with option `find_interior` [22].

The significance of tight bounds on the v-distance is illustrated on Table 5 for system $(1, 1, 1, 1; 2, 2, 1, 1)$, for which $\#B_i = 96$. The first row of Table 5 has a trivial β, whereas a dynamic β eliminates 289 candidate *projected* points for which $\delta_{\bar{v}}(\bar{p}) < \beta$. For subsequent point sets β is seldom lowered, even when the Q_{-i} are quite different, as in cyclic-7. The columns of Table 5 report, respectively, the number of candidate points, of linear programs, and running time.

Application of such bounding rules is not a boon, because of the cost associated to the computation of $\delta_v(p)$, which can become the dominating source of practical complexity, so careful tuning is in general

Table 6
Overall fraction of tableau construction and linear programming cost. Cost of computing v, $\bar{v}$-distances vs. computing mn, mx

System	Tableau	LP	$\delta_v, \delta_{\bar{v}}$	mn, mx
cyclic-6	12%	87%	57	43
cyclic-7	12%	87%	59	41
cyclic-8	4.8%	95%	58	42
(2, 1, 1; 1, 2, 2)	13%	81%	78	22
(1, 1, 1, 1; 3, 3, 3, 1)	11%	85%	70	30

required. Table 6 shows the percentage of overall time spent on building the linear programming tableaus and on linear programming itself. We note that sorting the integer points is largely dominated as is the time to compute the Newton polytopes: at most 0.5% and 3.1%, respectively, on the systems of Table 6. Moreover, the table shows the breakdown of the total linear programming costs for computing v- or $\bar{v}$-distances as opposed to computing mn, mx. Nonetheless, in both families of examples (cyclic and multihomogeneous), the checks on $\delta_{\bar{v}}(\bar{p})$ helped reduce the number of candidate points to such a degree that payed off for these checks' cost.

The remainder of this section focuses on linear programming and discusses our experiments with different software. All timings reported so far are based on our implementation of the simplex method based on the code from [42]. Here we consider the linear programming capabilities of `lrs` and `cdd+` and find out that they do not offer any speedup. Both software has been tested on the particular subproblem of computing v- and $\bar{v}$-distances. Our code generates hyperplane-represented linear programs using rationals (in order to express vector v or $\bar{v}$).

We used version `4.0` of `lrs` [2] which provides two main programs: the faster `lrs1` that runs with fixed-length long integers but provides no overflow checking, and `glrs` using the arbitrary-precision integers of the `GNU MP 2.0.2` library. Both implement the reverse search paradigm. Certain options are provided in order to better specify the number of digits used and the cache size; namely `digits` was set to a value between 20 and 32, whereas a typical value for `cache` was 500. Moreover option `lponly` was set, which accelerated execution considerably, and `linearity` was used to specify those constraints that were exact equalities. `lrs1` was not sufficiently accurate for our problems. On the other hand `glrs`, was definitely slower on the cyclic-8 $\bar{v}$-distance computations. The main reason is that it performs exact arithmetic over long integers. A secondary reason is that the positivity of the variables, implied by the simplex algorithm, must be explicitly stated thus increasing the number of constraints considerably. In particular, calling the stand-alone `glrs` program with file inputs ran at least 100 times slower.

Of the `C++` implementation of the double description method [22], we used version `0.76` of `cddf+` which runs on floating-point arithmetic, with options `lponly`, `stdout_off`, `dynout_off`, but no scaling of the input nor any kind of `linearity` option. More importantly, we reduced space allocation to arrays of at most 200 elements, then tried the code on cyclic-8. `cddf+` was able to compute all v- and $\bar{v}$-distances and produced exactly the same integer point set, though the distances differed from those of our program in their third significant digit (which did not affect the points' ordering).

We experimented with options `dual-simplex` (the default) and `criss-cross`, which calls upon the Terlaky–Wang method. For each algorithm, we tried 6 options for sorting the constraints: `lexmin` (the default), `lexmax`, `mincutoff`, `maxcutoff`, `maxindex` and `random`. The dual simplex method was faster for every ordering, whereas we could not observe significant differences between the orderings; a slight advantage should be recognized for `maxindex` and `random`. Yet, `cddf+` was about 2 times slower than the simplex code we have been using in calculating the v- and $\bar{v}$-distances for cyclic-8; overall execution was slowed down to about 117 seconds from the 60 seconds of Table 2.

8. Further work

Most of the aforementioned methods for computing the Minkowski sum in low dimensions exploit the interplay between convolution and Minkowski addition. The former provides an economical representation of the sum and might thus be exploited for the problem at hand [5,43].

Efficient linear programming software, adapted to the type of our problems may dramatically accelerate the program: first, there is a large difference between the number of variables to that of constraints. Second, the input is very sparse and the current code that fills in a dense matrix spends considerable time in creating the tableaus (see Table 6). Furthermore, in computing intervals $[mn, mx]$, the same set of constraints is processed with the same objective function, once to minimize it and once to maximize it. In addition, successive linear programs are very similar, since only a few coordinates in the equations change or a large subset of common constraints is common to several of them. In both cases, dimension is an input variable, which rules out direct use of the algorithms conceived under the hypothesis of fixed dimension, discussed in Section 2.

It is conceivable to compute mn, mx in step (2) of Algorithm 2 so that the new point $\bar{p}$ has $\delta_{\bar{v}}(\bar{p}) \geqslant \beta$, for $k > 1$. This eliminates explicit tests on $\delta_{\bar{v}}(\bar{p})$ but has not been experimentally validated. Implementation improvements are possible to reduce the most critical parameter of complexity, namely the number of variables in linear programming. For instance, projections of the Minkowski sum are now defined by the projections of all the vertices in the summands, which can have a significantly higher cardinality than if we first projected the vertices and computed their convex hull.

A related combinatorial question in sparse elimination concerns the modelling of the algebraic system. Given a system of (sparse) polynomials, there is no automatic procedure to decide whether it is multihomogeneous or even close to such. An interesting open problem is to devise a combinatorial algorithm that will determine the optimal multihomogeneous structure valid for the given system, namely to find the number of variable subsets r and the type of a system whose single Newton polytope includes all given (possibly different) supports. Optimality is equivalent to minimizing the number of points that may have to be added to the supports in order to fill in the multihomogeneous Newton polytope. A simpler measure would be to minimize the latter's volume. This problem is NP-hard [20]. Decomposing an integral polytope into a sum of polytopes is actually known to be NP-complete, even for polygons [24].

Acknowledgements

With acknowledgement to the France/Hong-Kong Joint Research Scheme PROCORE for its financial support.

References

[1] I. Adler, R. Shamir, A randomized scheme for speeding up algorithms for linear and convex programming with high constraints-to-variables ratio, Math. Programming 61 (1993) 39–52.

[2] D. Avis, User's guide for lrs—version 4.0, 2000, http://www.lab2.kuis.kyoto-u.ac.jp/~avis.

[3] D. Avis, D. Bremner, R. Seidel, How good are convex hull algorithms?, Computational Geometry Theory and Applications 7 (1997) 265–301.

[4] A.I. Barvinok, J. Pommersheim, An algorithmic theory of lattice points in polyhedra, in: L.J. Billera, A. Björner, C. Greene, R. Simion, R.P. Stanley (Eds.), New Perspectives in Algebraic Combinatorics, MSRI Publications, Vol. 38, Cambridge Univ. Press, Cambridge, UK, 1999, pp. 91–147.

[5] J. Basch, L.J. Guibas, G.D. Ramkumar, L. Ramshaw, Polyhedral tracings and their convolution, in: J.-P. Laumond, M. Overmars (Eds.), Algorithms for Robotic Motion and Manipulation, A.K. Peters, Wellesley, MA, 1996.

[6] D.N. Bernstein, The number of roots of a system of equations, Funct. Anal. Appl. 9 (2) (1975) 183–185. Translated from Funktsional'nyi Analiz i Ego Prilozheniya 9 (3) (1975) 1–4.

[7] D.N. Bernstein, The number of integral points in integral polyhedra, Funct. Anal. Appl. 10 (1976) 223–224. Transl. from Funktsional'nyi Analiz i Ego Prilozheniya 10 (3) (1976) 72–73.

[8] D. Bertsimas, J. Tsitsiklis, Introduction to Linear Optimization, Series in Optimization and Neural Computation, Athena Scientific, Belmont, MA, 1997.

[9] J.-D. Boissonnat, E. de Lange, M. Teillaud, Slicing Minkowski sums for satellite antenna layout, Computer-Aided Design 30 (1998) 255–265.

[10] J. Canny, P. Pedersen, An algorithm for the Newton resultant, Technical Report 1394, Comp. Science Dept., Cornell University, Ithaca, NY, 1993.

[11] J.F. Canny, I.Z. Emiris, A subdivision-based algorithm for the sparse resultant, J. ACM 47 (3) (2000) 417–451.

[12] T. Chan, Fixed-dimensional linear programming queries made easy, in: Proc. 12th Annual ACM Symp. on Computational Geometry, Philadelphia, PA, 1996, pp. 284–290.

[13] T. Christof, A. Loebel, M. Stoer, PORTA, version 1.3, University of Heidelberg and ZIB Berlin, 1999, http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/PORTA.

[14] K.L. Clarkson, Las Vegas algorithms for linear and integer programming when the dimension is small, J. ACM 42 (2) (1995) 488–499.

[15] D. Cox, J. Little, D. O'Shea, Using Algebraic Geometry, Graduate Texts in Mathematics, Vol. 185, Springer-Verlag, New York, 1998.

[16] E. Ehrart, Sur un problème de géométrie diophantienne, I. Polyèdres et réseaux, J. Reine Angew. Math. 226 (1967) 1–29.

[17] I.Z. Emiris, On the complexity of sparse elimination, J. Complexity 12 (1996) 134–166.

[18] I.Z. Emiris, Computing integer points in Minkowski sums, in: Proc. 16th Annual ACM Symp. on Computational Geometry, Hong Kong, 2000, pp. 29–36.

[19] I.Z. Emiris, J.F. Canny, Efficient incremental algorithms for the sparse resultant and the mixed volume, J. Symbolic Comput. 20 (2) (1995) 117–149.

[20] I.Z. Emiris, M. Rojas, Sparse results in sparse elimination, Draft, 2000.

[21] D. Eppstein, Zonohedra and zonotopes, Mathematica in Education and Research 5 (4) (1996) 15–21.

[22] K. Fukuda, cdd/cdd+ reference manual, version 0.61/0.76, EPF Lausanne, Switzerland, 1999, http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html.

[23] K. Fukuda, H. Brönnimann, Personal communication, 1999.

[24] S. Gao, A.G.B. Lauder, Decomposition of polytopes and polynomials, Discret. Comput. Geometry 26 (1) (2001) 89–104.

[25] B. Gärtner, E. Welzl, Random sampling in geometric optimization: New insights and applications, in: Proc. 16th Annual ACM Symp. on Computational Geometry, Hong Kong, 2000, pp. 91–99.

[26] M. Giusti, J. Heintz, J.E. Morais, J. Morgenstern, L.M. Pardo, Straight-line programs in geometric elimination theory, J. Pure Appl. Algebra 124 (1998) 101–146.
[27] P. Gritzmann, V. Klee, On the complexity of some basic problems in computational convexity II: Volume and mixed volumes, in: T. Bisztriczky, P. McMullen, R. Schneider, A. Ivic Weiss (Eds.), Polytopes: Abstract, Convex and Computational, Kluwer, Boston, MA, 1994, pp. 373–466.
[28] P. Gritzmann, B. Sturmfels, Minkowski addition of polytopes: computational complexity and applications to Groebner bases, SIAM J. Disc. Math. 6 (2) (1993) 246–269.
[29] P. Gritzmann, J.M. Wills, Lattice points, in: P.M. Gruber, J.M. Wills (Eds.), Handbook for Convex Geometry, Vol. B, North Holland, Amsterdam, 1993.
[30] M. Grötschel, L. Lovász, A. Schrijver, Geometric Algorithms and Combinatorial Optimization, 2nd edn., Springer-Verlag, Berlin, 1993.
[31] B. Grünbaum, Convex Polytopes, Wiley-Interscience, New York, 1967.
[32] L.J. Guibas, R. Seidel, Computing convolutions by reciprocal search, in: Proc. 2nd Annual ACM Symp. on Computational Geometry, Yorktown, Heights, NY, 1986, pp. 90–99.
[33] B. Huber, B. Sturmfels, Bernstein's theorem in affine space, Discr. Comput. Geom. 17 (2) (1997) 137–142.
[34] A. Kaul, M.A. O'Connor, V. Srinivasan, Computing Minkowski sums of regular polygons, in: Proc. 3rd Canadian Conf. on Computational Geometry, Vancouver, BC, 1991, pp. 74–77.
[35] A.G. Khovanskii, Newton polyhedra and the genus of complete intersections, Funktsional'nyi Analiz i Ego Prilozheniya 12 (1) (1978) 51–61.
[36] A.G. Khovanskii, Fewnomials, AMS Press, Providence, RI, 1991.
[37] A.G. Kushnirenko, The Newton polyhedron and the number of solutions of a system of k equations in k unknowns, Uspekhi Mat. Nauk. 30 (1975) 266–267.
[38] E. Lutwak, Volume of mixed bodies, Trans. AMS 294 (2) (1986) 487–500.
[39] J. Matoušek, Linear optimization queries, J. Algorithms 14 (1993) 432–448.
[40] E. Mayr, A. Meyer, The complexity of the word problem for commutative semigroups and polynomial ideals, Adv. Math. 46 (1982) 305–329.
[41] Optimization Technology Center, Linear programming frequently asked questions, Northwestern University and Argonne National Laboratory, 2000, http://www.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html.
[42] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing, 2nd edn., Cambridge University Press, Cambridge, 1992.
[43] G.D. Ramkumar, An algorithm to compute the Minkowski sum outer-face of two simple polygons, in: Proc. 12th Annual ACM Symp. on Computational Geometry, Philadelphia, PA, 1996, pp. 234–241.
[44] E.A. Ramos, Linear programming queries revisited, in: Proc. 16th Annual ACM Symp. on Computational Geometry, Hong Kong, 2000, pp. 176–181.
[45] J.M. Rojas, A convex geometric approach to counting the roots of a polynomial system, Theor. Comput. Sci. 133 (1) (1994) 105–140.
[46] S.S. Skiena, The Algorithm Design Manual, Springer-Verlag, Berlin, 1997.
[47] P. Vaidya, An algorithm for linear programming which requires $O((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations, Math. Programming 41 (1990) 175–201.