Exact Volume Computation for Polytopes: A Practical Study *

Benno Büeler,† Andreas Enge,†‡ and Komei Fukuda†§

January 12, 1998; revised August 7, 1998

Abstract

We study several known volume computation algorithms for convex d-polytopes by classifying them into two classes, triangulation methods and signed-decomposition methods. By incorporating the detection of simplicial faces and a storing/reusing scheme for face volumes we propose practical and theoretical improvements for two of the algorithms. Finally we present a hybrid method combining advantages from the two algorithmic classes. The behaviour of the algorithms is theoretically analysed for hypercubes and practically tested on a wide range of polytopes, where the new hybrid method proves to be superior.

1 Introduction

A *convex polytope* P is the convex hull $\mathrm{conv}(V)$ of a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of points in $\mathbb{R}^d$. Equivalently, it is a bounded subset of $\mathbb{R}^d$ which is the intersection of a finite set of half spaces. We shall omit 'convex' since we only deal with such polytopes, and use *d-polytope* to mean a d-dimensional polytope. When $P = \mathrm{conv}(V)$, V is called a *vertex representation* or simply *$\mathcal{V}$–representation* of P. When $P = \{x | A\, x \leq b\}$ for some $m \times d$ real matrix A and m-vector b, the pair (A, b) is called a *halfspace representation* or simply *$\mathcal{H}$–representation* of P. In this paper we treat the *volume computation problem* as to compute the volume $\mathrm{Vol}(P)$ of a polytope P given by both $\mathcal{V}$– and $\mathcal{H}$–representations. It seems that the theoretical complexity of this problem is unknown and not easy to evaluate, although the same problem when only one (either $\mathcal{V}$– or $\mathcal{H}$–) representation is provided is known to be #P-hard, see [5]. Main reasons for considering this problem setting are (1) the volume computation appears to be already hard with two representations given, (2) the transformation between the two representations is itself a hard fundamental problem which has been studied both theoretically and practically [6, 10, 1, 3, 9], and (3) many different algorithms, some requiring only one and some requiring both representations, can be compared on the same ground in our setting.

The relevance of volume computation is nicely demonstrated in the excellent survey [11], where also several basic approaches for volume computation are discussed in depth. In this paper we consider five methods and classify them into two groups. Triangulation methods on the one hand decompose the polytope into simplices for which the volume is easily computed and summed up; members of this group are the boundary triangulation, Delaunay triangulation and Cohen & Hickey's triangulation [4]. Signed decomposition methods, on the other hand, decompose a given polytope into signed simplices such that the signed sum of their volumes is the volume of the polytope; in this group we consider

*This research was partially supported by the ETHZ-EPFL joint research project on optimization and geometric computation, Switzerland. See http://www.ifor.math.ethz.ch/ifor/staff/fukuda/OGC_home/OGC.html.

†Institute for Operations Research, Swiss Federal Institute of Technology, CH-8092 Zurich, Switzerland.

‡Lehrstuhl für Diskrete Mathematik, Optimierung und Operations Research, Institut für Mathematik, Universität Augsburg, Deutschland.

§Department of Mathematics, Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland.

Lasserre's [13] and Lawrence's [15] methods. As explained in Section 2, these methods are in a sense dual to triangulation methods. A basic algorithm excluded from our investigations is the incremental construction of a triangulation using the beneath-beyond method [16, 7] for convex hull computation (see also [11, Section 4]). Concerning random approximation algorithms (see [12]) we know so far of no implementation and have therefore ignored this highly interesting new approach.

During an earlier stage of our study, we observed that no method described in the literature works efficiently on a wide range of polytopes. In fact, each of the investigated methods is hopelessly impractical for either the class of simple polytopes or the class of simplicial polytopes. For example, all triangulation methods work poorly for simple polytopes, and Lawrence's signed decomposition method crawls for simplicial polytopes. This is not surprising, since the sizes of a triangulation and a signed decomposition of the same polytope can be drastically different, see Section 2.4. This observation was our major motivation to revise some of the known methods so that they behave more uniformly and efficiently for a broader range of polytopes. In particular, in Lasserre's method we found a significant potential for improvements. These include storing and reusing intermediate results (in this case volumes of some polytope faces) and the efficient detection of empty faces. Cohen & Hickey's triangulation profits considerably from some more obvious improvements. Our modifications shall be related in detail in Section 4. Their success incited us to design a hybrid algorithm, combining the strong sides of both Cohen & Hickey's and Lasserre's methods. This new algorithm has an overall convincing performance, to which the laurels can be awarded in our competition.

While experiments showed differences among the algorithms in CPU-time and memory up to several orders of magnitude, the theoretical analysis is difficult. One major source is related to the difficulty of classifying polytopes, for example by a satisfactory measure for 'how simple' or 'how simplicial' a polytope is. Another difficulty is introduced by the recursive or iterative structure of the algorithms; it is usually not only the structure of the original polytope, but of all intermediate polytopes generated in the solution process which determine the complexity. Despite these difficulties we give complexity estimates for some concrete algorithms and polytopes accepting their limited relevance and hoping that it is a first step in the explanation of the real algorithmic behaviour for a wider class of polytopes. In this state experiments may prove useful to gain a better understanding of polytopal structures and the behaviour of different algorithmic concepts. For a broader survey on complexity results see e.g. [11].

It is important to note that the algorithms differ in the input they require; because the transformation $\mathcal{V} \rightleftarrows \mathcal{H}$ can be prohibitively costly for some polytopes, this can influence decisively the choice of algorithms. Though highly depending on the concrete structure of the polytope under consideration, one can expect tractability on workstations for polytopes with up to 10^2 hyperplanes and 10^4 vertices, or, 10^4 hyperplanes and 10^2 vertices, in dimension up to 15, requiring memory up to the range of 10^2– 10^3 MB, see Table 3.

The paper is organised as follows. In Section 2 we discuss two basic ideas of volume computation together with their dual relation, allowing a deeper understanding of the complexity observed. Section 3 is devoted to the five basic algorithms we have chosen from the literature. Then, in Section 4, we propose a number of improvements for Cohen & Hickey's and Lasserre's methods and present a hybrid algorithm. We conclude in Section 5 by presenting experimental results for seven classes of polytopes. As a by-product of our work, the code of all algorithms discussed is publicly available; the sources are given in the Appendix.

For a theoretical analysis, we assume a computer device with arbitrary precision. Namely we assume that all elementary arithmetic operations require an amount of time which is independent of the number of digits needed in the representation of the numbers. In such a computational model, the time complexity is proportional to the number of elementary arithmetic operations. While this assumption makes complexity analysis much simpler, it has an obvious drawback when some algorithms require much higher precision than others. We shall discuss the numerical issue independently for each algorithm in the Sections 3 and 5.

2 Basic Algorithmic Concepts and Duality

All known algorithms for exact volume computation decompose a given polytope into simplices, and thus they all rely, explicitly or implicitly, on the volume formula of a simplex:

$$\mathrm{Vol}(\Delta(v_0,\ldots,v_d)) = \frac{|\det(v_1-v_0,\ldots,v_d-v_0)|}{d!},$$

where $\Delta(v_0,\ldots,v_d)$ denotes the simplex in $\mathbb{R}^d$ with vertices $v_0,\ldots,v_d \in \mathbb{R}^d$. There are two types of methods for volume computation, depending on how a given polytope P is decomposed into simplices. As we shall explain later, these two algorithm classes are related by polarity of convex polytopes.

2.1 Triangulation Methods

A *triangulation*[1] of a d-polytope P is a set $\{\Delta_i : i = 1,\ldots,s\}$ of d-simplices such that $P = \cup_{i=1}^{s}\Delta_i$, and no distinct simplices have an interior point in common. Then the volume of P is simply the sum of the volumes of the simplices:

$$\mathrm{Vol}(P) = \sum_{i=1}^{s} \mathrm{Vol}(\Delta_i). \tag{1}$$

Figure 1 illustrates a triangulation using an interior point e; it is often used when the boundary of a polytope can be easily triangulated or is already triangulated as in the case of simplicial polytopes. Besides such a boundary triangulation method we will also consider the Delaunay triangulation and Cohen & Hickey's combinatorial triangulation by dimensional recursion. We postpone a detailed description of these methods to Section 3.1. A first important difference is that the former two methods need only a $\mathcal{V}$–representation while the last method requires both the $\mathcal{V}$– and $\mathcal{H}$–representations.

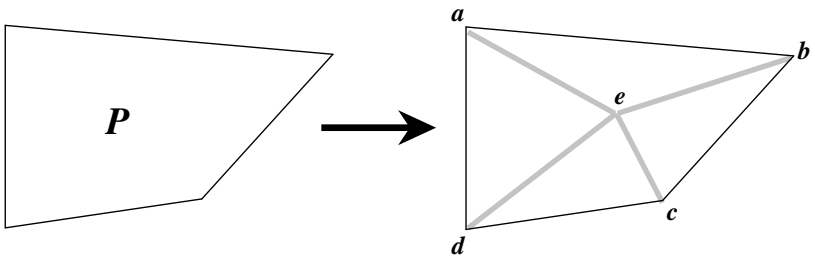

Figure 1: Boundary triangulation

2.2 Signed Decomposition Methods

Instead of triangulating a polytope P, one can decompose P into 'signed' simplices whose signed union is exactly P. More specifically, we represent P as a signed union of simplices Δ_i, $i = 1,\ldots,s$,

$$P = \biguplus_{i=1}^{s} \sigma_i\Delta_i, \tag{2}$$

where σ_i is either $+1$ or -1 and the right hand side is considered as a multi-set whose elements have (possibly negative) multiplicity. Equation (2) means that each point x in $\mathbb{R}^d$ not on the boundary of any

[1] This definition is slightly weaker than the usual one that requires the intersection of any two simplices being their common face. This additional property is satisfied by most known constructions of triangulations, but not necessary for the purpose of volume computation.

of the Δ_i's has the following property: If x is in P, it appears in exactly one more positive Δ_i than in negative Δ_i's, and otherwise it appears equally often in positive and in negative Δ_i's. Then the volume of P is

$$\mathrm{Vol}(P) = \sum_{i=1}^{s} \sigma_i \,\mathrm{Vol}(\Delta_i). \tag{3}$$

In this paper we consider two signed decomposition methods, Lawrence's decomposition as illustrated in Figure 2, and Lasserre's decomposition. These methods are presented in detail in Section 3.2. Fundamental to Lawrence's method for simple polytopes is the introduction of an extra hyperplane e

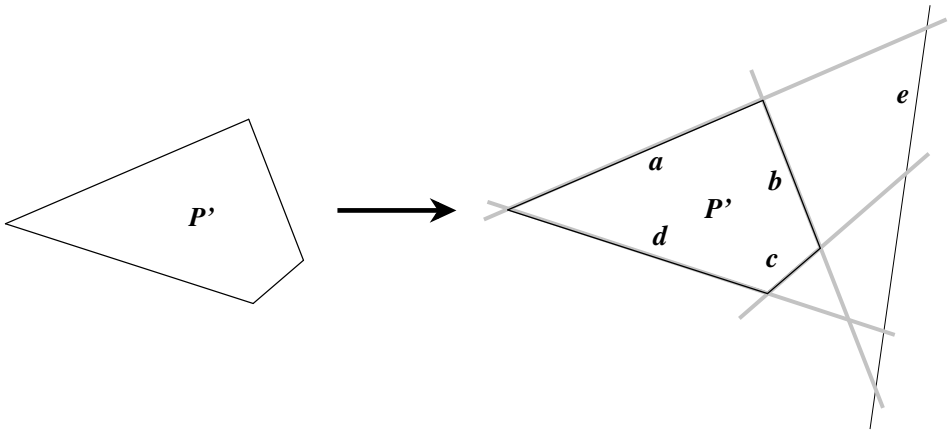

Figure 2: Lawrence's decomposition scheme

which is not parallel to any edge of the polytope. The simplices in the signed decomposition are generated by passing once through each vertex, and taking the simplex built by the vertex-touching hyperplanes together with e. In the example shown in Figure 2 the decomposition contains two positive simplices, ade and cbe, and two negative simplices, abe and cde, where xyz denotes the unique simplex determined by the hyperplanes x, y and z.

Note that Lasserre's method requires only the $\mathcal{H}$–representation, while Lawrence's method requires both the $\mathcal{V}$– and the $\mathcal{H}$–representations.

2.3 Duality

Assume that the origin lies in the interior of a polytope P; then the polar polytope P' is defined by $P' := \{x \,|\, y^T x \leq 1 \ \forall y \in P\}$. Note that the vertices are mapped onto the facets and vice versa as is illustrated in Figure 3. Filliman [8] showed that there is a dual relation between a triangulation of a polytope and a signed decomposition of its polar. While the left polytope is triangulated into the four simplices $\Delta(abe)$, $\Delta(bce)$, $\Delta(cde)$ and $\Delta(ade)$, the polar P' is decomposed into signed simplices determined by the same triples. The sign of each simplex xyz in the decomposition of P' is determined by the *separation parity* of the corresponding simplex $\Delta(xyz)$ in the triangulation of P, which is the parity of the number of facet-defining hyperplanes of $\Delta(xyz)$ which separate the simplex from the origin. The simplex $\Delta(bce)$ has even parity $(+1)$ since it has two facet-defining lines be and ce separating it from the origin. The simplex cde has odd parity (-1) since it has only one such line de. When the parity is odd, the corresponding simplex in the signed decomposition of the polar is negative. Thus we have two positive simplices cbe, ade and two negative simplices cde, abe.

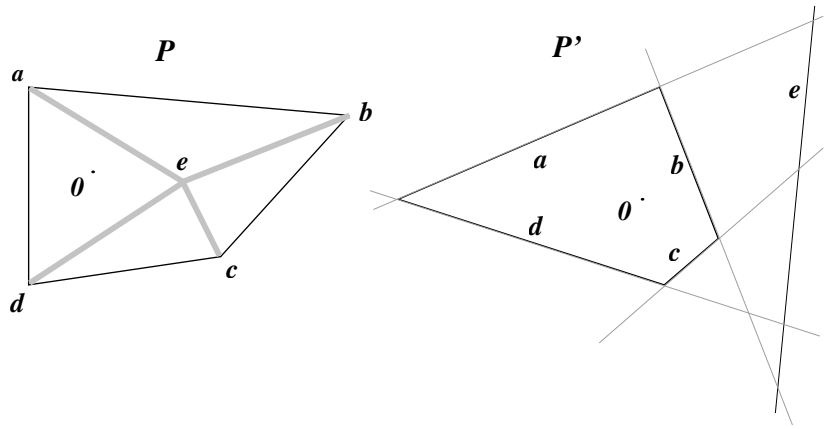

Figure 3: Filliman's duality

More generally Filliman showed:

> Let P be a d-polytope containing the origin in its interior and let $\mathcal{T}$ be a triangulation of P such that the origin is not contained in the union of hyperplanes spanned by vertices of the triangulation. Then the triangulation induces a signed decomposition of the polar P' of P in such a way that each simplex $\Delta(a_0, a_1, \ldots, a_d)$ is in $\mathcal{T}$ if and only if the unique bounded simplex determined by the corresponding hyperplanes in the polar appears as a simplex in the decomposition. The sign of each simplex in the signed decomposition is determined by its separation parity in the triangulation.

Observe that the assumption on the location of the origin is important in defining the separation parity properly.

2.4 Duality and Efficient Volume Computation

What does duality tells us for volume computation? Should one triangulate or determine a signed decomposition of a polytope to compute the volume? While these questions do not have a simple answer, they raise a basic point that should be kept in mind when we design or use volume computation algorithms. That is, for any fixed polytope, the size of a triangulation and that of a signed decomposition might be drastically different.

To illustrate this claim the unit d-hypercube C^d of volume 1 and its polar, the cross polytope, can serve as an example. C^d has 2^d vertices and $2d$ facets, and consequently Lawrence's signed decomposition produces 2^d simplices. On the other hand, a simple recursive triangulation of C^d (see Cohen & Hickey in Section 3 for details) has exactly $d!$ simplices. Are there any triangulations with much fewer simplices? To estimate a simple lower bound for the number of simplices in a triangulation we can use Hadamard's upper bound for the maximum volume of simplices inscribed in the unit hypercube, $d^{d/2}/d!$. Although the resulting lower bound for the number of simplices, $O((d/2)!)$, might be far from the unknown minimum number, the growth in the number of simplices is much faster for any triangulation method compared to Lawrence's signed decomposition. If we take the cross polytope, the situation is completely reversed.

When we know the number of simplices generated by an algorithm, this yields a trivial lower bound for the complexity of the algorithm. The above observation shows that such a lower bound can be prohibitively high, while there might be much better alternatives.

Indeed, in the case of the methods drawn from the literature we experimentally found a behaviour following exactly these hypotheses: All triangulation methods fail for hypercubes and convince for cross

polytopes, while Lawrence's signed decomposition method behaves in the opposite way. This was a strong motivation to develop a hybrid algorithm that combines the advantages of different methods.

It should be noted that Lasserre's method to be described in the next section is a signed decomposition, but its complexity does not relate well with those of the other algorithms. The only case we could analyse easily was the hypercube, for which the complexity is $O(d!)$, whence the algorithm behaves more like triangulation methods, see Section 3.2.

3 Basic Algorithms

In this section representatives of triangulation methods and signed decomposition methods described in the literature are presented. To allow a theoretical comparison we give some complexity analyses. While it is perfectly possible to give general upper bounds as in Section 2.4, these bounds are usually far from being tight. Hence, they hardly explain why an algorithm behaves differently on various polytope classes. On the other hand, it seems to be very difficult to obtain tight complexity results for a specific class of polytopes, since the volume computation algorithms depend on associated geometric and combinatorial structures of a given d-polytope that are not fully understood. So we decided to analyse the performance of the different methods on hypercubes, whose geometric and combinatorial structures are quite simple. We expect (and this could be observed in experiments, see Section 5) that the behaviour of the different methods on hypercubes is typical of a broader class of polytopes, namely the simple or near-simple polytopes, so that by duality, our analysis should well enlighten the practical performance of the algorithms.

3.1 Triangulation Methods

Delaunay triangulation

The geometric idea behind a Delaunay triangulation of a d-polytope is to 'lift' it on a paraboloid in dimension $d+1$. This is done on the level of vertices $v \to \bar{v}$, $(x_1, \ldots, x_d) \mapsto (x_1, \ldots, x_d, \sum_{i=1}^{d} x_i^2)$. If the resulting vertices are in general position, the convex hull of $\bar{V}$ yields a triangulation, whose 'lower half' is a Delaunay triangulation. In our experiments the underlying convex hull algorithm uses the 'beneath-beyond' method (see [11]). This method requires only the $\mathcal{V}$-representation.

Boundary triangulation

Even if P itself is not simplicial, lexicographic perturbation (either symbolically or numerically) of the vertices leads to a simplicial polytope. Computing the convex hull of the perturbed points and interpreting the result in terms of the original vertices leads to a triangulation of the boundary which, by linking with a fixed interior point, yields a triangulation of P. For the convex hull computation we chose the reverse search algorithm [1], where only the $\mathcal{V}$-representation is required as input.

Triangulation by Cohen & Hickey

This recursive scheme triangulates a d-polytope P by choosing any vertex $v \in P$ as apex and connecting it with the $d-1$-dimensional simplices resulting from a triangulation of all facets of P not containing v, see [4]. In the following, the notation e^k stands for some k-dimensional face of P. Let η be a map which associates to each face one of its vertices. Then the pyramids with apex $\eta(e^d)$ and bases among all facets e^{d-1} with $\eta(e^d) \notin e^{d-1}$ form a dissection of the polytope. Applying this scheme recursively to all e^{d-1} under consideration results in a set of decreasing chains of faces, $\mathcal{C} := \{(e^d, \ldots, e^1, e^0) \mid e^k \supset e^{k-1} \text{ and } \eta(e^k) \notin e^{k-1} \text{ for all } 1 \le k \le d\}$. Then the set of corresponding simplices $\{\Delta(\eta(e^0), \ldots, \eta(e^d)) \mid (e^d, \ldots, e^0) \in \mathcal{C}\}$ is a triangulation of P. To implement this recursive method extensive use of the double description as $\mathcal{V}$– and $\mathcal{H}$–polytope is made by representing all faces

as sets of vertices. Based on this combinatorial representation, we compute the list of possible e^{k-1} from a given e^k by intersecting the set of vertices of e^k with the facets e^{d-1} of P not containing the vertex $\eta(e^k)$. We do this by a binary look-up of the vertices of e^k in the set of vertices of e^{d-1}. To prevent the multiple generation of the same face in degenerate cases, a list is maintained containing all faces of e^k generated so far; only faces which are not in this list are accepted. In the sequel we call this basic algorithmic scheme *C&H*.

Note that in the case of C&H compared to a boundary triangulation all simplices in the facets containing the apex v are eliminated and thereby the number of simplices is usually reduced.

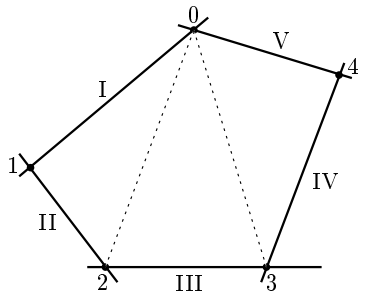

Figure 4: Cohen & Hickey's triangulation scheme

As an example consider Figure 4 where η assigns to each face its vertex with the lowest number, so $\eta(P) = 0$. Now all facets which do not contain the vertex 0 are examined, that is, II, III and IV. The scheme is now applied to facet II with $\eta(\text{II}) = 1$. II is intersected with all facets not containing the vertex 1, these are III, IV and V. The intersections with IV and V are empty, so these recursion branches are unsuccessful. The intersection with III yields the vertex 2, and the fixed vertices 0, 1 and 2 form a first simplex. The other simplices obtained from III and IV are also marked in the figure.

Proposition 1 *The time complexity of C&H applied to a d-hypercube is* $O(d^3 d!)$.

Proof: There are $2^{d-k}\binom{d}{k}$ faces of dimension k, each of which contains 2^k vertices. The recursion tree resembles the face lattice where some faces do not appear whereas others are present multiple times. The (unique) root is the original d-polytope P, and in each recursion level k *some* of the incident $(k-1)$-dimensional faces e^{k-1} are included. Denote by s_k the number of faces of dimension k appearing in the recursion tree; from the construction we know $s_d = 1$. In recursion k we intersect a face e^k with all facets not containing $\eta(e^k)$, and since there are k such facets we have $s_{k-1} = k s_k$, implying $s_k = \frac{d!}{k!}$. Hence the number of simplices, s_0, equals $d!$. Because for each simplex a determinant has to be computed, the overall complexity bound derived so far is $O(d^3 d!)$, assuming conventional matrix computation techniques. In the rest of the proof we verify that the additional computational burden of the algorithm per simplex is bounded by $O(d^3)$. Suppose that the intersection of e^k with a facet e^{d-1} is done by a binary look-up of the vertices of e^k in e^{d-1}, resulting in $O(|e^k| \log |e^{d-1}|) = O((d-1)2^k) \subseteq O(d2^k)$ comparisons, where the notation $|e^k|$ designates the number of vertices in the face e^k. To check further whether the newly determined face e^{k-1} appears in the list of at most $k-1$ faces already derived from e^k requires another $O((k-1)|e^{k-1}|) \subseteq O(d2^k)$ comparisons. The facet-intersection and the maintenance of the list have to be done for each facet, resulting in a total complexity of $O(\sum_{k=1}^{d} s_k k(d2^k + d2^k))$, where

$$\sum_{k=1}^{d} s_k k d 2^k = dd! \sum_{k=1}^{d} \frac{2^k}{k!} k < 2dd! \sum_{k=0}^{\infty} \frac{2^k}{k!} k = 2e^2 dd!.$$

Thus the additional computational burden per simplex is bounded by $O(d)$ which can be neglected compared to the determinant computation. □

3.2 Signed Decomposition Methods

Lasserre's recursive algorithm

The volume is computed via a recursive scheme based on Euler's formula for homogeneous functions. Denote by $\mathrm{Vol}(d, A, b)$ the volume sought, by a_i the i^{th} row of A, and by $\mathrm{Vol}_i(d-1, A, b)$ the volume in the appropriate $\mathbb{R}^{d-1}$ subspace defined by the face $P \cap \{a_i^T x = b_i\}$. Then the volume of P can be expressed by the following recursive structure:

$$\mathrm{Vol}(d, A, b) = \frac{1}{d} \sum_{i=1}^{m} \frac{b_i}{\|a_i\|} \mathrm{Vol}_i(d-1, A, b). \tag{4}$$

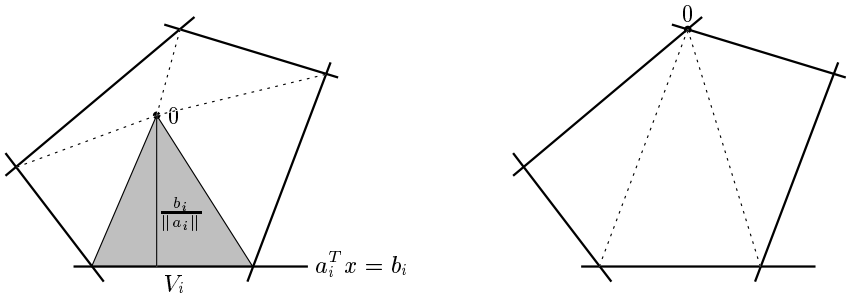

Figure 5: Triangulation induced by Lasserre's method if the origin is inside a facet (left) or if it coincides with a vertex (right). The volume $\frac{b_i}{d\|a_i\|} \mathrm{Vol}_i(d-1, A, b)$ of a single triangle is gray shaded in the left graph.

Now (4) is not yet an implementable recursion scheme because the terms in the sum have to be evaluated in appropriate lower dimensional spaces. To compute $\mathrm{Vol}_i(d-1, A, b)$ Lasserre [13] suggests a projection scheme as follows. For the constraint to be fixed choose a pivot element $a_{ij} \neq 0$ and set $x_j = (b_i - \sum_{k \neq j} a_{ik} x_{ik})/a_{ij}$. Substituting x_j in this way defines a new system $\tilde{A}\tilde{x} \leq \tilde{b}$ with $m-1$ constraints and $\tilde{d} := d-1$ variables for which we have $\mathrm{Vol}_i(d-1, A, b) = (\|a_i\|/a_{ij}) \mathrm{Vol}(\tilde{d}, \tilde{A}, \tilde{b})$. Geometrically $\mathrm{Vol}(\tilde{d}, \tilde{A}, \tilde{b})$ is the volume of $P \cap \{a_i^T x = b_i\}$ projected onto the subspace $\{x_j = 0\}$. Finally, using (4) gives the recursive scheme we implemented:

$$\mathrm{Vol}(d, A, b) = \frac{1}{d} \sum_{i=1}^{m} \frac{b_i}{a_{ij}} \mathrm{Vol}(\tilde{d}, \tilde{A}, \tilde{b}). \tag{5}$$

At the bottom of the recursion tree, the volume of a line segment bounded by (at most) $m - d + 1$ constraints has to be computed. It is of critical importance in this formula that no constraint appears more than once in any (recursively generated) face. Even though this is usually given for the input, it turns out that in many interesting examples multiple constraints occur in lower dimensional faces. In the process of eliminating identical constraints non-identical parallel constraints are treated simultaneously, where two cases are possible: either the intersection of two parallel halfspaces is non-empty or empty. While we can stop the recursion branch in the latter case, we check in the former case whether the normal vectors point in the same direction and if so drop the 'outer' of the two parallel hyperplanes. For hypercubes this implies that in dimension k there are $2k$ constraints in the reduced system.

Looking at (5) another simple improvement is possible by making as many components of b as possible equal to zero and thereby suppressing the recursion for the related branch. This could be done by shifting the most degenerate vertex of the face under consideration into the origin. In practice, however, we do not want to use information about vertices in order to preserve a pure $\mathcal{H}$-based algorithm. Thus we shift a face in dimension k only into a basis solution by solving an arbitrary, non-degenerate $k \times k$ subsystem of equations. In the case of hypercubes this decreases the number of constraints with non-zero b-component

in the reduced systems of dimension k from $2k$ to k. The resulting algorithm, including parallel face detection and shifting, is called *LAS*.

Proposition 2 *The time complexity of LAS applied to a d-hypercube is* $O(d!)$.

Proof: A similar analysis as in Proposition 1 can be made. To pass from e^k to e^{k-1} we choose among k faces with non-zero b-component, resulting in the recursive equation $s_{k-1} = ks_k$. With $s_d = 1$ it follows that $s_k = \frac{d!}{k!}$. Fixing one constraint together with shifting requires $O(k^3)$ operations for solving a system of linear equations, and the subsequent check for multiple restrictions (which detects the hyperplane parallel to the newly fixed one) needs $O(k^3)$ steps, yielding a complexity bounded by $O(\sum_{k=1}^{d} s_k k k^3) = O(d! \sum_{k=1}^{d} \frac{k^3}{(k-1)!})$. Now $\sum_{k=1}^{d} \frac{k^3}{(k-1)!}$ is bounded by $22.5 + 11 \sum_{k=0}^{d-4} \frac{1}{k!}$, and with $\sum_{k=0}^{d} \frac{1}{k!} < e$ we derive a total complexity bound of $O(d!)$. □

Note that no extra determinant computation is needed because this is implicitly done in the course of fixing constraints. LAS improves on C&H by a factor of d^3.

Lawrence's volume formula

Assume P is *simple* and choose a vector $c \in \mathbb{R}^d$ and a scalar q such that the function $x \mapsto c^T x + q$ is not constant along any edge. For each vertex $v \in V$ let A_v be the $d \times d$-matrix composed by the rows of A which are binding at v. Then A_v is invertible and $\gamma^v := (A_v^T)^{-1} c$ is well defined up to permutations of the entries. The assumption imposed on c assures that none of the entries of γ^v is zero. Lawrence [15] shows that then

$$\mathrm{Vol}(P) = \sum_{v \in V} \frac{(c^T v + q)^d}{d! \,|\det A_v| \prod_{i=1}^{d} \gamma_i^v}. \tag{6}$$

Lawrence's formula easily generalises to the non-simple case using standard lexicographic perturbation techniques. Consequently the summation in (6) must be done over all lexicographically feasible cobases of all $v \in V$.

An open question is how to choose c and q properly; even if $c^T x + q$ is not constant on any edge, a 'nearly constant' choice results in very small entries γ_i^v in the denominator, causing potentially severe numerical problems. Indeed, such numerical instabilities were regularly found in our experiments. It should be noted that a theoretically satisfactory[2] choice is given in [15], but its practical importance is probably very small since the number of digits needed to represent a theoretically guaranteed vector will be too long to perform any reasonable computation.

The time complexity for the simple case is $O(d^3 n)$, where n denotes the number of vertices, and for cubes this implies a complexity of $O(d^3 2^d)$.

4 Improvements and a Hybrid Method

The algorithms described in the previous section can be improved both theoretically and practically by a few simple ideas. In this section we describe such improvements for C&H and LAS. Moreover we present a hybrid approach called HOT for 'hybrid orthonormalisation technique'. To underline the qualitative differences of the methods a few complexity estimates are given.

4.1 Modified Cohen & Hickey's Method (C&H)

The major advantage of a triangulation method using both the $\mathcal{H}$- and $\mathcal{V}$-representation is the possibility for a computationally cheap, but powerful test on empty or simplicial faces. Consider in the recursive

[2] It is possible to select the vectors c and q whose binary lengths are polynomially bounded by the input size.

process a face e^k; then we know that the volume of e^k in the appropriate k-dimensional affine embedding is zero if the number of vertices in e^k is smaller than $k+1$. And if the number of vertices is exactly $k+1$ the recursive process can stop with a face which is either a k-simplex or has volume zero. Even though this does not change the behaviour on simple polytopes like hypercubes, i.e. Proposition 1 still applies, simplicial polytopes have a reduced complexity of $O(md^3)$, where m denotes the number of facets.

Another improvement uses degeneracy information: If the vertices are sorted decreasingly by the number of incident hyperplanes, the number of e^{k-1}'s not containing v_k may drop significantly.

We abbreviate the algorithm obtained by incorporating these modifications again by C&H.

4.2 Revised Lasserre's Method (r-LAS)

Observing that when the same constraints are fixed in a different order, the same face may appear several times in (4), it is natural for Lasserre's method to store and subsequently reuse the (projected) face volumes; storing can be done, e.g., in a balanced tree. We call the resulting algorithm r-LAS. Due to the projection, not only the indices of the fixed constraints I characterising the face e^k have to be stored, but also the fixed variables $\bar{J}$ representing the projection space $\Pi_J = \mathbb{R}^d \cap \{x \,|\, x_j = 0 \;\forall j \in \bar{J}\}$, where $J = \{1, \ldots, d\} \backslash \bar{J}$ denotes the set of free variables. Once the volume of a face e^k projected onto Π_J is known, its projection onto $\Pi_{J'}$, where $J \neq J'$ but $|J| = |J'|$, can be found directly by computing the determinant of the transformation $\Pi_J \to \Pi_{J'}$. Denote by A_{IJ} the matrix consisting of the rows and columns of A indexed by I and J respectively. In a first step, we 'deproject' $\Pi_J \to \mathbb{R}^d$, $x_J \mapsto x = \left[\begin{smallmatrix} x_{\bar{J}} \\ x_J \end{smallmatrix}\right]$, with $x_{\bar{J}} = A_{I\bar{J}}^{-1}(b_I - A_{IJ}x_J)$. In a second step x is projected on $\Pi_{J'}$, $x \mapsto x_{J'}$, by dropping the components related to $\bar{J}'$. Therefore the rows J' of the linear part of the map $x_J \mapsto x$ yield a nonsingular $(d-k)\times(d-k)$ matrix whose determinant s fixes the projective effect, i.e. $\mathrm{Vol}(\Pi_{J'}(e^k)) = |s| \cdot \mathrm{Vol}(\Pi_J(e^k))$.

Proposition 3 *The time complexity of r-LAS applied to a d-hypercube is $O(d^4 3^d)$.*

Proof: (Cf. the proof of Proposition 2) Again, let s_k denote the number of faces considered in dimension k. Starting with any of the k-dimensional faces, we have to fix k constraints and subsequently check for identical and parallel constraints. Each of these steps can be done in time $O(k^3)$. We obtain ks_k faces of dimension $k-1$. Each of these faces has to be looked up in the balanced tree containing the projected face volumes. Taking into account that at most all 3^d faces of the cube are stored in the tree, and that one key comparison can be done with $O(m-k) \subseteq O(d)$ index comparisons, the effort for one look-up is in $O(d^2)$. Notice now that some of the ks_k faces are found in the tree, causing each an overhead of $O(d^3)$ for the deprojection. So the total complexity is in $O(ks_k(k^2+d^2+d^3)) \subseteq O(d^4 s_k)$. We now claim that s_k, the number of k-dimensional faces stored in the tree, is bounded above by the total number $2^{d-k}\binom{d}{k}$ of k-dimensional faces. This is true since in a hypercube, which is simple, each non-empty k-dimensional face is the intersection of a unique set of $d-k$ facets. Moreover, empty faces are not stored, for this would involve an intersection of parallel hyperplanes. Hence the overall arithmetic complexity is $O(d^4 \sum_{k=0}^{d} 2^{d-k}\binom{d}{k}) = O(d^4 3^d)$. □

4.3 A Hybrid Method (HOT)

In the course of our experiments we identified two critical factors explaining an essential part of the differences in computation time if Cohen & Hickey's triangulation method is compared with Lasserre's signed decomposition method: Empty face detection and storing/reusing intermediate results.

Triangulation methods using vertex and incidence information permit an effective test on simplicial or empty faces by simply counting the number of vertices. Signed decomposition methods like Lasserre's one are usually facet-based where a test on empty faces, e.g. by means of linear programming, is significantly more costly. The situation is reversed when storing/reusing intermediate results is considered. Here triangulation methods like Cohen & Hickey's algorithm suffer from the fact that they are based on simplex volumes and not on face volumes. Lasserre's scheme on the other hand works directly with the

(projected) volumes of faces, offering a simple way of storing and reusing face volumes in the recursion process.

Among the possibilities to construct a hybrid method, one can include an empty face detection in Lasserre's method or a storing/reusing scheme in Cohen & Hickey's method. Concerning the first possibility we tested an LP-based empty face detection in Lasserre's algorithm, but it turned out to be not competitive in most cases.[3] On the other hand, storing the whole sub-triangulation of a face in Cohen & Hickey's algorithm would require by far too much memory.

So we adopted a different approach, which nevertheless is based on Cohen & Hickey's enumeration of faces and profits from all the improvements described in Section 4.1. Notice that the volume of P can be derived directly from the volume of its facets by the formula

$$\mathrm{Vol}_d(e^d) \quad = \quad \frac{1}{d} \sum_{e^{d-1}: v_d \notin e^{d-1}} \mathrm{Vol}_{d-1}(e^{d-1}) \, \mathrm{dist}(v_d, \mathrm{aff}(e^{d-1})), \tag{7}$$

where Vol_d denotes the d-dimensional volume, aff the affine hull of a set, and $v_d = \eta(e^d)$. This corresponds to a subdivision of P into pyramids with apex v_d and bases e^{d-1} instead of simplices. The distance between v_d and $\mathrm{aff}(e^{d-1})$ is easily computed if an orthonormal basis of the affine subspace spanned by e^{d-1} is known, which can be computed using Householder transformations. The face volumes $\mathrm{Vol}_k(e^k)$ are stored in a balanced tree with the vertices as key; they are reused whenever the same face reappears during the recursion. We call the resulting algorithm *hybrid orthonormalisation technique* (HOT). Since storing the orthonormal bases would require an enormous amount of memory, they are recomputed from scratch when a $\mathrm{Vol}_k(e^k)$ is reused.

Proposition 4 *The time complexity of HOT applied to a d-hypercube is* $O(d^2 4^d)$.

Proof: (Cf. the proof of Proposition 3) In dimension k, a number of s_k faces are intersected by k facets, and each of the resulting faces is then looked up in the tree, requiring $O(k\, s_k\, 2^k d)$ comparisons, cf. the related complexity analysis for C&H in Proposition 1. By s_{k-1} we denote the number of $(k-1)$-dimensional faces where the recursion proceeds because the corresponding volume is not yet known. After these recursions are finished, $O(d^2 s_{k-1})$ steps are needed for adding one vector to the orthonormal basis. For the remaining $ks_k - s_{k-1}$ faces, which are retrieved from the tree, orthonormal bases are computed from scratch, taking $O(d^3(ks_k - s_{k-1}))$ steps. Adding together, the total complexity is bounded by $O(\sum_{k=1}^{d}(ks_k 2^k d + d^2 s_{k-1} + d^3(ks_k - s_{k-1}))) \subset O(d^2 \sum_{k=0}^{d} s_k 2^k + d^4 \sum_{k=0}^{d} s_k)$. To estimate s_k note that a face is uniquely determined by the set of its vertices, so the same face can be stored only once in the tree. As before, it follows that $s_k \leq 2^{d-k}\binom{d}{k}$. We obtain therefore an overall arithmetic complexity of $O(d^2 2^d \sum_{k=0}^{d} \binom{d}{k} + d^4 \sum_{k=0}^{d} 2^{d-k}\binom{d}{k}) = O(d^2 4^d + d^4 3^d)$. □

Note that even though both r-LAS and HOT store and reuse face volumes, the former does it as projected volumes and the latter as the original (absolute) volumes. As a consequence, one has to deal either with deprojections or the computation of bases (by orthonormalisation). In principle one could also set up a Lasserre scheme based on unprojected volumes. While this would not change the essential behaviour of Lasserre's algorithm, it might be interesting to test its practicality. We leave this as a future research project.

5 Experimental Results

The implemented algorithms have been tested on a broad range of examples, varying from polytopes created with random vertices or hyperplanes to specially structured polytopes as they arise in combinatorics. We denote by d the dimension, by m the number of hyperplanes and by n the number of vertices. The groups of examples are ordered from large to small ratio n/m:

[3] Our tests are based on the efficient callable library of cplex using restarting techniques.

- cube–d: Hypercube in dimension d.
- rh–d–m: d-dimensional problems with m randomly generated constraints. The normal vectors of the constraints have integer components and a vector norm of approximately 1000; the right hand side is fixed to 1000. The resulting polytopes are (most probably) simple.
- $\mathrm{CC}_d(k)$: The product of two cyclic polytopes over k points in dimension d.
- Fm–d: One facet of the metric polytope in dimension d.
- ccp–v: Complete cut polytope on v vertices, scaled by two and translated by one.
- rv–d–n: d-dimensional convex hulls of n randomly generated vertices on a sphere with radius 1000. The resulting polytopes are (most probably) simplicial.
- cross–d: Cross polytope of dimension d; it is the polar of the d-dimensional cube.

A discussion of these examples can be found in [1]. We emphasise that our main concern is the comparison of the general behaviour of the methods and not of specific implementations; furthermore, most of the codes are experimental in nature leaving room for improvements. The observed time should therefore be interpreted cautiously.

Unless otherwise stated the computed volume is identical for all methods for at least six digits and is therefore given only once. The general abbreviations for the codes used in the sequel are given below in the first column; in the second column the shorter ones used in Table 3 and 4 are stated.

BND	Bnd	boundary triangulation using 'lrs'
DEL	Del	Delaunay triangulation using 'qhull'
C&H	CH	Cohen & Hickey's triangulation including our improvements
HOT	HOT	hybrid orthonormalisation technique
LAW-nd	Lnd	Lawrence's formula in the non-degenerate case
LAW-d	Ld	Lawrence's formula in the general case using 'lrs' for computing all lexicographically feasible cobases
LAS		original Lasserre
r-LAS	rL	revised Lasserre

Sources for the codes are given in the appendix. The only code using rational arithmetic is 'lrs'. All computations have been done on an HP 7000/735–99. Among the many parameters influencing the behaviour of the codes the one determining the level of storing face volumes is of major importance in case of HOT and r-LAS. For HOT and most of the examples computed with r-LAS it was possible to store all faces; in the remaining examples the choice was done to take most advantage of the memory available which was limited to approximately 500 MB. Obviously, for larger problems there is a decisive trade-off between memory and CPU usage if face volumes are stored/reused. In case of r-LAS another influencing parameter is the minimal absolute value MIN_PIVOT accepted for pivoting; the strategy is to go through a row and pick the first element which exceeds MIN_PIVOT in absolute value. If none is found the absolutely largest element is chosen. The larger MIN_PIVOT, the slower r-LAS because the probability of projecting the same face onto the same subspace decreases. In our experiments we set MIN_PIVOT to 0.01. A too small MIN_PIVOT, however, contains the risk of numerical instabilities.

In the following two sections we examine the effect of our modifications on Cohen & Hickey's and Lasserre's algorithms. Then a comparison of the modified methods with all other codes on the full set of test problems is presented. Here we drop the original C&H and LAS because they are not competitive.

5.1 C&H and Its Modified Version

As described in the previous section, the original Cohen & Hickey triangulation has been improved in two ways: Detection of simplicial faces leads to an early abort strategy, and ordering the vertices following the

Problem	d/m/n	original C&H	modified C&H
cube–7	7/14/124	1.5	1.0
cube–8	8/16/256	13	7.6
cube–9	9/18/512	130	82
cube–10	10/20/1024	1400	940
cube–11	11/22/2048	18000	14000
cross–7	7/124/14	0.3	0.1
cross–8	8/256/16	0.8	0.2
cross–9	9/512/18	3.4	0.3
cross–10	10/1024/20	15	0.6
cross–11	11/2048/22	71	1.4
cross–12	12/4096/24	340	4.3
cross–13	13/8192/26	1700	15
cross–14	14/16384/28	7600	65

Table 1: The original and the modified version of C&H

number of incident hyperplanes potentially decreases the number of computed simplices. Table 1 reports the observed running time for both the original and the improved code on cubes and cross polytopes.

Since both cubes and cross polytopes have a constant number of incident hyperplanes for all vertices, vertex reordering has no effect. Concerning the detection of simplicial faces, cubes profit only a little; because they are simple the recursion still has to go down to the edges.

The situation changes dramatically for cross polytopes. Since they are simplicial, the recursive scheme is reduced to computing a determinant for half of the simplicial facets.

As an example where vertex reordering has a significant impact, we observed in case of Fm_6 a drop from 8400 CPU-seconds (reversed ordering) to 25 s. (ordered following maximal incidence), where in both cases simplicial face detection was enabled. Fm_6 also demonstrates the practical gain of our early abort strategy for neither simple nor simplicial polytopes. Using the optimal vertex ordering, the running time rises from 25 s to 4600 s if the detection of simplicial faces is turned off.

5.2 LAS versus r-LAS

In Table 2 the effects of shifting into a basis solution and storing face volumes are demonstrated for cubes and cross polytopes. The first component in each pair of signs relates to shifting, the second component to storing. A '+' means active, a '−' means not enabled. The column $(+,-)$ represents Lasserre's original algorithm, $(+,+)$ stands for our revised code r-LAS.

		Variants of Lasserre			
Problem	d/m/n	(−, −)	(+, −)	(−, +)	(+, +)
cube–7	7/14/124	4.4	0.13	0.26	0.0
cube–8	8/16/256	69	1.4	1.4	0.0
cube–9	9/18/512	1200	13	6.3	0.13
cube–10	10/20/1024	25000	130	21	0.26
cross–7	7/124/14	22	11		
cross–8	8/256/16	340	180		
cross–9	9/512/18	5400	2900		
cross–10	10/1024/20	92000	48000		

Table 2: Lasserre's algorithm with (+) or without (−) shifting and storing, respectively. The case (+, −) represents LAS, whereas (+, +) is r-LAS.

For cubes the effects of these two factors reduce the CPU-time drastically, where the effect of storing is slightly more important than that of shifting. In case of cross polytopes, however, the facet-based approach of r-LAS has severe difficulties exploiting the structure. First of all, storing has no effect because no face ever appears twice, thus $(?, -)$ equals $(?, +)$, for $? \in \{+, -\}$. Secondly, shifting is only done once at the very beginning because in every recursion level $k > 0$, already at least $d-k$ components of the right hand side are zero. Therefore we observe a CPU-time without shifting which is roughly twice the time needed when shifting is active.

5.3 Comparing the Main Codes

In Table 3 the timing in CPU-seconds for all codes and examples is given; because the transformation $\mathcal{V}\to\mathcal{H}$ or $\mathcal{H}\to\mathcal{V}$ can be as demanding as the volume computation itself, and furthermore we need both representations for certain methods, we give in the last two columns the transformation time when using 'lrs' and 'pd' with exact arithmetic, see Appendix 6.

						Triangulation				**Signed decmp.**				
						$\mathcal{V}$**–r.**		$\mathcal{V}$**– and** $\mathcal{H}$**–r.**		$\mathcal{H}$**–r.**				
Exp.	$\frac{n}{m}$	d	m	n	vol.	Bnd	Del	CH	HOT	Lnd	Ld	rL	$\mathcal{H}\to\mathcal{V}$	$\mathcal{V}\to\mathcal{H}$
cube-9	28.4	9	18	512	512	25e3	270	82	4.0	.3	1.6	.3	1.1	12
cube-10	51.2	10	20	1024	1024	**[c]	**[a]	940	18	.6	3.3	.5	2.7	36
cube-14	585	14	28	16384	16384	**[c]	**[a]	**[c]	3300	13	90	10	88	3500
rh-8-20	56	8	20	1115	37576	**[c]	**[b]	93	7.8	1.5	28	14	28	1900
rh-8-25	104	8	25	2596	786.0	**[c]	**[b]	430	27	3.5	80	120	80	4600
rh-10-20	109	10	20	2180	13883	**[c]	**[b]	3000	63	5.2	90	31	91	9500
rh-10-25	309	10	25	7724	5729.5	**[c]	**[b]	34e3	390	15	390	600	380	44e3
$CC_8(9)$	1.5	8	54	81	13340	410	40	11	3.4	**[c]	75	140	72[d]	140[d]
$CC_8(10)$	1.43	8	70	100	156816	1300	80	28	7.7	**[c]	210	880	200[d]	340[d]
$CC_8(11)$	1.38	8	88	121	1.39e6	3500	150	65	17	**[c]	550	4400	550[d]	800[d]
Fm-6	3	15	59	177	286114	11e5	**[a]	25	12	**[c]	**[b]	6900[9]	13e3[d]	6600
ccp-5	0.3	10	56	16	2.3117	.7	.5	.2	.2	**[c]	200	2900[6]	4.5	.7
ccp-6	0.09	15	368	32	1.3458	520	150	43	23	**[c]	**[c]	**[c]	4400[d]	2000[d]
rv-8-10	0.42	8	24	10	1.41e19	.3	.2	.1	.1	**[c]	82	.2	16	.5
rv-8-11	0.2	8	54	11	3.05e18	.4	.1	.2	.1	**[c]	2100	79	33	.9
rv-8-30	0.007	8	4482	30	7.35e21	170	4.0	6.9	6.8	**[c]	**[c]	**[c]	7200	150
rv-10-12	0.34	10	35	12	2.14e22	.4	.2	.1	.2	**[c]	1300	.3	92	1.0
rv-10-14	0.08	10	177	14	2.93e23	1.9	.2	.2	.1	**[c]	**[c]	**[c]	510	5.1
cross-8	0.063	8	256	16	6.35e-3	.5	.3	.2	.2	**[c]	2800	170	4.2	.5
cross-9	0.035	9	512	18	1.41e-3	0.9	.5	.3	.3	**[c]	**[b]	2700	12	1.2

Table 3: Timing for the different methods and examples. [a]beyond memory limit, [b]volume computed incorrectly, [c]problem is intractable with this method, [d]'cdd' is faster by a factor of at least 100, [6,9]storage performed for 6 and 9 levels resp.

Next, to measure potential numerical difficulties, Table 4 presents condition numbers for some algorithms. As different methods face different numerical problems we decided to define two different condition numbers: For triangulation methods all partial volumes are positive; the only problem lies in summing up volumes of potentially different magnitudes. Hence in this case we define the condition number as $\log_{10}$ of the biggest simplex volume divided through the smallest nonzero one (zero volumes occur with 'qhull', which returns some degenerate simplices). When evaluating Lawrence's formula the main problem is that

positive and negative numbers have to be summed up whose absolute values may be much larger than the final result. Here we use as condition number the biggest absolute summand divided through the correct volume, again as logarithm to the base 10. This number would be at most zero for triangulations. All condition numbers are rounded up.

Example	**Del**	**CH**	**Ld**
cube–9	2	0	8
cube–10	**[a]	0	9
cube–14	**[a]	**[c]	14
rh–8–20	**[b]	16	7
rh–8–25	**[b]	15	10
rh–10–20	**[b]	17	13
rh–10–25	**[b]	16	12
$CC_8(9)$	6	6	10
$CC_8(10)$	7	7	8
$CC_8(11)$	8	8	8
Fm-6	**[a]	3	**[b]
ccp–5	1	1	11
ccp–6	2	1	**[c]
rv–8–10	1	1	7
rv–8–11	3	2	11
rv–8–30	6	6	**[c]
rv–10–12	1	1	10
rv–10–14	4	3	**[c]
cross–8	0	0	11
cross–9	0	0	**[b]

Table 4: Condition numbers for the different methods and examples. [a]beyond memory limit, [b]volume computed incorrectly, [c]problem is intractable with this method.

Based on the Tables 3 and 4 we comment on the efficiency and the numerical stability of each algorithm separately. As a general remark we find huge differences in CPU time among the different codes for the same polytope. Some problems are even intractable with certain methods whereas they are quite efficiently solved by others, demonstrating the relevance of a good choice. Moreover, we regularly faced practical problems ranging from insufficient memory to numerical instabilities.

The boundary triangulation BND, done by 'lrs' using rational (exact) arithmetic, requires in general more CPU-time, but can nevertheless be competitive for some near-simplicial polytopes with small ratio n/m. In all tested cases this method produced the largest number of simplices compared to the other triangulation methods.

The Delaunay triangulation DEL produced by 'qhull' exhibits numerical problems in many examples. For specially structured polytopes, where the vertices are not in general position, numerical perturbation is needed, which may cause a break–down as observed with hypercubes. Moreover, the memory space needed for the quick hull algorithm and in fact all incremental algorithms, see [2], is not bounded above by a polynomial in the input and output size, and quite often a memory overflow occurs. The condition number is comparable to C&H, as well as the number of generated simplices.

C&H turns out to be numerically very robust due to its combinatorial nature. For special problems it profits a lot from degeneracy information, e.g. for Fm-6 where one vertex lies in all but one facet. (This information is not used by r-LAS.) The bigger the ratio n/m, however, the more C&H is slowed down by the increasing number of simplices.

The new hybrid code, HOT, is uniformly faster than C&H. Compared to r-LAS the situation is slightly different. Leaving apart hypercubes, the hybrid method is faster in most cases, but suffers from the

burden of computing orthonormal bases for retrieved faces. Considering the memory requirement, HOT is in most problems more modest than r-LAS. To explain the tremendous effect of reusing face volumes in HOT, we observed for increasing $\frac{n}{m}$ an increased average reuse of already computed face volumes. Thus HOT handles just the examples efficiently where C&H fails while making still use of C&H's strong point in the other examples, namely its detection of simplicial faces.

Numerical instabilities, partly due to the unresolved problem of choosing the objective function, are a serious drawback of Lawrence's approach presented in the columns labelled LAW-nd (non-degenerate) and LAW-d (degenerate: 'lrs' is used for finding all cobases). However, for most of the examples the result is correct, even though in some cases positive and negative summands occurred with absolute value going by a factor of 10^{14} beyond the final volume. In such a situation numerical cancellation can drastically destroy the accuracy of the final result. For simple polytopes LAW-nd is the fastest method exploiting efficiently the double description.

r-LAS is quite efficient for near-simple polytopes with a high ratio $\frac{n}{m}$. It profits a lot from memory for storing intermediate results. Due to the detection of parallel facets it works extremely fast on hypercubes. If the number of hyperplanes increases, however, time and/or memory requirement can grow exceedingly large, see ccp-6, rv-8-30 or rv-10-14. Numerically r-LAS performs almost as robustly as C&H in our examples.

The transformation times between $\mathcal{H}$– and $\mathcal{V}$–representations, given in the last two columns, have to be interpreted with extra caution. Most of the known algorithms face an 'easy' and a 'difficult' direction, where, roughly speaking, the easy transformation is the one from few vertices or facets to many facets or vertices, and the difficult situation is the converse direction. (For a thorough discussion of these issues, see [1].) We observed that especially the more demanding transformation can be several times more time consuming than the volume computation itself. A new result, however, changes this situation: It shows that using the easy direction as an oracle to the hard one, this latter task is considerably simplified [3]. Up to date, only one implementation of these so-called primal-dual algorithms is available, namely 'pd', which uses reverse search with exact arithmetic as the oracle. In Table 3 the transformation times using 'lrs' for the 'easy' and 'pd' for the 'difficult' direction are reported. Still, the transformations can be more time consuming than the actual volume computation, so it seems worthwhile to take the representation into account when choosing an algorithm for the volume computation. We hope, however, that such primal-dual approaches, especially when using floating point arithmetic, will considerably ease the burden of transformation in the near future. It should also be noted that some hard transformations, like $\mathcal{V}$– to $\mathcal{H}$–representation for random hyperplanes, are very unlikely to occur in practice.

6 Conclusions

The experimental results indicate that vertex based triangulations are superior for a small ratio n/m, namely for simplicial polytopes, whereas constraint based signed decomposition codes behave favourably for large n/m, e.g. for simple polytopes. Outstanding is—except for hypercubes—the behaviour of the hybrid code HOT; it convinces on both near-simple and near-simplicial polytopes, and exhibits a modest need for memory on our set of examples. Thus, in the general case, given both representations or an efficient transformation code, HOT seems to be the method of choice.

While we investigated many of the existing techniques for computing the volume of polytopes, this research can be considered as a starting point. For example, it might be interesting to study how a specific volume computation algorithm can be extended to compute mixed-volumes (see, e.g. [17]) and to investigate their complexities. Another natural extension is to study the efficient integral computation over a polytope, cf. [14]. We leave these for future research.

Appendix A: Availability of the Codes

All algorithms described in Section 3 and 4 have been implemented in C by the authors, using four publicly available additional programs, namely 'cdd' (double description in C, transfers between $\mathcal{H}$- and $\mathcal{V}$-representations), 'qhull' (quick hull, computes convex hulls for a given set of vertices), 'lrs' (lexicographic reverse search, constructs convex hulls and cobases using lexicographic perturbation), and 'pd' (primal-dual version of reverse search). The resulting volume computation package called 'vinci' provides a common framework for the different methods and codes. While C&H, HOT and r-LAS are implemented directly in 'vinci', the remaining algorithms—Delaunay triangulation, boundary triangulation and Lawrence's method—are realised using 'qhull' for the first and 'lrs' for the other two methods; in the special case of a simple polytope the incidence information suffices to generate Lawrence's signed decomposition and hence a direct summation, omitting 'lrs', can be performed. The necessary communication is done via text files. 'Vinci' checks the input and calls the appropriate external code if necessary. After the termination of 'qhull' or 'lrs', 'vinci' reads the resulting triangulation or signed decomposition and makes the appropriate summation of simplex volumes. Concerning Lawrence's method we use random values for c and q. A thorough discussion of 'cdd', 'qhull' and 'lrs' can be found in [1, 3], the algorithm implemented in 'pd' is described in [3]. 'Vinci' and the additional programs are freely available at the following sites:

vinci	authors:	Benno Büeler (bueeler@ifor.math.ethz.ch) and Andreas Enge (enge@ifor.math.ethz.ch)
	www page:	http://www.mathpool.uni-augsburg.de/~enge
	ftp site:	ftp.ifor.math.ethz.ch (129.132.154.13)
	directory:	pub/volume
	file name:	vinci-*.tar.gz, where '*' stands for the actual version number
lrs	author:	David Avis (avis@cs.mcgill.ca)
	ftp site:	mutt.cs.mcgill.ca (132.206.3.13)
	directory:	pub/C
	file name:	lrs*.c, where '*' stands for the actual version number
pd	author:	Ambros Marzetta (marzetta@inf.ethz.ch)
	www page:	http://wwwjn.inf.ethz.ch/ambros/pd.html
qhull	authors:	Brad Barber (bradb@geom.umn.edu) and Hannu Huhdanpaa (hannu@geom.umn.edu)
	www page:	http://www.geom.umn.edu/locate/qhull
	ftp site:	ftp.geom.umn.edu (128.101.25.35)
	directory:	pub/software
	file name:	qhull-*.*.tar.Z, where '*.*' stands for the actual version number
cdd+	author:	Komei Fukuda (fukuda@ifor.math.ethz.ch)
	www page:	http://www.ifor.math.ethz.ch/ifor/staff/fukuda/cdd_home/cdd.html
	ftp site:	ftp.ifor.math.ethz.ch (129.132.154.13)
	directory:	pub/fukuda/cdd
	file name:	cdd+-***.tar.gz, where '***' stands for the actual version number

Acknowledgement: We are grateful to Jean-Bernard Lasserre for encouragement and fruitful discussions related to his volume computation method, and to Paul A. Vixie[4] for giving us his efficient avl-tree implementation. We thank Günter Rote for fruitful discussions, and Gyula Karolyi for pointing out Filliman's paper [8] to us, which turned out to be very useful in classifying algorithms.

[4]Internet Software Consortium, Star Route Box 159A, Woodside, CA 94062 USA, vixie@vix.com, http://www.isc.org/isc/

References

[1] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms. *Computational Geometry: Theory and Applications*, 7:265–302, 1997.

[2] D. Bremner. Incremental convex hull algorithms are not output sensitive. Preprint, School of Computer Science, McGill University, Montreal, Canada, 1997. To appear in Discrete Comput. Geom.

[3] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 49–56, 1997. Full paper to appear in Discrete Comput. Geom.

[4] J. Cohen and T. Hickey. Two algorithms for determining volumes of convex polyhedra. *Journal of the ACM*, 26(3):401–414, July 1979.

[5] M. E. Dyer and A. M. Frieze. The complexity of computing the volume of a polyhedron. *SIAM J. Comput.*, 17:967–974, 1988.

[6] M.E. Dyer. The complexity of vertex enumeration methods. *Math. Oper. Res.*, 8:381–402, 1983.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

[8] P. Filliman. The volume of duals and sections of polytopes. *Mathematika*, 39:67–80, 1992.

[9] K. Fukuda, T. M. Liebling, and F. Margot. Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron. *Computational Geometry*, 8:1–12, 1997.

[10] K. Fukuda and A. Prodon. Double description method revisited. In M. Deza, R. Euler, and I. Manoussakis, editors, *Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91–111. Springer-Verlag, 1996.

[11] P. Gritzmann and V. Klee. On the complexity of some basic problems in computational convexity: II. Volume and mixed volumes. In T. Bisztriczky, P. McMullen, R. Schneider, and A.I. Weiss, editors, *Polytopes: Abstract, convex and computational (Scarborough, ON, 1993)*, NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., 440, pages 373–466. Kluwer Acad. Publ., Dordrecht, 1994.

[12] R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Struct. Algorithms*, 11(1):1–50, 1997.

[13] J. B. Lasserre. An analytical expression and an algorithm for the volume of a convex polyhedron in $\mathbb{R}^n$. *J. of Optimization Theory and Applications*, 39(3):363–377, 1983.

[14] J. B. Lasserre. Integration on a convex polytope. Technical Report 96173, Centre National de la Recherche Scientifique, Laboratoire d'Analyse et d'Architectures des Systems, Toulouse, 1996.

[15] J. Lawrence. Polytope volume computation. *Mathematics of Computation*, 57(195):259–271, 1991.

[16] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6:423–434, 1991.

[17] J. Verschelde, K. Gatermann, and R. Cools. Mixed-volume computation by dynamic lifting applied to polynomial system solving. *Discrete Comput. Geom.*, 16:69–112, 1996.