Efficient volume and edge-skeleton computation for polytopes defined by oracles *

Ioannis Z. Emiris† Vissarion Fisikopoulos† Bernd Gärtner‡

Abstract

We design and implement total polynomial-time algorithms for computing the exact edge-skeleton and for approximating the volume of polytopes given by optimization oracles. That is, the time complexity of the algorithm is bounded by a polynomial in the input and the output size. The main algorithmic step is to obtain efficient separation oracles given an optimization oracle, which is reduced to solving a linear program in the polar polytope. This separation oracle is used to yield polynomial-time Monte Carlo algorithms for approximating the volume of the polytope. Next, we use this separation oracle to derive the first total polynomial-time algorithm for the edge skeleton of the polytope, when we are also given a superset of the polytope's edges, with cardinality bounded by a polynomial in the number of those edges. Finally, we briefly discuss our implementation and experimental results of optimization and volume approximation algorithms, based on random walks.

1 Introduction

Convex polytopes in general dimension admit a number of alternative representations. The best known, explicit representations for a polytope P is either as the set of its vertices (possibly with additional information about the positive-dimensional faces), or as a bounded intersection of halfspaces. In the latter case, a linear programming problem (LP) on P consists in finding a vertex of P that maximizes the inner product with a given objective vector c. In this paper we study the case where a polytope is given by an implicit representation. That is, the only allowable access to P is a black box subroutine (oracle) that solves the LP problem on P for a given vector c. Then, we say that P is given by an *optimization*, or *LP*, or *vertex* oracle. Given such an oracle, the entire polytope can be reconstructed and its explicit representation can be found using the Beneath-Beyond method [2]. This is implemented in [8, 4] for a special case of polytopes, called resultant polytopes.

Another important implicit representation for a polytope P is to be given by a *separation* oracle. That is, given a point x the oracle returns yes if $x \in P$ or a hyperplane that separates P from x otherwise. To acquire an optimization oracle for P, one has to solve a linear program over P, using the separation oracle. This can be done by (variants of) the ellipsoid method (Sect. 2).

Now we pose the opposite (dual) question. Given an optimization oracle for a polytope P, compute a separation oracle for P. This boils down to solving a linear program in the polar dual space where the optimization oracle of P is a separation oracle for the polar polytope P^* (Sect. 2).

Proposition 1 *An approximate separation oracle for a well-rounded polytope $P \subseteq \mathbb{R}^n$, given by an optimization oracle of complexity T, is computed in time $O^*(nT+n^{3.38})$, where $O^*(\cdot)$ hides polylog factors in the argument.*

Well-rounded means that the radii of some bounding and some inscribed ball differ by a constant factor not depending on n. Also, the approximation error is assumed to be constant, so that the bound is simple and only depends on n and T. Prop. 1 is the main algorithmic tool used in the sequel. This leads to our first contribution, namely an implementation of a linear program solver based on the randomized algorithm of [1] (Sect. 4), which is also valuable because of its independent interest.

Regarding the *volume* computation problem, it is known that the exact computation is hard. However, randomized poly-time approximation algorithms exist when the polytope is given by a separation oracle, and currently the best complexity is $O^*(n^4)$ oracle calls [13]. The literature on implementing such algorithms is limited. A notable exception is [12], which implements [13] and computes the volume of n-cubes, $n = 2, 5, 8$, in $807, 1901, 7551$ secs respectively. Our second contribution consist in providing an implemen-

*Work partially supported from project "Computational Geometric Learning", which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the 7th Framework Programme for research of the European Commission, under FET-Open grant number: 255827. This research was partially funded by the University of Athens Special Account of Research Grants no 10812. We also thank K. Fukuda, C. Müller, S. Stich for discussions and bibliographic suggestions.

†Department of Informatics and Telecommunications, University of Athens, Greece. {emiris,vfisikop}@di.uoa.gr

‡Institut für Theoretische Informatik, Swiss Federal Institute of Techology (ETHZ), Zurich, Switzerland. gaertner@inf.ethz.ch

tation, based on the $O^*(n^5)$ algorithm of [9], which is simpler and appears to have competitive performance (Sect. 4). Regarding volume computation for polytopes given by optimization oracle, Prop. 1 is used in conjunction with the above randomized algorithm.

Edge skeleton computation and vertex enumeration from edge directions and suitable oracles is a problem of independent interest. It has been studied before [14]; their solution is used as a subroutine in efficiently solving convex integer programming in fixed dimension [11]. Our third contribution is the design of total poly-time algorithms for computing the exact edge-skeleton of polytopes given by optimization oracles, when we are also given a superset of the polytope's edges, with cardinality bounded by a polynomial in the number of those edges.

Applications. Two applications have motivated this paper. The first is the Minkowski sum $P \subset \mathbb{R}^n$ of polytopes $P_1, \ldots, P_r \subset \mathbb{R}^n$ which are given by the set of their vertices. Here, optimization oracles are naturally and easily available, whereas it is not straightforward to construct the separation oracles. To illustrate this assume we are given a direction. Then, the extremal vertex of each polytope summand can be efficiently computed by computing the interior product of the direction with each vertex of the summand. The extremal vertex of P towards the given direction is the sum of the extremal vertices of the summands. The above can be generalized for *summand polytopes* given by optimization oracles.

Proposition 2 *If $P_1, \ldots, P_r$ are given by optimization oracles, each of complexity bounded by v, then by Prop. 1, a separation oracle for $P = \sum_{i=1}^{r} P_i$ is computed in $O^*(nrv + n^{3.38})$. The edge skeleton of P can be computed in $O^*(m^2 n^{3.5})$ and an approximation of its volume in $O^*(n^{7.38})$.*

Our second application is resultant polytopes. Resultant polynomials are fundamental in algebraic geometry since they generalize determinants to nonlinear systems [6]. The Newton polytope of resultant R, or *resultant polytope*, is the convex hull of the exponent vectors corresponding to nonzero terms. A resultant is defined for $d+1$ pointsets in $\mathbb{Z}^d$. If R lies in $\mathbb{Z}^n$, the total number of input points is $n + 2d + 1$, and we assume that they are in generic position. If m is the number of vertices in R, typically $m \gg n \gg d$, so d is assumed fixed. A poly-time optimization oracle is described in [4]. This approach can be used for the secondary and discriminant polytopes [6].

Proposition 3 *Given an optimization oracle for $R \subset \mathbb{Z}^n$ we can compute the edge-skeleton of $R \subset \mathbb{Z}^n$ in $O^*(m^3 n^{\lfloor (d/2)+1 \rfloor} + m^4 n)$ for input points in generic position, and an approximation of its volume in $O^*(n^{\lfloor (d/2)+5 \rfloor})$, where $d > 5$.*

2 Polytope oracles

We introduce all tools needed to prove Prop. 1. Following [7], we define 5 basic oracles for polytope $P \subset \mathbb{R}^n$ and, for completeness, describe *exact poly-time* procedures that connect them.
Optimization ($OPT_P(c)$): Given $c \in \mathbb{R}^n$, find $x \in P$ maximizing $c^T x, x \in P$, or assert $P = \emptyset$.
Validity ($VAL_P(c)$): Given $c \in \mathbb{R}^n$, decide whether $c^T x \leq 1$ holds for all $x \in P$.
Violation ($VIOL_P(c)$): Given $c \in \mathbb{R}^n$, call $VAL_P(c)$; if negative, find $y \in P \ : \ c^T y > 1$.
Membership ($MEM_P(y)$): Given $y \in \mathbb{R}^n$ decide whether $y \in P$.
Separation ($SEP_P(y)$): Given $y \in \mathbb{R}^n$ call $MEM_P(y)$. If it answers negatively, find the normal to a hyperplane that separates y from P; i.e. $c \in \mathbb{R}^n \ : \ c^T y > \max\{c^T x \mid x \in P\}$.

We use the *polar dual polytope* of P in dual space $(\mathbb{R}^n)^*$, as defined in e.g. [16]:

$$P^* := \{c \in \mathbb{R}^n : c^T x \leq 1, \text{ for all } x \in P\} \subseteq (\mathbb{R}^n)^*,$$

where we assume that the origin $\mathbf{0} \in int(P)$, the relative interior of P, i.e. $\mathbf{0}$ is not contained in any face of P of dimension $< n$. This hypothesis can be ensured by an appropriate affine translation.

It is easy to see that having OPT_P we can derive $VIOL_P$ and then VAL_P. Similarly, having SEP_P we can derive MEM_P. For a polytope $P \subseteq \mathbb{R}^n$, its polar $P^* \subseteq (\mathbb{R}^n)^*$ and $c \in \mathbb{R}^n$, it holds $VIOL_P(c) = SEP_{P^*}(c^T)$ hence $VIOL_{P^*}(c^T) = SEP_P(c)$ [16, Thm 2.11].

Given $VIOL_P(c)$ we compute $OPT_P(c)$ by performing binary search on the value of $c^T x$, for $x \in P$. That is, we test feasibility, or non-emptiness, of $\{x \in P \ : \ c^T x \leq c_0\}$ for various constants c_0, by calling $VIOL(c)$ on suitably chosen translations of P.

Let $B(\rho)$ denotes the n-ball of radius ρ centered at the origin. Assume that $P \subseteq B(\rho)$ and $B(r) \subseteq P$ if P is not empty. Then, let L denote the log-ratio of the bounding balls of P, i.e. $L = \lg(\rho/r)$. Computing $VIOL_P$ from SEP_P is a fundamental question. Some poly-time algorithms for this problem are the ellipsoid method [10] with complexity $O(n^2 L T_S + n^4 L)$ that [15] improves to

$$O(nLT_S + n^{3.38} L) = O^*(nT_S + n^{3.38}) \qquad (1)$$

and the randomized algorithm of [1] which runs in $O(nLT_S + n^7 L)$, where T_S is the complexity of the separation oracle.

For proving Prop. 1 we have to compute a separation oracle for a polytope P, SEP_P, given an optimization oracle for P, OPT_P. As it is explained above this can be done by solving a linear program in the polar dual space. The fastest algorithm used to solve this linear program is [15], with complexity of

Eq. (1). However, we implement the slightly slower but simpler algorithm of [1].

3 Computing the edge skeleton

We are implicitly given a polytope $P \subseteq \mathbb{R}^n$ via an optimization oracle $OPT_P(x)$ of P and we are explicitly given a superset E of all edge directions of P, i.e.

$$E \supseteq D(P) := \{v - w : v, w \text{ adjacent vertices of } P\},$$

with cardinality $|E|$ bounded by a polynomial in $|D(P)|$. The goal is to efficiently compute the edge skeleton of P, i.e. its vertices and the edges connecting the vertices. Even if $E = D(P)$, this set does not in general provide enough information to perform the task, so we need additional information about P; here we assume an optimization oracle.

Proposition 4 [14] *Let $P \subseteq \mathbb{R}^n$ be a polytope given by an optimization oracle $OPT_P(c)$, and let $E \supseteq D(P)$ be a superset of the edge directions of P, With $O(|E|^{n-1})$ arithmetic operations and $O(|E|^{n-1})$ calls to $OPT_P(c)$, all vertices of P can be computed.*

If P has m vertices, then $|D(P)| \leq \binom{m}{2}$, and this is tight for neighborly polytopes in general position. This means that the bound of Prop. 4 is $\Theta\left(m^{2n-2}\right)$ in the worst case.

We improve over this result and compute the edge skeleton with a number of arithmetic operations and calls to $OPT_P(x)$ which are polynomial in m, n, and L^*, the ratio between the radii of balls enclosing and included in the polar P^*. This ratio comes in since we construct a separation oracle $SEP_P(y)$ from $OPT_P(x)$, according to Prop. 1. We therefore get rid of the exponential dependence on n in Prop. 4, but at the cost of an additional dependence on L^* which in general cannot be bounded by m, n. On the other hand, L^* is related to L.

The algorithm is as follows. Using $OPT_P(c)$, we find the unique vertex v of minimum x_n coordinate. We maintain sets V_P, E_P of vertices and edges that have already been found, along with a priority queue W of the vertices that are in V_P. When we process the next vertex v from the queue, it remains to find its incident edges. To find the neighbors of v, we first build a set V_{cone} of candidate vertices. V_{cone} can be constructed using $SEP_P(y)$ that we build from $OPT_P(x)$ as described above. In a final step, we remove the candidates that do not yield vertices. For this, we first solve a linear program to compute a hyperplane separating v from the candidates; w.l.o.g. this hyperplane is $\{x_n = 1\}$. Then we compute the extreme points of $C \cap \{x_n = 1\}$, giving us the extremal rays of C. Finally, we remove every point from V_{cone} that is not on an extremal ray, or not highest on its extremal ray.

We bound the *time complexity* of the algorithm: We call $OPT_P(x)$ to find the first vertex of P. Then, there are $O(m)$ iterations; one iteration calls $SEP_P(y)$ at most $|E|$ times, each call requiring $O(nL^*T + n^{3.38}L^*)$ time, where T is the time to execute $OPT_P(x)$. Then we compute the (at most m) extreme points from a set of at most $|E|$ points, which can be done in

$$O(|E| \cdot \mathrm{LP}^*(n+1, m+1) + n|E|m)$$

time and $O(n|E|)$ space [3], where $\mathrm{LP}^*(a, b)$ is the time to solve a linear program with a variables and b inequality constraints.

Assuming $|E| = O(|D(P)|) = O(m^2)$, we obtain the following result, and since $\mathrm{LP}^*(n, m)$ is polynomial in n, m in the bit model [10], an overall polynomial bound follows.

Theorem 5 *The algorithm runs in time $O(m^3(nL^*T + n^{3.38}L^* + LP^*(n, m)))$, where T is the time to perform one call to $OPT_P(x)$.*

4 Implementation and experiments

We implement optimization and volume computation algorithms based on random walks. The *Hit-and-Run* random walk is used to generate uniformly distributed points in $P \subset \mathbb{R}^n$ in $O^*(n^3)$ per point [13]. In a feasibility problem we have to answer if a given polytope P is empty or compute a feasible point in P. We implement optimization algorithms based on [1] that solves the feasibility problem. The advantage of this algorithm is its simplicity and the re-usage of procedures, such as random walks, in volume computation.

We also implement randomized approximate volume computation algorithms of polytopes given by separation oracles. Moreover, we implement the algorithm in [9] which approximates of the volume of a polytope P given by a separation oracle by computing uniformly distributed points in P. Assuming $B(1) \subseteq P \subseteq B(\rho)$, the algorithm returns an estimation of $\mathrm{vol}(P)$, which lies between $(1-\varepsilon)\mathrm{vol}(P)$ and $(1+\varepsilon)\mathrm{vol}(P)$ with probability $\geq 3/4$, making

$$O\left(\frac{n^4\rho^2}{\varepsilon^2} \ln n \ln \rho \ \ln^2 \frac{n}{\varepsilon}\right) = O^*(n^4\rho^2)$$

oracle calls with probability $\geq 9/10$. Combining the above implementations we also provide an implementation for volume approximation of Minkowski sums. The code is in C++ and is publicly available at `http://sourceforge.net/projects/randgeom`.

We perform an experimental analysis of the above implemented algorithms on an Intel Core i5-2400 3.1GHz, 6MB L2 cache, 8GB RAM, 64-bit Debian GNU/Linux. The optimization algorithms are able to run in less than a minute for up to dimension 11 when

n	$vol(P)$	exact(sec)	r_2	w_s	$(1+\varepsilon)vol(P)$	min	max	μ	σ	appr(sec)
2	4	0.06	2218	8	6.09	3.84	4.12	3.97	0.05	0.23
4	16	0.06	2738	7	30.4	14.99	16.25	15.59	0.32	1.77
6	64	0.09	5308	38	121.6	60.85	67.17	64.31	1.12	39.66
8	256	2.62	8215	16	486.4	242.08	262.95	252.71	5.09	46.83
10	1024	388.25	11370	40	1945.6	964.58	1068.22	1019.02	30.72	228.58
12	4096	–	14725	82	7782.4	3820.94	4247.96	4034.39	80.08	863.72

Table 1: Volume computation for hypercubes; '–': the exact method was unable to compute the volume.

n	$vol(P)$	exact(sec)	r_2	w_s	min	max	μ	σ	appr(sec)
2	14.00	0.01	216	11	12.60	19.16	15.16	1.34	119.00
3	45.33	0.01	200	7	42.92	57.87	49.13	3.92	462.65
4	139.33	0.03	100	7	100.78	203.64	130.79	21.57	721.42
5	412.26	0.23	100	7	194.17	488.14	304.80	59.66	1707.97

Table 2: Volume computation of the Minkowski sum of a hypercube and a crosspolytope.

tested on hypercubes and their polar duals, namely crosspolytopes. The volume approximation algorithm tested on hypercubes and crosspolytopes compute the volume up to dimension 12 within minutes, whereas it is intractable to compute in more than 10 dimensions with exact methods, such as Polymake [5]. For 20 runs, the code's computed values have less than 2% error from the average one. Additionally, the minimum and maximum computed values bounds the exact volume providing tighter bounds than the theoretical ones, i.e. $(1 \pm \varepsilon)\text{vol}(P)$. Table 1 shows experimental results, where r_2 is the number of random points computed, w_s is the number of the steps of a Hit-and-Run random walk, and max, min, μ and σ denote, respectively, the maximum, the minimum, the average and the average absolute deviation of the computed volume approximation.

Finally, we compute an approximation of the volume of the Minkowski sum of a hypercube and a crosspolytope. We perform 10 experiments for each polytope sum. The results show that the min and max computed values bound the exact one. However, we are unable to compute in dimensions higher than 5 since each membership test for P runs one of the optimization algorithms. On the other hand, there is space for improvement in many places of the prototype implementations of optimization and volume computation, which will improve the Minkowski sum volume computation as well. Table 2 shows experimental results with the same notation as in Table 1.

References

[1] D. Bertsimas and S. Vempala. Solving convex programs by random walks. *J. ACM*, 51(4):540–556, 2004.

[2] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[3] K.L. Clarkson. More output-sensitive geometric algorithms. In *Proc. IEEE FOCS*, pages 695–702, 1994.

[4] I.Z. Emiris, V. Fisikopoulos, C. Konaxis, and L. Peñaranda. An output-sensitive algorithm for computing projections of resultant polytopes. In *Proc. ACM Symp. on Comp. Geom.*, pages 179–188, 2012.

[5] E. Gawrilow and M. Joswig. Polymake: an approach to modular software design in computational geometry. In *ACM Symp. on Comp. Geometry*, pages 222–231, 2001.

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

[7] P. Huggins. ib4e: A software framework for parametrizing specialized LP problems. In A. Iglesias and N. Takayama, editors, *Mathematical Software - ICMS*, volume 4151 of *LNCS*, pages 245–247. Springer, 2006.

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

[9] L.G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Doklady*, 20(1):191–194, 1979.

[10] J.A. De Loera, R. Hemmecke, S. Onn, U.G. Rothblum, and R. Weismantel. Convex integer maximization via Graver bases. *J. Pure & Applied Algebra*, 213(8):1569–1577, 2009.

[11] L. Lovász and I. Deák. Computational results of an $O(n^4)$ volume algorithm. *European J. Operational Research*, 216(1):152–161, 2012.

[12] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *J. Comp. Syst. Sci.*, 72(2):392–417, 2006.

[13] S. Onn and U.G. Rothblum. The use of edge-directions and linear programming to enumerate vertices. *J. Combin. Optim.*, 14:153–164, 2007.

[14] P.M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. IEEE FOCS*, pages 338–343, 1989.

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