Efficient Random-Walk Methods for Approximating Polytope Volume

Ioannis Z. Emiris* Vissarion Fisikopoulos*

Abstract

We study the fundamental problem of computing the volume of a convex polytope given as an intersection of linear inequalities. We implement and experimentally evaluate practical algorithms for accurately approximating the polytope's volume in high dimensions (one hundred). Our code is significantly faster than exact computation and more accurate, as well as faster, than existing approximation methods.

1 Introduction

A fundamental problem in discrete and computational geometry is to compute the volume of a convex body in general dimension or, more particularly, of a polytope. In the past 15 years, randomized algorithms for this problem have witnessed a remarkable progress. Starting with the breakthrough poly-time algorithm of [4], subsequent results brought down the exponent on the dimension from 27 to 4 [10]. However, the question of a practical implementation that handles general polytopes in high dimensions (a few hundred) had remained open. We tackle this question here by offering fast, accurate, public-domain software.

Convex bodies are typically given by a *membership oracle*, i.e. given point p, the oracle decides whether p lies in the body. A polytope $P \subseteq \mathbb{R}^d$ can also be represented as the convex hull of vertices (V-polytope) or, as is the case here, as the (bounded) intersection $P := \{x \in \mathbb{R}^d \mid Ax \leq b\}$ of m halfspaces given by $A \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$ (H-polytope); ∂P is its boundary.

Volume computation is #-P hard for V- and for H-polytopes [5]. Several exact algorithms are surveyed in [2] and implemented in `VINCI`, the state-of-the-art software for exact volume computation, which, however, cannot handle general polytopes for dimension $d > 15$. An interesting challenge is the volume of the n-Birkhoff polytope, computed only for $n \leq 10$ using highly specialized software (Sect. 3).

The landmark randomized poly-time algorithm in [4] approximates the volume of a convex body with high probability and arbitrarily small relative error. The best complexity, as a function of d, given a membership oracle, is $O^*(d^4)$ oracle calls [10], while a simpler, more geometric algorithm, which we shall use, requires $O^*(d^5)$ calls [7]. $O^*(\cdot)$ hides polylog factors in the argument. All approaches except [10] produce uniform point samples in successively larger convex subsets of P so as to approximate their volume.

Concerning software, in [9] they implement [10], focusing on variance-decreasing techniques, and an empirical estimation of mixing time. Very recently, a randomized algorithm in `Matlab` has been announced [1]; we show our software is faster and more accurate in sect. 3.

The key ingredient of all approaches is sampling points (almost) uniformly distributed in P. No simple sampling exists unless P is, e.g., a simplex, cube, or ellipsoid. Acceptance-rejection techniques are inefficient in high dimensions: the number of uniform points needed in a bounding box before finding one in P is exponential in d. A Markov chain is the only known method, and it may use *(geometric) random walks* such as the grid walk, the ball walk, and Hit-and-run [12]. The chain makes a large number of steps, before the generated point becomes distributed approximately uniformly. We focus on Hit-and-run which yields the fastest algorithms today.

Our contributions are multifold. Concerning point sampling we experimentally determine and set the length W of the random walk to $O(d)$ which is much lower than the theoretical bounds and obtain a $< 1\%$ error in up to 100 dimensions (Sect. 3). Our emphasis is to exploit the underlying geometry. Our algorithm uses a sequence of co-centric balls, and samples points in their intersections with P (Sect. 2). Unlike previous methods, this forms a sequence of *diminishing* radii thus allowing us to only sample *partial generations* of points in each intersection with P, instead of sampling N points for each. Unlike most theoretical approaches, that use involved rounding procedures, we use a new simple method for *iterative rounding* that allows us to handle skinny polytopes efficiently (Sect. 2). Utilizing Coordinate Direction Hit-and-run, we design an oracle with $O(m)$ amortized complexity (Sect. 2). We offer a series of experiments establishing that our code handles dimensions substantially larger than existing exact approaches, e.g., cubes and products of simplices within an error

*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece. {emiris,vfisikop}@di.uoa.gr. This work is co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS - UOA (MIS 375891).

[1] http://www.cc.gatech.edu/~bcousins/Volume.html

of 1% for $d \leq 100$, in about 20 min. This paper is an extended abstract of [6], which additionally offers a study that exploits duality to reduce the oracle to ε-nearest neighbor search.

2 The volume algorithm

Random walks, oracles, and sampling. Assume we possess procedure $\texttt{Line}(p)$, which returns line ℓ through point $p \in P \subseteq \mathbb{R}^d$; ℓ will be specified below. The main procedure of Hit-and-run is $\texttt{Walk}(p, P, W)$, which reads in point $p \in P$ and repeats W times: (i) run $\texttt{Line}(p)$, (ii) move p to a random point uniformly distributed on $P \cap \ell$. We shall consider two variants of Hit-and-run.

In *Random Directions Hit-and-run* (RDHR), $\texttt{Line}(p)$ returns ℓ defined by a random vector uniformly distributed on the unit sphere centered at p. The vector coordinates are drawn from the standard normal distribution. RDHR generates a uniformly distributed point in $10^{30}O^*(d^2r^2)$, or $10^{11}O^*(d^3r^2)$ oracle calls starting at an arbitrary, or at a uniformly distributed point (aka warm start), respectively, where r is the ratio of the radius of the smallest enclosing ball over that of the largest enclosed ball in P, and $O^*(\cdot)$ hides no constant [10].

In *Coordinate Directions Hit-and-run* (CDHR), $\texttt{Line}(p)$ returns ℓ defined by a random vector uniformly distributed on the set $\{e_1, \ldots, e_d\}$, where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, $i = 1, \ldots, d$. This is a continuous variant of the Grid walk. As far as the authors know, the mixing time has not been analyzed. In [6] we offer experimental evidence that CDHR is faster than RDHR and sufficiently accurate.

In contrast to other walks, Hit-and-run requires at every step the intersection of line ℓ with ∂P. In general, this reduces to binary search on the line, calling the membership oracle at every step. For H-polytopes, the intersection is obtained by a *boundary oracle*; for this, we employ ray-shooting with respect to the m facet hyperplanes. We focus on CDHR, therefore we may suppose ℓ is vertical. Let us consider $\texttt{Walk}(p_0, P, W)$ and vertical line $\ell = \{x \in \mathbb{R}^d : x = \lambda v + p_0\}$, where $p_0 \in \mathbb{R}^d$ lies on ℓ, and v is the vertical direction. We compute the intersection of ℓ with the i-th hyperplane $a_i x = b_i$, $a_i \in \mathbb{R}^d, b_i \in \mathbb{R}$, namely $p_i := p_0 + \frac{b_i - a_i p_0}{a_i v} v$, $i \in \{1, \ldots, m\}$. We seek points p^+, p^- at which ℓ intersects ∂P, namely $p^+ v = \min_{1 \leq i \leq m}\{p_i v \mid p_i v \geq 0\}$, $p^- v = \max_{1 \leq i \leq m}\{p_i v \mid p_i v \leq 0\}$. This is computed in $O(md)$ arithmetic operations. In practice, only the $\lambda^\pm$ are computed, where $p^\pm = p_0 + \lambda^\pm v$. However, in CDHR, after the computation of the first pair p^+, p^-, all other pairs can be computed in $O(m)$ arithmetic operations. This is because two sequential points produced by the walk differ only in one coordinate.

Given polytope $P \subseteq \mathbb{R}^d$ and approximation factor $\epsilon > 0$, the volume algorithm executes *sandwiching* and *Multiphase Monte Carlo* (MMC) [11].

Rounding and sandwiching. There is an abundance of methods in literature for sandwiching (cf. [11] and references therein). The goal is to compute ball B and scalar $\rho > 1$ such that $B \subseteq P \subseteq \rho B$. However, here we develop a simpler method that instead of computing ρ such that ρB covers P, computes B' such that $B' \cap P$ contains almost all the volume of P. Our method handles efficiently skinny polytopes where ρ is large (Sect. 3).

To this end we perform *rounding* of P. We sample a set S of $O(n)$ random points in P using the random walk methods described above. Then we approximate the minimum volume ellipsoid $\mathcal{E}$ that covers S, and satisfies the inclusions $\frac{1}{(1+\varepsilon)d}\mathcal{E} \subseteq \text{conv}(S) \subseteq \mathcal{E}$, in time $O(nd^2(\varepsilon^{-1} + \ln d + \ln \ln n))$ [8]. Let us write

$$\mathcal{E} = \{x \in \mathbb{R}^d \,|\, (x - c_{\mathcal{E}})^T\, E\, (x - c_{\mathcal{E}}) \leq 1\} \qquad (1)$$

where $E \subseteq \mathbb{R}^{d \times d}$ is a positive semi-definite matrix and $L^T L$ its Cholesky decomposition. By substituting $x = (L^T)^{-1} y + c_{\mathcal{E}}$ we map the ellipsoid to the ball $\{y \in \mathbb{R}^d \,|\, y^T y \leq 1\}$. Applying this transformation to P we have $P' = \{y \in \mathbb{R}^d \,|A(L^T)^{-1} \leq b - Ac_{\mathcal{E}}\}$ which is the rounded polytope, where $\text{vol}(P) = \det(L^T)^{-1}\text{vol}(P')$. We iterate this procedure until the variance of the set of ellipsoid axes reaches some user-defined threshold.

For *sandwiching* P we first compute the *Chebychev ball* $B(c, r)$ of P, i.e. the largest inscribed ball in P. It suffices to solve the LP: $\{$maximize R, subject to: $A_i x + R\|A_i\|_2 \leq b_i$, $i = 1, \ldots, m$, $R \geq 0\}$, where A_i is the i-th row of A, and the optimal values of R and $x \in \mathbb{R}^d$ yield, respectively, the radius r and the center c of the Chebychev ball.

Then we may compute a uniform random point in $B(c, r)$ and use it as a start to perform a random walk in P, eventually generating N random points. Now, set ρ to be the largest distance between each of the N points and c; this defines a (approximate) bounding ball. Finally, define the sequence of balls $B(c, 2^{i/d})$, $i = \alpha, \alpha + 1, \ldots, \beta$, where $\alpha = \lfloor d \log r \rfloor$ and $\beta = \lceil d \log \rho \rceil$.

Multiphase Monte Carlo (MMC). MMC constructs a sequence of bodies $P_i := P \cap B(c, 2^{i/d})$, $i = \alpha, \alpha + 1, \ldots, \beta$, where $P_\alpha = B(c, 2^{\alpha/d}) \subseteq B(c, r)$ and P_β (almost) contains P. Then it approximates $\text{vol}(P)$ by the telescopic product $\text{vol}(P_\alpha) \prod_{i=\alpha+1}^{\beta} \frac{\text{vol}(P_i)}{\text{vol}(P_{i-1})}$, where $\text{vol}(P_\alpha) = 2\pi^{d/2}(2^{\lfloor \log r \rfloor})^d / d\Gamma(d/2)$.

This reduces to estimating the ratios $\text{vol}(P_i)/\text{vol}(P_{i-1})$, which is achieved by generating N uniformly distributed points in P_i and by counting how many of them fall in P_{i-1}.

For point generation we use random walks as in Sect. 2. We set the walk length $W = \lfloor 10 + d/10 \rfloor =$

$O(d)$, which is of the same order as in [9] but significantly lower than theoretical bounds. This choice is corroborated experimentally (Sect. 3).

Unlike typical approaches, which generate points in P_i for $i = \alpha, \alpha+1, \ldots, \beta$, here we proceed inversely. First, we generate an (almost) uniformly distributed random point $p \in P_\alpha$, which is easy since $P_\alpha = B(c, 2^{\alpha/d}) \subseteq B(c,r)$. Then we use p to start a random walk in $P_\alpha, P_{\alpha+1}, P_{\alpha+2}$ and so on, until we obtain a uniformly distributed point in P_β. We perform N random walks starting from this point to generate N (almost) uniformly distributed points in P_β and then count how many of them fall into $P_{\beta-1}$. This yields an estimate of $\text{vol}(P_\beta)/\text{vol}(P_{\beta-1})$. Next we keep the points that lie in $P_{\beta-1}$, and use them to start walks so as to gather a total of N (almost) uniformly distributed points in $P_{\beta-1}$. We repeat until we compute the last ratio $\text{vol}(P_{\alpha+1})/\text{vol}(P_\alpha)$.

The implementation is based on a data structure D that stores the random points. In step $i > \alpha$, compute $\text{vol}(P_{\beta-i})/\text{vol}(P_{\beta-i-1})$ and D contains N random points in $P_{\beta-i+1}$ from the previous step. The computation in this step consists in removing from D the points not in $P_{\beta-i}$, then sampling $N - size(D)$ new points in $P_{\beta-i}$ and, finally, counting how many lie in $P_{\beta-i-1}$. Testing whether such a point lies in P_i reduces to testing whether $p \in B(2^{i/d})$ because $p \in P$.

One main advantage of our method is that it creates partial generations of random points for every new body P_i, as opposed to having always to generate N points. This has a significant effect on runtime since it reduces it by a constant raised to β.

Complexity. Our algorithm is in the family of algorithms using a sequence of concentric balls, like [7]. Assuming $B(1) \subseteq P \subseteq B(\rho)$, the algorithm of [7] returns an estimation of $\text{vol}(P)$, which lies between $(1-\epsilon)\text{vol}(P)$ and $(1+\epsilon)\text{vol}(P)$ with probability $\geq 3/4$, making $O\left(\frac{n^4\rho^2}{\epsilon^2}\ln n \ln \rho \, \ln^2 \frac{n}{\epsilon}\right) = O^*(n^4\rho^2)$ oracle calls with probability $\geq 9/10$, by setting $N = 400\epsilon^{-2} d \log d$, which we shall also use here.

Lemma 1 *Given H-polytope P, the volume algorithm performs k phases of rounding in $O^*(d^3mk)$, and approximates $\text{vol}(P)$ in $O(md^3 \log d \log(\rho/r))$ arithmetic operations, assuming $\epsilon > 0$ is fixed, where r and ρ denote the radii of the largest inscribed ball and of the co-centric ball covering P.*

Proof. We generate $d \log(\rho/r)$ balls. In each ball intersected with P, we generate $\leq N = 400\epsilon^{-2} d \log d$ random points, in $W = O(d)$ steps of CDHR for each. The amortized complexity of a CDHR step is $O(m)$, since k CDHR steps require $O(dm + (k-1)m + kd)$ ops and $d = O(m)$, $k = \Omega(d)$.

Each rounding iteration runs in $O(nd^2(\varepsilon^{-1} + \ln d + \ln\ln(n)))$, where n stands for the number of sampled points, and ε is the approximation of the minimum volume ellipsoid of Eq. (1). We generate $n = O(d)$ points, each in $O(m)$ arithmetic operations. Cholesky decompositions takes $O(d^3)$. Hence, rounding runs in $O^*(d^3mk)$, where ε is fixed. ☐

3 Experiments

We implement and experimentally test[2] the above algorithms and methods in the software package `VolEsti`. The code is open-source (sourceforge), in C++. It relies on `CGAL` [3] for its d-dimensional kernel to represent objects such as points and vectors, for its LP solver, for the approximate minimum ellipsoid, and for generating random points in balls. Arithmetic uses the `double` data type of C++. The following polytopes are tested:

- cube-d: $\{x = (x_1, \ldots, x_d) \,|\, x_i \leq 1, x_i \geq -1, x_i \in \mathbb{R}$ for all $i = 1, \ldots, d\}$,
- cross-d: cross polytope, the dual of cube, i.e. $\text{conv}(\{-e_i, e_i,\ i = 1, \ldots, d\})$,
- rh-d-m: polytopes constructed by randomly choosing m hyperplanes tangent to the sphere,
- ccp-n: complete cut polytopes on n vertices,
- Fm-d: one facet of the metric polytope [3] in $\mathbb{R}^d$,
- Δ-d: the d-dimensional simplex $\text{conv}(\{e_i,$ for $i = 0, 1, \ldots, d\})$,
- Δ-d-d: product of two simplices, i.e $\{(p, p') \in \mathbb{R}^{2d} \,|\, p \in \Delta\text{-}d, p' \in \Delta\text{-}d\}$,
- s-cube-d: $\{x = (x_1, \ldots, x_d) \,|\, x_1 \leq 100, x_1 \geq -100, x_i \leq 1, x_i \geq -1, x_i \in \mathbb{R}\ i = 2, \ldots, d\}$, rotated by 30^o in the plane defined by the first two coordinate axes,
- $\mathcal{B}(n)$: the n-Birkhoff polytope (defined below).

Each experiment is repeated 100 times. We keep track of the *min* and the *max* computed values, the mean μ, the standard deviation and the mean error of approximation $(\text{vol}(P)\text{-}\mu)/\text{vol}(P)$. Our method is more accurate than indicated by the theoretical bounds in [7]. In particular, in all experiments all computed values are contained in the interval $((1-\epsilon)\text{vol}(P), (1+\epsilon)\text{vol}(P))$, while theory guarantees only 75% of them. In general our experimental results show that our software can approximate the volume of general polytopes up to dimension 100 in less than 2 hours with mean approximation error at most 2% (cf. Table 1).

We set $W = \lfloor 10 + d/10 \rfloor$. Our experiments indicate that, with this choice, $(\text{vol}(P)\text{-}\mu)/\text{vol}(P)$ is $< 2\%$ up to $d = 100$ (Table 1). Moreover, for higher W the improvement in accuracy is not significant, which supports the claim that asymptotic bounds are unrealistically high.

[2]All timings are on an Intel Core i5-2400 3.1GHz, 6MB L2 cache, 8GB RAM, 64-bit Debian GNU/Linux.

[3]Comp. geometry algorithms library, http://www.cgal.org

P	d	m	vol(P)	N	μ	$[min,\ max]$	std-dev	$\frac{\text{vol}(P)-\mu}{\text{vol}(P)}$	`VolEsti` (sec)	`exact` (sec)
cube-10	10	20	1.024E+03	9210	1.027E+03	[0.950E+03,1.107E+03]	3.16E+001	0.0030	0.42	0.01
cube-15	15	30	3.277E+04	16248	32477.93	[3.037E+04,3.436E+04]	941.68	0.0088	1.44	0.40
cube-20	20	40	1.048E+06	23965	1.046E+06	[0.974E+06,1.116E+06]	3.15E+004	0.0028	4.62	swap
cube-50	50	100	1.126E+15	78240	1.125E+15	[1.003E+15,1.253E+15]	4.39E+013	0.0007	117.51	swap
cube-100	100	200	1.268E+30	184206	1.278E+30	[1.165E+30,1.402E+30]	4.82E+028	0.0081	1285.08	swap
Δ-50	60	61	1.202E-082	98264	1.21E-082	[1.07E-082,1.38E-082]	6.44E-084	0.0068	183.12	0.01
Δ-100	100	101	1.072E-158	184206	1.07E-158	[9.95E-159,1.21E-158]	4.24E-160	0.0032	907.52	0.02
Δ-40-40	80	82	1.502E-096	140224	1.50E-096	[1.32E-096,1.70E-096]	7.70E-098	0.0015	452.05	0.01
Δ-50-50	100	102	1.081E-129	184206	1.10E-129	[1.01E-129,1.19E-129]	4.65E-131	0.0154	919.01	0.02
cross-10	10	1024	2.822E-04	9210	2.821E-04	[2.693E+04,2.944E+04]	5.15E-06	0.0003	1.58	388.50
cross-18	18	262144	4.09E-011	20810	4.027E-11	[3.97E-11,4.08E-11]	5.58E-013	0.0165	5791.06	—
Fm-5	10	25	7.110E+03	9210	7.116E+03	[6350.72,8103.78]	3.01E+002	0.0009	0.69	0.02
Fm-6	15	59	2.861E+05	16248	2.850E+05	[241851.70,321864.30]	1.55E+004	0.0038	3.24	swap
ccp-5	10	56	2.312E+00	9210	2.326E+00	[2.159411,2.527959]	7.43E-02	0.0064	0.49	38.23
ccp-6	15	368	1.346E+00	16248	1.346E+00	[1.264474,1.451714]	3.81E-02	0.0002	6.14	swap
$\mathcal{B}_9$	64	81	2.60E-33	425869	2.62E-33	[2.33E-33, 2.83E-33]	3.01E-42	0.0098	1674.09	8 days
$\mathcal{B}_{10}$	81	100	8.78E-46	569520	8.34E-46	[8.097E-46, 9.19E-46]	3.08E-56	0.0189	3382.46	6160 days
$\mathcal{B}_{11}$	100	121	???	736827	1.14E-60	[9.61E-61, 1.21E-60]	6.90E-62	???	6204.84	–
$\mathcal{B}_{12}$	121	144	???	928465	5.54E-78	[4.41E-78, 6.39E-78]	7.24E-79	???	11322.23	–
s-cube-10	10	20	1.024E+05	9210	5.175E+04	[2.147E+04,1.228E+05]	18286.0	0.4946	0.69	0.01
r-s-cube-10	10	20	1.024E+05	9210	1.029E+05	[8.445E+04,1.149E+05]	5312.3	0.0050	0.71	0.01
s-cube-20	20	40	1.049E+08	23965	4.193E+07	[2.497E+07,7.259E+07]	9268346.0	0.6001	5.59	swap
r-s-cube-20	20	40	1.049E+08	23965	1.040E+08	[8.458E+07,1.163E+08]	6615707.0	0.0084	6.70	swap

Table 1: Overall results; $\epsilon = 1$, "swap" indicates it ran out of memory and started swapping; r-s-cube denotes that we use rounding; "???" indicates that the exact volume is unknown; "–" indicates it didn't terminate after at least 10h. `VINCI` is used for exact volume computation except Birkhoff polytopes where `birkhoff` is used instead.

To experimentally test the effect of rounding we construct skinny hypercubes s-cube-d. We rotate them to avoid CDHR taking unfair advantage of the degenerate situation where the long edge is parallel to an axis. Table 1 shows that rounding reduces approximation error by 2 orders of magnitude.

We test against `VINCI` 1.0.5 [2] (Table 1). For all inputs, there is a threshold dimension for which `VINCI` takes too much time (e.g. > 4 hrs for cube-20) and consumes all system memory, thus starts swapping.

Testing the most relevant approximation method implemented in `Matlab` (cf. Sect. 1) with default options and $\epsilon = 0.1$, our implementation runs at least 2 times faster and returns significantly more accurate results, e.g. from 4 to 100 times smaller error on cube-d when $d > 70$, and from 5 to 80 times on Birkhoff polytopes (Table 1). For example, for $\mathcal{B}_{10}$ `Matlab` and `VolEsti` compute $\frac{\text{vol}(P)-\mu}{\text{vol}(P)} = 0.1207$ and 0.0207 in 3437.74 and 2660.20 secs respectively and for cube-100 compute $\frac{\text{vol}(P)-\mu}{\text{vol}(P)} = 0.0357$ and 0.0081 in 3805.37 and 1285.08 secs respectively.

The n-th *Birkhoff polytope* $\mathcal{B}_n = \{x \in \mathbb{R}^{n\times n} \mid x_{ij} \geq 0,\ \sum_i x_{ij} = 1,\ \sum_j x_{ij} = 1,\ 1 \leq i \leq n\}$, is also described as the polytope of the perfect matchings of the complete bipartite graph $K_{n,n}$. In [1], they present a complex-analytic method for this volume, implemented in package `birkhoff`, which has managed to compute vol$(\mathcal{B}_{10})$ in parallel execution, which corresponds to a single processor running at 1 GHz for almost 17 years. Our software computes the volume of polytopes up to $\mathcal{B}_{10}$ in < 1 hour with mean error of $\leq 2\%$ (Table 1). We decrease ϵ and obtain an error of 0.7% for vol$(\mathcal{B}_{10})$, in 6 hours, i.e., with two correct digits. More interestingly, using $\epsilon = 0.5$ we compute an approximation as well as an interval of values for vol$(\mathcal{B}_{11})$, vol$(\mathcal{B}_{12})$, whose exact values are unknown (Table 1).

References

[1] M. Beck and D. Pixton. The Ehrhart polynomial of the Birkhoff polytope. *DCG*, 30(4):623–637, 2003.

[2] B. Büeler, A. Enge, and K. Fukuda. Exact volume computation for polytopes: A practical study. In *Polytopes: Combinatorics and Computation*, volume 29, pages 131–154. Birkhäuser, 2000.

[3] A. Deza, M. Deza, and K. Fukuda. On skeletons, diameters and volumes of metric polyhedra. LNCS, pages 112–128. Springer, 1996.

[4] M. Dyer, A. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

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

[6] I.Z. Emiris and V. Fisikopoulos. Efficient random walk methods for approximating polytope volume, 2013. In arXiv:1312.2873.

[7] 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.

[8] L.G. Khachiyan. Rounding of polytopes in the real number model of computation. *Math. Oper. Res.*, 21(2):307–320, 1996.

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

[10] 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.

[11] M. Simonovits. How to compute the volume in high dimension? *Math. Program.*, pages 337–374, 2003.

[12] R.L. Smith. Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.