Project number IST-25582

CGL

Computational Geometric Learning

Algorithms for volume approximation of convex bodies

STREP

Information Society Technologies

Period covered:	November 1, 2012 – October 31, 2013
Date of preparation:	November 9, 2013
Date of revision:	November 9, 2013
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

Algorithms for volume approximation of convex bodies

Ioannis Z. Emiris* Vissarion Fisikopoulos *

November 9, 2013

Abstract

We survey results on the fundamental problem of computing the volume of d-dimensional convex bodies, with emphasis on randomized poly-time approximation algorithms for bodies represented by a Membership oracle. We implement and experimentally study efficient algorithms for approximating the volume of polytopes given as an intersection of halfspaces, by developing efficient hit-and-run methods. Our emphasis is to exploit the geometry of the problem so as to improve runtime (or accuracy) for general dimensional polytopes. Our publicly available C++ software is the first to handle polytopes in dimensions substantially larger than exact volume computation software can do, e.g., it approximates the volume of d-dimensional cubes with at least 3 correct digits, for d up to 100, in about 20 min, whereas state-of-the-art exact software VINCI does not seem able to handle d-cubes for $d > 20$.

Keywords: volume computation, randomized algorithms, algorithmic engineering

1 Introduction

A fundamental problem in discrete and computational geometry is to compute the volume of a convex body $K \subseteq \mathbb{R}^d$ or, more particularly, of a polytope. In the past 15 years, randomized algorithms for this problem have witnessed a remarkable progress: starting with a breakthrough polynomial-time algorithm, subsequent results brought down the exponent on the dimension from 27 to 4, as discussed below.

Convex bodies may be given by a membership oracle, which is typical in randomized approximation algorithms for volume computation. A polytope $P \subseteq \mathbb{R}^d$ can also be represented as the convex hull $P := \text{conv}\{v_1, \ldots, v_n\}$ of vertices $v_i \in \mathbb{R}^d$ (V-polytope) or, equivalently, as the (bounded) intersection $P := \{x \in \mathbb{R}^d \mid Ax \leq b\}$ of halfspaces given by $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$ (H-polytope). Let A_i be the i-th row of A and b_i the i-th element of b.

Computing the volume of a body is an extremely difficult task. In fact, there are "impossibility" results in this direction: in [5], improving a result of [18], they proved that if the convex body is given by a Separation oracle, then any deterministic algorithm that approximates the volume within a factor of $d^{O(d)}$ necessarily takes exponential time. Dyer and Frieze [15], and Khachiyan [22, 23] showed that the problem of computing the volume exactly (deterministically) is $\#P-$hard, even for explicitly described polytopes.

A breakthrough in the opposite direction is due to Dyer, Frieze and Kannan (DFK) [14], who designed a *fully polynomial randomized approximation scheme* (FPRAS) for approximating the volume of a convex body $K \subset \mathbb{R}^d$. Their ϵ-*approximation* algorithm with parameters ϵ, $\delta > 0$ be small positive numbers, compute a random variable ζ such that with the probability at $1-\delta$, the volume of K is

$$(1-\epsilon)\zeta < \text{vol}(K) < (1+\epsilon)\zeta,$$

*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Greece. emiris,vfisikop@di.uoa.gr

where vol$(\cdot)$ denotes Euclidean volume. So randomization brings exponential approximation down to an arbitrarily small one.

Let us offer a short survey of those ideas whose various combinations lead to many improvements. The DFK algorithm consists of two main phases: the first phase makes the body "reasonably" round by applying an affine transformation to K so that the image contains the unit ball B and is contained in a ball of radius R. The second phase is the construction of a series of bodies $K_0 = B \subseteq K_1 \subseteq \cdots \subseteq K_m = K$, where we start with a ball $K_0 = B$, or some body whose volume is known, and the volume of the convex body $K_m = K$ is the one we wish to compute. Next, we have to estimate the ratio of two subsequent convex bodies $\mathrm{vol}(K_i)/\mathrm{vol}(K_{i-1})$ and we repeat this for each $i = 1, \ldots, m$. To compute this ration, it is enough to generate sufficiently many independent (almost) uniformly distributed random points in K_i and counting how often K_{i-1} is hit. The product of these estimations is an estimate of $\mathrm{vol}(K_0)/\mathrm{vol}(K)$.

The Markov chain is the only known technique for producing uniform points inside a convex body, using geometric random walks such that *grid* walk, *ball* walk or *hit-and-run* technique. How many number of steps the Markov chain needs to generate point according to the uniform distribution? The necessary number of steps to reach the stationary distribution is called the *mixing time.* In order to decide when we are likely to close to the stationary distribution, conductance is used to bound the *mixing time,* and isoperimetric inequalities are used to bound the conductance. Most algorithms follow DFK and use a *Multiphase Monte-Carlo Markov chain* to reduce volume computation to point sampling; the latter is achieved by random walks.

Let $O^*(\cdot)$ denote an upper bound on asymptotic complexity where we hide polylog factors in the argument. The algorithm of DFK [14] was in $O^*(d^{23})$. Since that time, many improvements have been made: In [26], Lovàsz and Simonovits gave an algorithm with complexity $O^*(d^{16})$, by proving an appropriate isoperimetric inequality based on the idea that integral inequalities in $\mathbb{R}^d$ can be reduced to one-dimensional inequalities. In [3], Applegate and Kannan obtained an algorithm with complexity $O^*(d^{10})$, by introducing the integration of log-concave functions. Next, Lovàsz introduced the ball walk in [25], obtaining an algorithm with complexity $O^*(d^{10})$. In [16], Dyer and Frieze obtained an algorithm with complexity $O^*(d^8)$, still using a grid walk with better error analysis. In [29], Lovàsz and Simonovits gave an algorithm in $O^*(d^7)$, using ball walk and the Metropolis algorithm. In [21], Kannan, Lovàsz and Simonovits, using the isotropic position for sandwiching, obtained an algorithm with complexity $O^*(d^5)$; here, both sandwiching and volume approximation admit the same asymptotic complexity.

In [30], Lovàsz and Vempala obtain $O^*(d^4)$ by constructing, instead of a "predetermined" sequence of bodies, a sequence of log-concave functions $f_0 \leq f_1 \leq \cdots \leq f_m$. The integral of f_0 over enlarged body K' can be easily determined, and the integral of f_m is the desired volume. In previous algorithms one computes ratios $\mathrm{vol}(K_i)/\mathrm{vol}(K_{i-1})$, whereas now one compute ratios of integrals $\int_{K_i} f_{i-1} / \int_{K_i} f_i$. By adapting the set of f_i chosen, the algorithm uses only $O^*(\sqrt{d})$ points per phase and $m = O^*(\sqrt{d})$ phases; the latter relies also on better sandwiching by an outer and inner ellipsoid. However, $O^*(d^3)$ still bounds the mixing time to sample each point. Moreover, they improve time for isoperimetric sandwiching to $O^*(d^4)$. As Vempala writes in his survey [34]: *"it is apparent that any improvement in the mixing rate of random walks will directly affect the complexity of volume computation. Such improvements seem to consistently yield interesting new mathematics as well".* For further details, see [9, 31, 34].

However, the literature on implementing randomized algorithms is quite limited. A notable exception is [28] implementing [30]. They offer variance decreasing statistical techniques, and an empirical estimation of the mixing time of the hit-and-run walk, which is faster than the theoretical upper bound. Since they aim at general convex bodies given by a membership oracle, their experiments consider only cubes up to $d = 8$, which take 7551 sec. They report they "could not experiment with other convex bodies than cubes, because the oracle describing the convex

bodies took too long to run".

In [24] they use a simple acceptance/rejection direct Monte-Carlo method, and compute a unique ball that lies completely in the convex body. Then, they generate random points in the body and count how many of them fall into the ball to estimate the ratio of the volumes of the body and the ball. They use fewer points than the theoretical estimate but without any experimental or heuristic study on their cardinality. Moreover, their pseudo-random number generator does not guarantee the uniform distribution of the points. Hence, their approach is not expected to work accurately in general dimension. The implementation has been tested only up to dimension 4.

In [1], the authors analyze 3 polytope samplers for polytopes in V-representation. The first, which is implemented in a straightforward manner (hence cannot run in high dimensions), applies Dirichlet's uniform distribution on an r-dimensional simplex, then projects it appropriately to the polytope defined by r vertices. They claim that statistical testers cannot distinguish this distribution from the exact uniform one. The second relies on a triangulation of the polytope and is thus impractical. The third applies an appropriate distribution (e.g. the Gamma distribution) on a sphere or simplex of same dimension as P, then maps the points to P by the moment map. However, the Jacobian of this map is hard to compute and the distribution difficult to control.

Good point samples in a polytope is a problem of independent interest. One application is to infer ill-posed linear inverse problems, a key task in machine learning, e.g., [8, 33]. The polytope sampler of [1] offers a feasible strategy to tackle the problem of sampling 2-way and multi-way contingency tables. It is also reported to be more than 60x faster and also more precise than the existing Metropolis-Hastings approach on the network tomography problem [33]. The polytope sampler yields 100K points in about 1 min, in a 7-dimensional V-polytope in $\mathbb{R}^{16}$. The points' average distance to the ground truth points is about 17 Mb and 6 Mb in the L_1 and L_2 norm, respectively.

In [35], in order to study the solution space containing steady-state flux distributions defined by a polytope, they compute its volume exactly, for $d < 10$. In higher dimensions, they sample points by a MC acceptance-rejection method on the polytope and a bounding box. Even this becomes inefficient in large dimensions (they only report they found 1 polytope point per 5000 points in the bounding box). Thus they try to "narrow the bounding box to a tighter fitting parallelepiped so that a higher fraction of random polytopes can be found" (sic). can be found" (sic). In [2] they use a hit-and-run method to sample from polytopes although they do not talk a lot about geometry and polytopes.

Producing uniformly distributed random points in a convex body is thus a fundamental question, and no simple method exists unless the body has standard shape, e.g., simplex, cube or ellipsoid. Acceptance/rejection techniques are out of the question, since they rapidly become inefficient, a side effect of dimensional explosion. A Markov chain is the only known way, which could use ball walk, grid walk, or hit-and-run technique. Generally the Markov chain has to make a (great) number of steps, before the generated point becomes distributed approximately according to the uniform distribution (which is the stationary limit distribution of the chain). We focus on hit-and-run which yields the fastest algorithms today.

The complexity of the membership oracle is crucial. In the case of convex polytopes it depends on the representation. For an exact oracle, data structures of linear size answer membership queries in time that tends to linear in the number of halfspaces, as dimension grows. Approximate membership oracles are connected with polytope approximation. Classic techniques [17, 11] show that $O((\delta/\varepsilon)^{(d-1)/2})$ facets (or vertices) suffice to approximate a convex body $K \subset \mathbb{R}^d$, where ε is the Hausdorff distance between the approximation and K, δ is its diameter, and d is fixed. This is improved to $O(\sqrt{a}/\varepsilon^{(d-1)/2})$ [4], which is worst-case opti-

mal, where a is the volume of the boundary of K. Given an H-polytope this bound implies space-time tradeoffs for approximate membership oracles ranging from $O(1/\varepsilon^{(d-O(1))/8})$ time and $O(1/\varepsilon^{(d-O(1))/2})$ space to $O(1)$ time and $O(1/\varepsilon^{(d-O(1))})$ space.

Contribution. We survey results on the fundamental problem of computing the volume of d-dimensional convex bodies, with emphasis on randomized poly-time approximation algorithms for bodies represented by a Membership oracle. We implement and experimentally study efficient algorithms for approximating the volume of polytopes given as an intersection of halfspaces, by developing efficient hit-and-run methods. Our emphasis is to exploit the geometry of the problem so as to improve runtime (or accuracy) for general dimensional polytopes. Our publicly available C++ software is the first to handle polytopes in dimensions substantially larger than exact volume computation software can do, e.g., it approximates the volume of d-dimensional cubes with at least 3 correct digits, for d up to 100, in about 20 min, whereas state-of-the-art exact software VINCI does not seem able to handle d-cubes for $d > 20$. This is current work hence, in the near future, we expect to be able to report on further results.

2 Oracles and samplers

Following [20], let us define 3 basic oracles for polytope $P \subseteq \mathbb{R}^d$ which are used by the volume approximation algorithms.
Optimization ($\mathrm{OPT}_P(c)$): Given vector $c \in \mathbb{R}^d$, find vector $y \in P$ maximizing $c^T x, x \in P$, or assert $P = \emptyset$.
Membership ($\mathrm{MEM}_P(y)$): Given a vector $y \in \mathbb{R}^d$ decide whether $y \in P$.
Separation ($\mathrm{SEP}_P(y)$): Given a vector $y \in \mathbb{R}^d$ call $MEM_P(y)$. If it answers negatively, find the normal to a hyperplane that separates y from P; i.e. $c \in \mathbb{R}^d \ : \ c^T y > \max\{c^T x \mid x \in P\}$.

For a H-polytope P, $\mathrm{OPT}_P(c)$ reduces to solve the linear program $\{\max c^T x \text{ s.t. } Ax \leq b\}$ while $\mathrm{MEM}_P(y)$ and $\mathrm{SEP}_P(y)$ reduce to check whether y satisfies the linear inequalities that define P in $O(dm)$.

For a V-polytope P we have the opposite situation. $\mathrm{OPT}_P(c)$ reduces to find the vertex v that maximizes $c^T v$ in $O(dn)$ while $\mathrm{MEM}_P(y)$ and $\mathrm{SEP}_P(y)$ reduce to test whether the set of linear inequalities $\{y = \sum_{i=0}^n \lambda_i v_i, \sum_{i=0}^n \lambda_i = 1, \lambda_i \geq 0 \text{ for all } i = 1, \ldots, n\}$ has a feasible solution which is polynomially equivalent to linear programming.

2.1 Random samplers

A common method to generate random points in a polytope P is by (geometric) random walks. Given a point in P, we run a geometric random walk by a sufficient number of steps to compute a random point in P. Geometric random walks proposed and analyzed in the literature include the grid walk, the ball walk, and the hit-and-run walk [31]. Here we focus on variations of the hit-and-run walk that generate a uniform distribution of points [32, 6].

Hit-and-run. The main procedure is $\mathrm{Walk}(x, P, s)$ takes as input a point $x \in P \subseteq \mathbb{R}^d$ and
(i) Run $\mathrm{Line}(x)$ which returns a line through x,
(ii) Move x to a random point uniformly distributed in $P \cap \ell$,
for s number of steps. We study and implement the following types of hit-and-run:

- Random Directions (RDHR): $\mathrm{Line}(x)$ return the line through x defined by a random vector uniformly distributed on the unit sphere centered at x. This walk generates a uniformly distributed point in $O^*(d^3)$ steps [27].

- Coordinate Directions (CDHR): Line(x) return the line through x defined by a random vector uniformly distributed on the set $\{e_1, \ldots, e_d\}$, where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, $i = 1, \ldots, d$. As far as the authors know, the time to reach the uniform distribution has not been analyzed yet.

2.2 Oracle implementation

For hit-and-run the procedure Walk(x, P, s) requires at every step the intersection of a line $\ell = \{p : p = \lambda v + x\}$ with the boundary of P. This is the *boundary oracle*, available if P is a H-polytope. Then we shall turn attention to the membership oracle.

Boundary oracle. Assuming an H-polytope, the straightforward method is to consider all m hyperplanes. Let us compute the intersection of $\ell = \{p : p = \lambda v + x\}$, with the i-th hyperplane:

$$\mathbf{p_i} := \mathbf{x} + \frac{b_i - \mathbf{a_i x}}{\mathbf{a_i v}} \mathbf{v}, \qquad i \in \{1, \ldots, m\},$$

then compute the two points at which ℓ intersects the boundary of P, namely

$$\mathbf{p}^+, \text{ s.t. } \mathbf{p}^+\mathbf{v} = \min_{1 \le i \le m} \{\mathbf{p_i v} \mid \mathbf{p_i v} \ge 0\}, \qquad \mathbf{p}^-, \text{ s.t. } \mathbf{p}^-\mathbf{v} = \max_{1 \le i \le m} \{\mathbf{p_i v} \mid \mathbf{p_i v} \le 0\}.$$

This is computed in $O(md)$ arithmetic operations. In practice, only the factor of $\mathbf{v}$ is computed so as to define $\mathbf{p}^+, \mathbf{p}^-$, i.e. the parameter specifying the corresponding points.

The parameter must be finally compared to that of the line point lying on the current sphere. Assuming the sphere is centered at the origin with radius R, its intersections with ℓ are $\mathbf{p} = \mathbf{x} + \lambda \mathbf{v}$ such that

$$\lambda^2 + 2\lambda \mathbf{x}\mathbf{v} + |\mathbf{x}|^2 - R^2 = 0. \tag{1}$$

Let $\mathbf{p}^\pm = \mathbf{x} + \lambda^\pm \mathbf{v}$, then if λ^+, λ^- give a negative sign when substituted to equation (1), then $\mathbf{p}^+, \mathbf{p}^-$ are the endpoints of the segment of ℓ lying in the intersection of P and the current ball. Otherwise, we have to compute one or two roots of equation (1) since the segment has one or two endpoints on the sphere.

However, in the CDHR walk after the computation of the first pair $\mathbf{p}^+, \mathbf{p}^-$, all other pairs can be computed in $O(m)$ arithmetic operations. The reason is that two sequential points produced by the walk differ only in one coordinate. Let ρ, ρ' be the walk coordinate of the previous and the current step respectively, then the current intersection point is:

$$\mathbf{p'_i} := \mathbf{x'} + \left(\frac{(b_i - \mathbf{a_i x}) + a_{i\rho}(x_{i\rho} - x'_{i\rho})}{a_{i\rho'}} \right) \mathbf{v}, \qquad i \in \{1, \ldots, m\},$$

Since $(b_i - \mathbf{a_i x})$ has been precomputed, only one division, one multiplication, and 3 additions are performed per hyperplane of P. In the latter expression for $\mathbf{p'_i}$, the parameter is the quantity in parenthesis.

Membership oracle. The most general approach, used by [28], is binary search on ℓ, where each step calls a membership oracle to test whether the point lies in P_i. In order to define the segment of the search, our code tries successively farther points on ℓ, by doubling their distance from x. It is possible to simply execute binary search on the segment between x and $\ell \cap B_i$. The binary search approximates point $\ell \cap P_i$ within distance τ along ℓ, after $O(i/d + \lg \tau)$ membership oracles, where i/d is the radius of B_i.

If the membership oracle is implemented in a straightforward manner, we have to check the query against a hyperplane. For points outside P_i it suffices to find one violated hyperplane;

for this, we sort the hyperplanes in random order at the beginning. For internal points, all hyperplanes must be checked for an exact answer.

To check the query against a hyperplane takes $O(d)$ operations. Essentially, we obtain the intersection of ℓ with the hyperplane and store it so that subsequent tests against this hyperplane take $O(1)$. Therefore the total complexity is $O(md+i/d+\lg\tau)$, which is worse than the boundary oracle. On the positive side, we may apply polytope approximation.

3 Volume approximation

Algorithms in this family are the current state-of-the-art, at least with respect to asymptotic complexity bounds. They take as a membership oracle for polytope $P \subseteq \mathbb{R}^d$ and execute two phases: *(isotropic) sandwiching* and *Multiphase Monte Carlo* (MMC) [31]. Sandwiching involves bounding $P \subseteq \mathbb{R}^d$ with ratio $\rho > 1$, i.e. computing an affine transformation A such that AP is in isotropic position, contains $B(1)$ and is contained in $B(\rho)$, where $B(\rho)$ is the ball of radius ρ centered at the origin. In this paper, as is not infrequent in similar studies [28], we do not address this phase. MMC constructs a sequence of bodies

$$P_i := P \cap B(2^{i/d}),\ i = 0, \ldots, \beta = \lceil d \lg \rho \rceil,$$

i.e., $P_0 = B(1)$ and $P_\beta = P$. Then, we estimate $\mathrm{vol}(P)$ by the telescopic product

$$\mathrm{vol}(P_0) \prod_{i=1}^{\beta} \frac{\mathrm{vol}(P_i)}{\mathrm{vol}(P_{i-1})}, \qquad \text{where } \mathrm{vol}(P_0) = \mathrm{vol}(B(1)) = \pi^{d/2}/\Gamma(1+d/2).$$

This reduces to estimating the ratios $\mathrm{vol}(P_i)/\mathrm{vol}(P_{i-1})$, which is achieved by generating N uniformly distributed points in P_i and by counting how many of them fall in P_{i-1}. The observation is that the volume ratio is constant, thus leading to a polynomial bound on the number of points, namely $N = 400\varepsilon^{-2} d \log d = O^*(d)$.

Proposition 1. [21] *Assuming $B(1) \subseteq P \subseteq B(\rho)$, the algorithm returns an estimation of vol(P), which lies between $(1-\varepsilon)$vol(P) and $(1+\varepsilon)$vol(P) with probability $\geq 3/4$, by*

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

oracle calls with probability $\geq 9/10$, where we have assumed ε is fixed. Sandwiching yields $\rho = \sqrt{d/\lg(1/\varepsilon)}$, implying $O^*(d^5)$ calls.

Interestingly, both sandwiching and MMC require $O^*(d^5)$ oracle calls. The bottleneck of MMC is point generation, each of which requires $O^*(d^3)$ oracles. It is widely believed, see e.g., [28], that this is quite loose, and this is confirmed by our experiments.

A more complicated approach based on simulated annealing reduces total complexity to $O^*(d^4)$ calls [30].

Computation of the sequence of balls. To compute the sequence of co-centric balls in practice we first compute the *Chebychev ball* $B(c,r)$ of P, i.e. the largest inscribed ball in P. It suffices to solve the linear program:

$$\begin{array}{ll} \text{maximize} & R \\ \text{subject to} & A_i x + R||A_i||_2 \leq b_i,\ i = 1, \ldots, m \\ & R \geq 0 \end{array}$$

Algorithm 1: VolEsti $(P, N, steps)$

Input : P: H-polytope, N: the number of generated random points, s: the number of random walk steps
Output: An approximation of vol(P)

compute the Chebychev ball $B(c, r)$;
generate a random point p in $B(c, r)$;
for $i = 1$ *to* N **do**
  $p \leftarrow \text{Walk}(p, P, steps)$;
  add p in L;
set ρ the largest distance from c to any point in L;
construct the sequence of balls $B(c, 2^{i/d})$ for $i = \lfloor d/\lg r \rfloor, \ldots, \lceil d/\lg \rho \rceil$;
$P_i \leftarrow P \cap B(c, 2^{i/d})$ for $i = \lfloor d/\lg r \rfloor, \ldots, \lceil d/\lg \rho \rceil$;
$vol \leftarrow 2\pi^{d/2}\rho^d/d\Gamma(d/2)$;
for $i = \lfloor d/\lg r \rfloor$ *down to* $\lceil d/\lg \rho \rceil$ **do**
  $count\_prev \leftarrow size(L)$;
  remove from L the points not in P_i;
  $count \leftarrow size(L)$;
  Set p to be an arbitrary point from L;
  for $j = 1$ *to* $N - count\_prev$ **do**
    $p \leftarrow \text{Walk}(p, P_i, steps)$;
    if $p \in B(i-1)$ **then**
      $count \leftarrow count + 1$;
      add p in L;
  $vol \leftarrow vol * (N/count)$;
return vol;

where the objective value and the value of x is the radius r and the center c of the Chebychev ball.

Then, we can compute a uniformly distributed random point, and use it as a start to perform a random walk in P and compute N random points. We compute the largest distance among each of the N points and the Chebychev center and set ρ to this value. Now, the sequence of balls is

$$B(c, 2^{i/d}), \quad i = \lfloor d/\lg r \rfloor, \ldots, \lceil d/\lg \rho \rceil.$$

Random points generation. Although we can easily generate (almost) uniform random points in balls (or other standard shapes like simplices and cubes), the situation is completely inverted with general convex bodies such as $P_i = P \cap B(2^{i/d})$. The state-of-the-art method is to use the random walk procedures in Sect. 2.1. For mixing time, it is important that (in most cases) we start a random walk in P_i from a point which is uniformly distributed P_i.

Unlike typical approaches, which generate points in P_i for $i = 0, 1, \ldots, \beta$, here we shall proceed inversely. First, let us describe initialization. We generate a uniformly distributed random point in the smallest ball, which is easy since it is completely inside P. Then we use p to start a random walk in P_1, P_2, P_3 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 vol(P_β)/vol$(P_{\beta-1})$. Next we keep those of the random points that lie in $P_{\beta-1}$, and

use them to start random walks so as to obtain a total of N (almost) uniformly distributed points in $P_{\beta-1}$. We repeat until we compute the last ratio $\text{vol}(P_1)/\text{vol}(P_0)$.

The implementation is based on a data structure L that stores the random points. In general, in step $i > 1$ we want to compute $\text{vol}(P_{\beta-i})/\text{vol}(P_{\beta-i-1})$ and L contains N random points in $P_{\beta-i+1}$ from the previous step. The computation in this step consists in removing from L the points not in $P_{\beta-i}$, then generating $N - size(L)$ more random points in $P_{\beta-i}$ and, finally, counting how many of them fall into $P_{\beta-i-1}$. Observe that testing whether such a point lies in some P_i reduces to testing whether $p \in B(2^{i/d})$ because we know $p \in P$. An important feature of our method is that it creates partial generations of random points for every new convex body P_i, as opposed to having always to generate N points. This method had been used in [7] in the context of convex optimization.

4 Experiments

We implement and experimentally test the above algorithms and methods. The code has been developed in C++ and is publicly available from `http://sourceforge.net/projects/randgeom`. It uses the the d-dimensional kernel of CGAL library [13] to represent geometric objects such as points, vectors and the CGAL linear programming solver [19]. We also use CGAL to generate random points in balls. We use floating point arithmetic, namely the `double` data type of C++, except from the linear programming solver which uses the *GNU Multiple Precision arithmetic library*, the current standard for multiple-precision arithmetic. The pseudo-random number generators relies on Boost [10]. All timings are on an Intel Core i5-2400 3.1GHz, 6MB L2 cache, 8GB RAM, 64-bit Debian GNU/Linux.

To compute the volume exactly we use `VINCI` [12] a software package that implements many state-of-the art algorithms for exact volume computation. In particular, since we assume input polytopes in H-representation we use `VINCI`'s methods that accept this representation as input. That is, Lasserre's method and Lawrence's formula in the general case. The latter yield runtime errors and thus have not tested in the experiments.

Polytopes dataset. The following polytopes from `VINCI` webpage are tested within our experiments.

- cube: hypercubes with -1 and 1 as vertex coordinates.
- cross: cross polytopes, the duals of the cubes above, in dimension 2 to 14.
- rh: polytopes constructed by randomly choosing hyperplanes tangent to the sphere; after unpacking, files rh_d_m will be created, where d stands for the dimension and m for the number of hyperplanes.
- rv: dually to the previous category these polytopes have vertices randomly distributed on the sphere.
- cc: cc_8_7 to cc_8_11, the product of two cyclic polyhedra with seven to eleven vertices, each in dimension four. The final dimension is therefore eight.
- ccp: complete cut polytopes on five to seven vertices. The polytopes have been scaled by 2 and then translated by 1 to contain the origin in their interior. So their vertices have coordinates +1 and -1 now.
- metric: after expansion, contains facets of the fourth (Fm_4) to sixth (Fm_6) metric polytope.

Table 1 shows experimental results.

polytope	d	m	vol	N	μ	[min, max]	std-dev	$\frac{vol-\mu}{vol}$	VolEsti (sec)	VINCI (sec)
cube_10	10	20	1.0240E+03	9210	1.0271E+03	[950.3755,1107.804]	3.16E+001	0.0030	0.42	0.00
cube_11	11	22	2.0480E+03	10550	2.0447E+03	[1872.304,2194.309]	6.40E+001	0.0016	0.5089	0.01
cube_12	12	24	4.0960E+03	11927	4.1021E+03	[3819.677,4371.094]	1.22E+002	0.0015	0.6601	0.03
cube_13	13	26	8.1920E+03	13337	8.2342E+03	[7341.784,8788.589]	2.71E+002	0.0051	0.8574	0.07
cube_14	14	28	1.6384E+04	14778	1.6275E+04	[15076.71,18030.89]	5.03E+002	0.0067	1.0777	0.18
cube_20	20	40	1.0486E+06	23965	1.0456E+06	[974257.5,1116488]	3.15E+004	0.0028	4.62	swap
cube_30	30	60	1.0737E+09	40814	1.0779E+09	[9.911e+08,1.159e+09]	3.89E+007	0.0039	17.96	swap
cube_40	40	80	1.0995E+12	59022	1.1038E+12	[1.014e+12,1.235e+12]	4.46E+010	0.0039	50.72	swap
cube_50	50	100	1.1259E+15	78240	1.1251E+15	[1.003e+15,1.253e+15]	4.39E+013	0.0007	117.51	swap
cube_60	60	120	1.1529E+18	98264	1.1588E+18	[1.063e+18,1.269e+18]	4.00E+016	0.0051	222.10	swap
cube_70	70	140	1.1806E+21	118957	1.1790E+21	[1.021e+21,1.323e+21]	5.42E+019	0.0013	358.93	swap
cube_80	80	160	1.2089E+24	140224	1.2079E+24	[1.133e+24,1.297e+24]	4.42E+022	0.0009	582.19	swap
cube_90	90	180	1.2379E+27	161993	1.2403E+27	[1.094e+27,1.441e+27]	5.18E+025	0.0019	875.69	swap
cube_100	100	200	1.2677E+30	184206	1.2779E+30	[1.165e+30,1.402e+30]	4.82E+028	0.0081	1285.08	swap
cross_10	10	1024	2.8219E-04	9210	2.8211E-04	[2.693e+04,2.944e+04]	5.15E-006	0.0003	1.58	error*
cross_11	11	2048	5.1307E-05	10550	5.1258E-05	[4.8884e-05,5.437e-05]	1.15E-006	0.0010	5.37	error*
cross_12	12	4096	8.5511E-06	11927	8.5573E-06	[8.130e-06,9.020e-06]	1.69E-007	0.0007	12.53	error*
cross_13	13	8192	1.3156E-06	13337	1.3197E-06	[1.251e-06,1.434e-06]	2.72E-008	0.0032	33.71	error*
cross_14	14	16384	1.8794E-07	14778	1.8806E-07	[1.785e-07,1.992e-07]	3.72E-009	0.0007	67.07	error*
rh_8_20	8	20	3.7576E+04	26616	3.5252E+04	[26266.94,49556.69]	4.19E+003	0.0619	2.28	0.14
rh_8_25	8	25	7.8599E+02	26616	7.8780E+02	[740.6302,830.1389]	1.67E+001	0.0023	1.36	1.14
rh_8_30	8	30	2.4739E+02	26616	2.4695E+02	[239.179,256.1362]	3.49E+000	0.0018	1.12	5.56
rh_10_20	10	20	1.3883E+04	36841	1.3900E+04	[13185.45,14757.85]	3.29E+002	0.0013	2.90	0.43
rh_10_25	10	25	5.7295E+03	36841	5.7177E+03	[5458.034,6053.999]	1.26E+002	0.0021	2.75	6.88
rh_10_30	10	30	2.0156E+03	36841	2.0164E+03	[1921.955,2092.74]	3.64E+001	0.0004	2.53	swap
rv_8_10	8	24	1.4096E+19	6654	1.4014E+19	[1.151e+19,1.691e+19]	9.71E+017	0.0059	0.58	0.01
rv_8_11	8	54	3.0477E+18	6654	1.5952E+18	[6.038e+17,3.467e+18]	5.34E+017	0.4766	1.48	0.54
rv_8_11	8	54	3.0477E+18	665421	3.1339E+18	[3.134e+18,3.134e+18]	0.00E+000	0.0283	157.46	0.54
rv_8_12	8	94	4.3858E+19	6654	4.3836E+19	[3.846e+19,5.121e+19]	2.50E+018	0.0005	0.90	261.37
rv_8_13	8	131	1.3341E+20	6654	1.3167E+20	[1.157e+20,1.488e+20]	7.75E+018	0.0131	1.31	swap
rv_8_14	8	218	2.1566E+20	6654	2.1457E+20	[1.858e+20,2.444e+20]	1.27E+019	0.0050	1.81	swap
rv_8_20	8	1191	2.6918E+21	6654	2.6811E+21	[2.369e+21,2.904e+21]	1.01E+020	0.0040	4.39	swap
rv_8_30	8	4482	7.3502E+21	6654	7.3423E+21	[6.723e+21,7.913e+21]	2.26E+020	0.0011	14.99	swap
rv_10_12	10	35	2.1360E+22	9210	1.9275E+22	[1.246e+22,2.787e+22]	3.13E+021	0.0976	1.53	0.01
rv_10_13	10	89	1.6329E+23	9210	1.6052E+23	[1.334e+23,1.894e+23]	1.19E+022	0.0170	2.00	59.50
rv_10_14	10	177	2.9314E+23	9210	2.9210E+23	[2.513e+23,3.335e+23]	1.89E+022	0.0035	2.91	swap
cc_8_5	8	10	1.7361E-03	6654	1.4690E-03	[0.000882,0.00261]	2.97E-004	0.1538	0.69	0.01
cc_8_6	8	18	4.4444E-01	6654	3.7595E-01	[0.2426,0.7265]	9.48E-002	0.1541	0.71	0.01
cc_8_7	8	28	2.7563E+01	6654	2.6080E+01	[19.14487,37.12707]	3.31E+000	0.0538	0.60	0.01
cc_8_8	8	40	7.8400E+02	6654	7.0977E+02	[464.7698,1120.471]	1.25E+002	0.0947	0.64	0.01
cc_8_9	8	54	1.3340E+04	6654	1.2663E+04	[9472.759,16521.93]	1.43E+003	0.0508	0.67	0.02
cc_8_10	8	70	1.5682E+05	6654	1.4499E+05	[111151.2,186540.5]	1.57E+004	0.0754	0.73	0.05
cc_8_11	8	88	1.3918E+06	6654	1.4009E+06	[998002.9,1994552]	1.83E+005	0.0066	0.85	0.08
Fm_4	6	7	8.6400E+01	4300	8.5927E+01	[71.38417,112.0267]	8.38E+000	0.0055	0.1901	0.01
Fm_5	10	25	7.1095E+03	9210	7.1157E+03	[6350.727,8103.786]	3.01E+002	0.0009	0.6926	0.02
Fm_6	15	59	2.8611E+05	16248	2.8502E+05	[241851.7,321864.3]	1.55E+004	0.0038	3.2361	swap
ccp_5	10	56	2.3117E+00	9210	2.3264E+00	[2.159411,2.527959]	7.43E-002	0.0064	0.49	38.00
ccp_6	15	368	1.3458E+00	16248	1.3460E+00	[1.264474,1.451714]	3.81E-002	0.0002	6.14	error*

Table 1: Volume computation experimental results; walk length is 10, $\epsilon = 1$, runs 100. Swap: the program run out of memory and start swapping; error*: VINCI complains that it cannot use 'rlass' with more than 254 hyperplanes

Acknowledgment. Section 1 is based on a text by Dr. Grigoris Karagiorgos (greg@di.uoa.gr).

References

[1] E. Airoldi and B. Haas. Polytope samplers for inference in ill-posed inverse problems. *Journal of Machine Learning Research - Proceedings Track*, 15:110–118, 2011.

[2] E. Almaas, B. Kovács, T. Vicsek, Z.N. Oltvai, and A-L. Barabási. Global organization of metabolic fluxes in the bacterium Escherichia coli. *Nature*, 427(6977):839–843, February 2004.

[3] A. Applegate and R. Kannan. *Sampling and integration of near log-concave functions.* In Proc. 23th ACM STOC, 156-163, 1990.

[4] S. Arya, G. Dias da Fonseca, and D.M. Mount. Optimal area-sensitive bounds for polytope approximation. In *ACM Symp. on Comp. Geometry*, pages 363–372, 2012.

[5] L. Bárány and Z. Füredi. Computing the volume is difficult. Proc. 18th ACM STOC 442–447, 1986.

[6] H.C.P. Berbee, C.G.E. Boender, A.H.G. Rinnooy Ran, C.L. Scheffer, R.L. Smith, and J. Telgen. Hit-and-run algorithms for the identification of nonredundant linear inequalities. *Math. Programming*, 37(2):184–207, 1987.

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

[8] Y. Bishop, S.E. Fienberg, and P. Holland. *Discrete multivariate analysis: theory and practice*. The MIT press, 1975.

[9] B. Bollobás. Volume Estimates and Rapid Mixing. MSRI Publications, Vol. 31, 1997.

[10] Boost: C++ libraries. http://www.boost.org.

[11] E. M. Bronshteyn and L. D. Ivanov. The approximation of convex sets by polyhedra. *Siberian Math. J.*, 16:852–853, 1976.

[12] B. Büeler and A. Enge. VINCI. http://www.math.u-bordeaux1.fr/∼aenge/index.php?category=software&page=vinci.

[13] CGAL: Computational geometry algorithms library. http://www.cgal.org.

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

[15] M. Dyer and A. Frieze. On the complexity of computing the volume of a polytope, *SIAM J. Comp.*, 17:967–974, 1988.

[16] M. Dyer and A. Frieze. *Computing the volume of convex bodies: a case where randomness provably helps*. In Probabilistic Combinatorics and Its Applications (ed. B. Bollobas), Proc. Symposia in Applied Mathematics, Vol. 44, 123–170, 1992.

[17] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approximation Theory*, 10:227–236, 1974.

[18] G. Elekes. A Geometric inequality and the complexity of computing volume. *Discrete and Comput. Geometry*, 1:289–292, 1986.

[19] Kaspar Fischer, Bernd Gärtner, Sven Schönherr, and Frans Wessendorp. Linear and quadratic programming solver. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.3 edition, 2013.

[20] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 2nd corrected edition, 1993.

[21] 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.

[22] L.G. Khachiyan. On the complexity of computing the volume of a polytope. *Uspekhi Akad. Nauk SSSR, Engineering Cybernetics*, 3:216–217, 1988.

[23] L.G. Khachiyan. The problem of computing the volume of polytopes is NP-hard. *Uspekhi Akad. Nauk*, 44:199–200, 1989.

[24] S. Liu, J. Zhang, and B. Zhu. Volume computation using a direct Monte Carlo method. In G. Lin, editor, *Computing and Combinatorics*, volume 4598 of *LNCS*, pages 198–209. Springer, 2007.

[25] L. Lovàsz. How to compute the volume? *Jber. d. Dt. Math.-Verein, Jubilaumstagung*, B.G. Teubner, Stuttgart, pages 138–151, 1992.

[26] L. Lovàsz and M. Simonovits. The mixing rate of Markov chains, an isoperimetric inequality, and Computing the volume. In Proc. 31st Symposium on Foundations of Computer Science, pages 346–354, 1990.

[27] L. Lovász. Hit-and-run mixes fast. *Math. Prog.*, 86:443–461, 1998.

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

[29] L. Lovàsz and M. Simonovits. Random walks in a convex body and an improved volume algorithm. Random Structures and Algorithms, 4:4, 359–412, 1993.

[30] 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.

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

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

[33] Y. Vardi. Network tomography: estimating source-destination traffic intensities from link data. *J. American Stat. Assoc.*, 91:365–377, 1996.

[34] S. Vempala, *Geometric Random Walks: A survey*, Combinatorial and Computational Geometry, MSRI Publications Vol. 52, 573-612, 2005.

[35] S.J. Wiback, I. Famili, H.J. Greenberg, and B. Palsson. Monte Carlo sampling can be used to determine the size and shape of the steady-state flux space. *J. Theoretical Biology*, 228(4):437–447, June 2004.