A Fast and Practical Method to Estimate Volumes of Convex Polytopes

Cunjing Ge[1,2](✉) and Feifei Ma[2]

[1] Institute of Software, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{gecj,maff}@ios.ac.cn

Abstract. The volume is an important attribute of a convex body. In general, it is quite difficult to calculate the exact volume. But in many cases, it suffices to have an approximate value. Volume estimation methods for convex bodies have been extensively studied in theory, however, there is still a lack of practical implementations of such methods. In this paper, we present an efficient method which is based on the Multiphase Monte-Carlo algorithm to estimate volumes of convex polytopes. It uses the coordinate directions hit-and-run method, and employs a technique of reutilizing sample points. We also introduce a new result checking method for performance evaluation. The experiments show that our method can efficiently handle instances with dozens of dimensions with high accuracy.

1 Introduction

Volume computation is a classical problem in mathematics, arising in many applications such as economics, computational complexity analysis, linear systems modeling, and statistics. It is also extremely difficult to solve. Dyer et al. [1] and Khachiyan [2,3] proved respectively that exact volume computation is #P-hard, even for explicitly described polytopes. Büeler et al. [4] listed five volume computation algorithms for convex polytopes. However, only the instances around 10 dimensions can be solved in reasonable time with existing volume computation algorithms, which is quite insufficient in many circumstances. Therefore we turn attention to volume estimation methods.

There are many results about volume estimation algorithms of convex bodies since the end of 1980s. A breakthrough was made by Dyer, Frieze and Kannan [5]. They designed a polynomial time randomized approximation algorithm (Multiphase Monte-Carlo Algorithm), which was then adopted as the framework of volume estimation algorithms by successive works. At first, the theoretical complexity of this algorithm is $O^*(n^{23})$[1], but it was soon reduced to $O^*(n^4)$ by Lovász, Simonovits et al. [6–9]. Despite the polynomial time results and reduced complexity, there is still a lack of practical implementation. In fact, there are

[1] "soft-O" notation O^* indicates that we suppress factors of $\log n$ as well as factors depending on other parameters like the error bound.

J. Wang and C. Yap (Eds.): FAW 2015, LNCS 9130, pp. 52–65, 2015.
DOI: 10.1007/978-3-319-19647-3_6

some difficulties in applying the above volume estimation algorithms. First, in theoretical research of randomized volume algorithms, oracles are usually used to describe the convex bodies and the above time complexity results are measured in terms of oracle queries. However, oracles are too complex and oracle queries are time-consuming. Second, there exists a very large hidden constant coefficient in the theoretical complexity [8], which makes the algorithms almost infeasible even in low dimensions. The reason leading to this problem is that the above research works mostly focus on arbitrary dimension and theoretical complexity. To guarantee that Markov Chains mix in high-dimensional circumstance, it is necessary to walk a large constant number of steps before determining the next point.

In this paper, we focus on practical and applicable method. We only consider specific and simple objects, i.e., convex polytopes. On the other hand, the size of problem instances is usually limited in practical circumstances. With such limited scale, we find that it is unnecessary to sample as many points as the algorithm in [8] indicates. We implement a volume estimation algorithm which is based on the Multiphase Monte-Carlo method. The algorithm is augmented with a new technique to reutilize sample points, so that the number of sample points can be significantly reduced. We compare two hit-and-run methods: the hypersphere directions method and the coordinate directions method, and find that the latter method which is employed in our approximation algorithm not only runs faster, but is also more accurate. Besides, in order to better evaluate the performance of our tool, we also introduce a new result checking method. Experiments show that our tool can efficiently handle instances with dozens of dimensions. To the best of our knowledge, it is the first practical volume estimation tool for convex polytopes.

We now outline the remainder of the paper: In Sect. 2, we propose our method in detail. In Sect. 3, we show experimental results and compare our method with the exact volume computation tool `VINCI` [10]. Finally we conclude this paper in Sect. 4.

2 The Volume Estimation Algorithm

A convex polytope may be defined as the intersection of a finite number of half-spaces, or as the convex hull of a finite set of points. Accordingly there are two descriptions for a convex polytope: half-space representation (H-representation) and vertex representation (V-representation). In this paper, we adopt the H-representation.

An n-dimensional convex polytope P is represented as $P = \{Ax \leq b\}$, where A is an $(m \times n)$ matrix. a_{ij} represents the element at the i-th row and the j-th column of A, and a_i represents the i-th column vector of A. For simplicity, we also assume that P is full-dimensional and not empty. We use $vol(K)$ to represent the volume of a convex body K, and $B(x, R)$ to represent the ball with radius R and center x.

We define ellipsoid $E = E(A, a) = \{x \in \mathbb{R}^n | (x - a)^T A^{-1} (x - a) \leq 1\}$, where A is a symmetric positive definite matrix.

Like the original multiphase Monte-Carlo algorithm, our algorithm consists of three parts: rounding, subdivision and sampling.

2.1 Rounding

The rounding procedure is to find an affine transformation T on polytope Q such that $B(0,1) \subseteq T(Q) \subseteq B(0,r)$ and a constant $\gamma = \frac{vol(Q)}{vol(T(Q))}$. If $r > n$, T can be found by the Shallow-β-Cut Ellipsoid Method [11], where $\beta = \frac{1}{r}$. It is an iterative method that generates a series of ellipsoids $\{E_i(T_i, o_i)\}$ s.t. $Q \subseteq E_i$, until we find an E_k such that $E_k(\beta^2 T_k, o_k) \subseteq Q$. Then we transform the ellipsoid E_k into $B(0,r)$. Note that this method is numerically unstable on even small-sized problems, such as polytopes in 20-dimensions. Therefore, we adopt a modification of Ellipsoid Method which described in [12].

This procedure could take much time when r is close to n, e.g. $r = n+1$. There is a tradeoff between rounding and sampling, since the smaller r is, the more iterations during rounding and the fewer points have to be generated during sampling. We set $r = 2n$ in our implementation. Rounding can handle very "thin" polytopes which cannot be subdivided or sampled directly. We use P to represent the new polytope $T(Q)$ in the sequel.

2.2 Subdivision

To avoid curse of dimensionality(the possibility of sampling inside a certain space in target object decreases very fast while dimension increases), we subdivide P into a sequence of bodies so that the ratio of consecutive bodies is at most a constant, e.g. 2. Place $l = \lceil n \log_2 r \rceil$ concentric balls $\{B_i\}$ between $B(0,1)$ and $B(0,r)$, where

$$B_i = B(0, r_i) = B(0, 2^{i/n}),\ i = 0, \ldots, l.$$

Set $K_i = B_i \cap P$, then $K_0 = B(0,1)$, $K_l = P$ and

$$vol(P) = vol(B(0,1)) \prod_{i=0}^{l-1} \frac{vol(K_{i+1})}{vol(K_i)} = vol(B(0,1)) \prod_{i=0}^{l-1} \alpha_i. \tag{1}$$

So we only have to estimate the ratio $\alpha_i = vol(K_{i+1})/vol(K_i)$, $i = 0, \ldots, l-1$. Since $K_i = B_i \cap P \subseteq B_{i+1} \cap P = K_{i+1}$, we get $\alpha_i \geq 1$. On the other hand, $\{K_i\}$ are convex bodies, then

$$K_{i+1} \subseteq \frac{r_{i+1}}{r_i} K_i = 2^{1/n} K_i,$$

we have

$$\alpha_i = \frac{vol(K_{i+1})}{vol(K_i)} \leq 2.$$

Specially, $K_{i+1} = 2^{1/n} K_i$ if and only if $K_{i+1} = B_{i+1}$ i.e. $B_{i+1} \subseteq P$. That is, $1 \leq \alpha_i \leq 2$ and $\alpha_i = 2 \Leftrightarrow B_{i+1} \subseteq P$.

2.3 Hit-and-Run

To approximate α_i, we generate *step_size* random points in K_{i+1} and count the number of points c_i in K_i. Then $\alpha_i \approx \mathit{step\_size}/c_i$. It is easy to generate uniform distributions on cubes or ellipsoids but not on $\{K_i\}$. So we use a random walk method for sampling. Hit-and-run method is a random walk which has been proposed and studied for a long time [13–15]. The hypersphere directions method and the coordinate directions method are two hit-and-run methods. In the hypersphere directions method, the random direction is generated from a uniform distribution on a hypersphere; in the coordinate directions method, it is chosen with equal probability from the coordinate direction vectors and their negations. Berbee et al. [14] proved the following theorems.

Theorem 1. *The hypersphere directions algorithm generates a sequence of interior points whose limiting distribution is uniform.*

Theorem 2. *The coordinate directions algorithm generates a sequence of interior points whose limiting distribution is uniform.*

Coordinate directions and their negations are special cases of directions generated on a hypersphere, hence the former theoretical research about volume approximation algorithm with hit-and-run methods mainly focus on the hypersphere directions method [8]. In this paper, we apply the coordinate directions method to our volume approximation algorithm. It starts from a point x in K_{k+1}, and generates the next point x' in K_{k+1} by two steps:

Step 1. Select a line L through x uniformly over n coordinate directions, $e_1, \dots e_n$.
Step 2. Choose a point x' uniformly on the segment in K_{k+1} of line L.

More specifically, we randomly select the dth component x_d of point x and get x_d's bound $[u, v]$ that satisfies

$$x|_{x_d=t} \in K_{k+1},\ \forall t \in [u, v] \tag{2}$$

$$x|_{x_d=u}, x|_{x_d=v} \in \partial K_{k+1} \tag{3}$$

("∂" denotes the boundary of a set). Then we choose $x'_d \in [u, v]$ with uniform distribution and generate the next point $x' = x|_{x_d=x'_d} \in K_{k+1}$.

Our hit-and-run algorithm is described in Algorithm 1. $R_i = 2^{i/n}$ is the radius of B_i. Note that $K_{k+1} = B_{k+1} \cap P$, so $x' \in B_{k+1}$ and $x' \in P$. We observe that

$$x' \in B_{k+1} \ \Leftrightarrow\ |x'| \le R_{k+1} \ \Leftrightarrow\ x'^2_d \le R^2_{k+1} - \sum_{i \ne d} x_i^2$$

$$x' \in P \ \Leftrightarrow\ a_i x' \le b_i \ \Leftrightarrow\ a_{id} x'_d \le b_i - \sum_{j \ne d} a_{ij} x_j = \mu_i,\ \forall i$$

Algorithm 1. Hit-And-Run Sampling Algorithm

```
1: function WALK($x$, $k$)
2:     $d \leftarrow random(n)$
3:     $c \leftarrow |x|^2 - x_d^2$
4:     $r \leftarrow \sqrt{R_{k+1}^2 - c}$
5:     $max \leftarrow r - x_d$
6:     $min \leftarrow -r - x_d$
7:     for $i \leftarrow 1, m$ do
8:         $bound_i \leftarrow (b_i - a_i x)/a_{id}$
9:         if $a_{id} > 0$ and $bound_i < max$ then
10:            $max \leftarrow bound_i$
11:        else if $a_{id} < 0$ and $bound_i > min$ then
12:            $min \leftarrow bound_i$
13:        end if
14:    end for
15:    $x_d \leftarrow x_d + random(min,\ max)$
16:    return $x$
17: end function
```

Let

$$u = max\{-\sqrt{R_{k+1}^2 - \sum_{i \neq d} x_i^2},\ \frac{\mu_i}{a_{id}}\}\ \forall i\ s.t.\ a_{id} < 0$$

$$v = min\{\sqrt{R_{k+1}^2 - \sum_{i \neq d} x_i^2},\ \frac{\mu_i}{a_{id}}\}\ \forall i\ s.t.\ a_{id} > 0$$

then interval $[u, v]$ is the range of x'_d that satisfies Formulas (2) and (3), and $u = x_d + min$, $v = x_d + max$ in Algorithm 1.

Usually, $Walk$ function is called millions of times, so it is important to improve its efficiency, such as use iterators in **for** loop and calculation of $|x|$. At the same time, we move the division operation (line 8), which is very slow for double variables, out of $Walk$ function because $(b_i - a_i x)/a_{id} = \frac{b_i}{a_{id}} - \frac{a_i}{a_{id}} x$, i.e., divisions only occur between constants.

2.4 Reutilization of Sample Points

In the original description of the Multiphase Monte Carlo method, it is indicated that the ratios α_i are estimated in natural order, from the first ratio α_0 to the last one α_{l-1}. The method starts sampling from the origin. At the kth phase, it generates a certain number of random independent points in K_{k+1} and counts the number of points c_k in K_k to estimate α_k. However, our algorithm performs in the opposite way: Sample points are generated from the outermost convex body K_l to the innermost convex body K_0, and ratios are estimated accordingly in reverse order.

The advantage of approximation in reverse order is that it is possible to fully exploit the sample points generated in previous phases. Suppose we have already generated a set of points $\mathcal{S}$ by random walk with almost uniform distribution in K_{k+1}, and some of them also hit the convex body K_k, denoted by $\mathcal{S}'$. The ratio α_k is thus estimated with $\frac{|\mathcal{S}'|}{|\mathcal{S}|}$. But these sample points can reveal more information than just the ratio α_k. Since K_k is a sub-region of K_{k+1}, the points in $\mathcal{S}'$ are also almost uniformly distributed in K_k. Therefore, $\mathcal{S}'$ can serve as part of the sample points in K_k. Furthermore, for any K_i $(0 \leq i \leq k)$ inside K_{k+1}, the points in K_{k+1} that hit K_i can serve as sample points to approximate α_i as well.

Based on this insight, our algorithm samples from outside to inside. Suppose to estimate each ratio within a given relative error, we need as many as *step_size* points. At the kth phase which approximates ratio α_{l-k}, the algorithm first calculates the number *count* of the former points that are also in α_{l-k+1}, then generates the rest $(\mathit{step\_size} - \mathit{count})$ points by random walk.

Unlike sampling in natural order, choosing the starter for each phase in reverse sampling is a bit complex. The whole sampling process in reverse order also starts from the origin point. At each end of the k-th phase, we select a point x in K_{k+1} and employ $x' = 2^{-\frac{1}{n}}x$ as the starting point of the next phase since $2^{-\frac{1}{n}}x \in K_k$.

It's easy to find out that the expected number of reduced sample points with our algorithm is

$$\sum_{i=1}^{l-1}(\mathit{step\_size} \times \frac{1}{\alpha_i}). \tag{4}$$

Since $\alpha_i \leq 2$, we only have to generate less than half sample points with this technique. Actually, results of expriments show that we can save over 70 % time consumption on many polytopes.

2.5 Framework of the Algorithm

Now we present the framework of our volume estimation method. Algorithm 2 is the Multiphase Monte-Carlo algorithm with the technique of reutilizing sample points.

The function *Preprocess* represents the rounding procedure and it returns the ratio of γ. In Algorithm 2, the formula $\lceil \frac{n}{2} \log_2 |x| \rceil$ returns index i that $x \in K_i \backslash K_{i-1}$. We use t_i to record the number of sample points that hit $K_i \backslash K_{i-1}$. Furthermore, the sum *count* of $t_0, \ldots, t_{k+1}$ is the number of reusable sample points that are generated inside K_{k+1}. Then we only have to generate the rest $(\mathit{step\_size} - \mathit{count})$ points inside K_{k+1} in the k-th phase. Then we use $2^{-\frac{1}{n}}x$ as the starting point of the next phase. Finally, according to Eq. (1) and $\gamma = \frac{vol(Q)}{vol(P)}$, we achieve the estimation of $vol(Q)$.

Algorithm 2. The Framework of Volume Estimation Algorithm

```
1: function EstimateVol
2:     $\gamma \leftarrow Preprocess(\,)$
3:     $x \leftarrow O$
4:     $l \leftarrow \lceil n \log_2 r \rceil$
5:     for $k \leftarrow l-1,\ 0$ do
6:         for $i \leftarrow count,\ step\_size$ do
7:             $x \leftarrow Walk(x, k)$
8:             if $x \in B_0$ then
9:                 $t_0 \leftarrow t_0 + 1$
10:            else if $x \in B_k$ then
11:                $m \leftarrow \lceil \frac{n}{2} \log_2 |x| \rceil$
12:                $t_m \leftarrow t_m + 1$
13:            end if
14:        end for
15:        $count \leftarrow \sum_{i=0}^{k} t_i$
16:        $\alpha_k \leftarrow step\_size/count$
17:        $x \leftarrow 2^{-\frac{1}{n}} x$
18:    end for
19:    return $\gamma \cdot unit\_ball(n) \cdot \prod_{i=0}^{l-1} \alpha_i$
20: end function
```

3 Experimental Results

We implement the algorithm in C++ and the tool is named `PolyVest` (Polytope Volume Estimation). In all experiments, *step_size* is set to $1600\,l$ for the reason discussed in Appendix A and parameter r is set to $2n$. The experiments are performed on a workstation with 3.40 GHz Intel Core i7-2600 CPU and 8 GB memory. Both `PolyVest` and `VINCI` use a single core.

3.1 The Performance of `PolyVest`

Table 1 shows the results of comparison between `PolyVest` and `VINCI`. `VINCI` is a well-known package which implements the state of the art algorithms for exact volume computation of convex polytopes. It can accept either H-representation or V-representation as input. The test cases include: (1) "cube_n": Hypercubes with side length 2, i.e. the volume of "cube_n" is 2^n. (2) "cube_n(S)": Apply 10 times random shear mappings on "cube_n". The random shear mapping can be represented as PQP, with $Q = \begin{pmatrix} I & M \\ 0 & I \end{pmatrix}$, where the elements of matrix M are randomly chosen and P is the products of permutation matrices $\{P_i\}$ that put rows and columns of Q in random orders. This mapping preserves the volume. (3) "rh_n_m": An n-dimentional polytope constructed by randomly choosing m hyperplanes tangent to sphere. (4) "rh_n_m(S)": Apply 10 times random shear mappings on "rh_n_m". (5) "cuboid_n(S)": Scaling "cube_n" by 100 in one direction, and then apply random shear mapping on it once. We use this instance to approximate a "thin stick" which not parallel to any axis. (6) "ran_n_m": An n-dimentional polytope constructed by randomly choosing integer coefficient from -1000 to 1000 of matrix A.

Table 1. Comparison between `PolyVest` and `VINCI`

			`PolyVest`		`VINCI`			
Instance	n	m	Result	Time(s)	Result	T_{rlass}(s)	T_{hot}(s)	T_{lawnd}(s)
cube_10	10	20	1015.33	0.380	1024	0.004	0.044	0.008
cube_15	15	30	33560.1	1.752	32768	0.300	212.8	0.156
cube_20	20	40	1.08805e+6	4.484	1.04858e+6	—	—	8.085
cube_30	30	60	1.0902e+9	23.197	—	—	—	—
cube_40	40	80	1.02491e+12	72.933	—	—	—	—
cube_10(S)	10	20	1027.1	0.184	1023.86	0.008	0.124	0.024
cube_15(S)	14	28	30898.2	0.784	32766.4	0.428	369.6	0.884
rh_8_25	8	25	793.26	0.132	785.989	0.864	0.160	0.016
rh_10_20	10	20	13710.0	0.240	13882.7	0.284	0.340	0.012
rh_10_25	10	25	5934.99	0.260	5729.52	5.100	1.932	0.072
rh_10_30	10	30	2063.55	0.280	2015.58	660.4[a]	5.772	0.144
rh_8_25(S)	8	25	782.58	0.136	785.984	1.268	0.156	0.032
rh_10_20(S)	10	20	13773.2	0.232	13883.8	0.832	0.284	0.032
rh_10_25(S)	10	25	5667.49	0.252	5729.18	11.949	1.960	0.104
rh_10_30(S)	10	30	2098.89	0.276	2015.87	1251.1[a]	6.356	0.248

[a]Enable the `VINCI` option to restrict memory storage, so as to avoid running out of memory.

In Table 1, T_{rlass}, T_{hot} and T_{lawnd} represent the time consumption of three parameters of methods in `VINCI` respectively. The "rlass" uses Lasserre's method, it needs input of H-representation. The "hot" uses a Cohen&Hikey-like face enumeration scheme, it needs input of V-representation. The "lawnd" uses Lawrence's formula, it is the fatest method in `VINCI` and both descriptions are needed. From "cube_20" to "cube_40", "rlass" and "hot" cannot handle these instances in reasonable time. We did not test instances "cube_30" and "cube_40" by "lawnd", because there are too many vertices in these polytopes.

Observe that the "rlass" and "hot" methods of `VINCI` usually take much more time and space as the scale of the problem grows a bit, e.g. "cube_n($n \geq 15$)" and "rh_10_30". With H- and V- representations, the "lawnd" method is very fast for instances smaller than 20 dimensions. However, enumerating all vertices of polytopes is non-trivial, as is the dual problem of constructing the convex hull by the vertices. This process is both time-consuming and space-consuming. As a result, "lawnd" method is slower than `PolyVest` for random polytopes around 15 dimensions with only H-representation. The running times

Table 2. Statistical results of PolyVest

Instance	Average volume $\overline{v}$	Std Dev σ	95% Confidence interval $\mathcal{I} = [p, q]$	Freq on $\mathcal{I}$	Error $\epsilon = \frac{q-p}{\overline{v}}$
cube_10[a]	1024.91	41.7534	[943.077, 1106.75]	947	15.9695 %
cube_20[a]	1.04551e+6	49092.6	[9.49284e+5, 1.14173e+6]	942	18.4067 %
cube_30	1.06671e+9	5.95310e+7	[9.50024e+8, 1.18339e+9]	96	21.8769 %
cube_40	1.09328e+12	4.85772e+10	[9.98073e+11, 1.18850e+12]	95	17.4175 %
cuboid_10(S)[a]	102258	3162.13	[96060.1, 108456]	953	12.1219 %
cuboid_20(S)[a]	1.04892e+8	388574e+6	[9.72760e+7, 1.12508e+8]	953	14.5217 %
cuboid_30(S)	1.07472e+11	4.42609e+9	[9.87968e+10, 1.16147e+11]	93	16.1440 %
ran_10_30[a]	11.0079	0.413874	[10.1967, 11.8191]	946	14.7383 %
ran_10_50[a]	1.48473	4.81726e-2	[1.39031, 1.57915]	952	12.7186 %
ran_15_30	290.575	12.8392	[265.410, 315.740]	92	17.3208 %
ran_15_50	3.30084	0.145495	[3.01567, 3.58601]	96	17.2787 %
ran_20_50	1.25062	6.60574e-2	[1.12115, 1.38010]	94	20.7053 %
ran_20_100	8.79715e-3	3.144633e-4	[8.18080e-3, 9.41350e-3]	96	14.0125 %
ran_30_60	195.295	10.37041	[174.969, 215.621]	97	20.8157 %
ran_30_100	2.21532e-5	1.13182e-6	[1.99348e-5, 2.43715e-5]	98	20.0276 %
ran_40_100	3.02636e-5	1.76093e-6	[2.68121e-5, 3.3715e-5]	96	22.8091 %

[a]Estimated 1000 times with POLYVEST.

of PolyVest appear to be more 'stable'. In addition, PolyVest only has to store some constant matrices and variable vectors for sampling.

Since PolyVest is a volume estimation method instead of an exact volume computation one like VINCI, we did more tests on PolyVest to see how accurate it is. We estimated 100 times with PolyVest for each instance in Table 2 and listed the statistical results. From Table 2, we observe that the frequency on $\mathcal{I}$ is approximately 950 which means $Pr(p \leq \overline{vol(P)} \leq q) \approx 0.95$. Additionally, values of ϵ (ratio of confidence interval's range to average volume $\overline{v}$) are smaller than or around 20 %.

3.2 Result Checking

For arbitrary convex polytopes with more than 10 dimensions, there is no easy way to evaluate the accuracy of PolyVest since the exact volumes cannot be computed with tools like VINCI in reasonable time. However, we find that a simple property of geometric body is very helpful for verifying the results.

Given an arbitrary geometric body P, an obvious relation is that if P is divided into two parts P_1 and P_2, then we have $vol(P) = vol(P_1) + vol(P_2)$.

For a random convex polytope, we randomly generate a hyperplane to cut the polytope, and test if the results of `PolyVest` satisfy this relation.

Table 3 shows the results of such tests on random polytopes in different dimensions. Each polytope is tested 100 times. Values in column "Freq." are the times that $(vol(P_1) + vol(P_2))$ falls in 95 % confidence interval of $vol(P)$, and these values are all greater than 95. The error $\frac{|Sum-\overline{vol(P)}|}{\overline{vol(P)}}$ is quite small. Therefore, the outputs of `PolyVest` satisfy the relation $vol(P) = vol(P_1) + vol(P_2)$. The test results further confirm the reliability of `PolyVest`.

Table 3. Result checking

n	$\overline{vol(P)}$	95 % Confidence interval	$\overline{vol(P_1)}$	$\overline{vol(P_2)}$	Sum	Error	Freq
10	916.257	[847.229, 985.285]	498.394	414.676	913.069	0.348 %	98
20	107.976	[97.4049, 118.548]	50.4808	57.3418	107.823	0.142 %	99
30	261424	[228471, 294376]	40332.7	218637	258969	0.939 %	96
40	5.08e+11	[4.58e+11, 5.57e+11]	9.44e+10	4.15e+11	5.09e+11	0.234 %	98

3.3 The Performance of Two Hit-and-Run Method

In Table 4, t_1 and t_2 represent the time consumption of the coordinate directions and the hypersphere directions method when each method is executed 10 million times. Observe that the coordinate directions method is faster than the other one. The reason is that the hypersphere directions method has to do more vector multiplications to find intercestion points and $m \times n$ more divisions during each walk step.

Table 4. Random walk by 10 million steps

n	m	time t_1(s)	time t_2(s)
10	20	6.104	13.761
20	40	10.701	24.502
30	60	17.541	40.455
40	80	27.494	61.484

In addition, we also compare the two hit-and-run methods on accuracy. The results in Table 5 show that the relative errors and standard deviations of the coordinate directions method are smaller.

Table 5. Comparison about accuracy between two methods

		Simplified			Original		
Instance	Exact Vol v	Volume $\overline{v}$	Err $\frac{\lvert\overline{v}-v\rvert}{v}$	Std Dev σ	Volume $\overline{v}'$	Err $\frac{\lvert\overline{v}-v\rvert}{v}$	Std Dev σ'
cube_10	1024	1024.91	0.089 %	41.7534	1028.31	0.421 %	62.6198
cube_14	16384	16382.3	0.010 %	3.020	16324.6	0.363 %	1145.76
cube_20	1.04858e+6	1.04551e+6	0.293 %	49092.6	1.04426e+6	0.412 %	81699.9
rh_8_25	785.989	786.240	0.032 %	23.5826	791.594	0.713 %	50.5415
rh_10_20	13882.7	13876.3	0.046 %	473.224	13994.4	0.805 %	963.197
rh_10_25	5729.52	5736.83	0.128 %	193.715	5765.18	0.622 %	368.887
rh_10_30	2015.58	2013.08	0.124 %	62.1032	2041.60	1.291 %	124.204

3.4 The Advantage of Reutilization of Sample Points

In Table 6, we demonstrate the effectiveness of reutilization technique. Values of n_1 are the number of sample points without this technique. Since our method is a randomized algorithm, the number of sample points with this technique is not a constant. So we list average values in column n_2. With this technique, the requirement of sample points is significantly reduced.

Table 6. Reutilize Sample Points

Instance	n_1	n_2	n_2/n_1
cube_10	2016000	535105.41	26.5 %
cube_15	5856000	1721280.3	29.4 %
cube_20	12249600	3789370.7	30.9 %
rh_8_25	1040000	181091.13	17.4 %
rh_10_30	2016000	304211.03	15.1 %
cross_7	809600	78428.755	9.69 %
fm_6	5856000	955656.79	16.3 %

4 Related Works

To our knowledge, there are two implementations of volume estimation methods in literature. Liu et al. [16] developed a tool to estimate volume of convex body with a direct Monte-Carlo method. Suffered from the curse of dimensionality, it can hardly solve problems as the dimension reaches 5. The recent work [17] is an implementation of the $O^*(n^4)$ volume algorithm in [9]. The algorithm is targeted for convex bodies, and only the computational results for instances within 10 dimensions are reported. The authors also report that they could not experiment with other convex bodies than cubes, since the oracle describing the convex bodies took too long to run.

5 Conclusion

In this paper, we propose an efficient volume estimation algorithm for convex polytopes which is based on Multiphase Monte Carlo algorithm. With simplified hit-and-run method and the technique of reutilizing sample points, we considerably improve the existing algorithm for volume estimation. Our tool, `PolyVest`, can efficiently handle instances with dozens of dimensions with high accuracy, while the exact volume computation algorithms often fail on instances with over 10 dimensions. In fact, the complexity of our method (excluding rounding procedure) is $O^*(mn^3)$ and it is measured in terms of basic operations instead of oracle queries. Therefore, our method requires much less computational overhead than the theoretical algorithms.

Appendix

A About the Number of Sample Points

From Formula (1),

$$\frac{vol(P)}{vol(B(0,1))} = \prod_{i=0}^{l-1} \alpha_i = \prod_{i=0}^{l-1} \frac{step\_size}{c_i} = \frac{step\_size^l}{\prod_{i=0}^{l-1} c_i},$$

which shows that to obtain confidence interval of $vol(P)$, we only have to focus on $\prod_{i=0}^{l-1} c_i$. For a fixed P, $\{\alpha_i\}$ are fixed numbers. Let $c = \prod_{i=1}^{l} c_i$ and $\mathbb{D}(l, P)$ denote the distribution of c. With statistical results of substantial expriments on concentric balls, we observe that, when *step_size* is sufficiently large, the distribution of c_i is unbiased and its standard deviation is smaller than twice of the standard deviation of binomial distribution in dimensions below 80. Though such observation sometimes not holds when we sample on convex bodies other than balls, we still use this to approximate the distribution of c_i. Consider random variables X_i following binomial distribution $\mathbb{B}(step\_size, 1/\alpha_i)$, we have

$$E(c) = E(c_1)\dots E(c_l) = E(X_1)\dots E(X_l) = step\_size^l \prod_{i=1}^{l} \frac{1}{\alpha_i},$$

$$\begin{aligned}
D(c) &= E((c_1 \dots c_l)^2) - E(c)^2 = \prod_{i=1}^{l} (D(c_i) + E(c_i)^2) - E(c)^2 \\
&= \prod_{i=1}^{l} (4D(X_i) + E(X_i)^2) - E(c)^2 \\
&= \prod_{i=1}^{l} \frac{step\_size^2}{\alpha_i^2} (1 + \frac{4\alpha_i}{step\_size}(1 - \frac{1}{\alpha_i})) - E(c)^2 \\
&= E(c)^2(\beta - 1),
\end{aligned}$$

where $\beta = \prod_{i=1}^{l}(1 + \frac{4\alpha_i}{step\_size} - \frac{4}{step\_size})$.

Suppose $\{\xi_1, \ldots, \xi_t\}$ is a sequence of i.i.d. random variables following $\mathbb{D}(l, P)$. Notice $D(c)$, the variance of $\mathbb{D}(l, P)$, is finite because $\beta - 1 \to 0$ as $t \to \infty$. According to **central limit theorem**, we have

$$\frac{\sum_{i=1}^{t} \xi_i - tE(c)}{\sqrt{t}D(c)} \xrightarrow{d} N(0,1).$$

So we obtain the approximation of 95 % confidence interval of c, $[E(c) - \sigma\sqrt{D(c)}, E(c) + \sigma\sqrt{D(c)}]$, where $\sigma = 1.96$. And

$$Pr(\frac{vol(B(0,1))step\_size^l}{E(c) + \sigma\sqrt{D(c)}} \leq \overline{vol(P)} \leq \frac{vol(B(0,1))step\_size^l}{E(c) - \sigma\sqrt{D(c)}}) \approx 0.95.$$

Let $\epsilon \in [0,1]$ denote the ratio of confidence interval's range to exact value of $vol(P)$, that is

$$\frac{vol(B(0,1))step\_size^l}{E(c) + \sigma\sqrt{D(c)}} - \frac{vol(B(0,1))step\_size^l}{E(c) - \sigma\sqrt{D(c)}} \leq vol(P) \cdot \epsilon \tag{5}$$

$$\iff \frac{1}{E(c) - \sigma\sqrt{D(c)}} - \frac{1}{E(c) + \sigma\sqrt{D(c)}} \leq \frac{\epsilon}{E(c)} \tag{6}$$

$$\iff \frac{1}{1 - \sigma\sqrt{\beta - 1}} - \frac{1}{1 + \sigma\sqrt{\beta - 1}} \leq \epsilon \tag{7}$$

$$\iff 4\sigma^2(\beta - 1) \leq \epsilon^2(1 + \sigma^2 - \sigma^2\beta)^2 \tag{8}$$

$$\iff \epsilon^2\sigma^2\beta^2 - 2\epsilon^2(1 + \sigma^2)\beta - 4\beta + (\frac{1}{\sigma} + \sigma)^2 + 4 \geq 0. \tag{9}$$

Solve inequality (9), we get $\beta_1(\epsilon, \sigma)$, $\beta_2(\epsilon, \sigma)$ that $\beta \leq \beta_1$ and $\beta \geq \beta_2$ (ignore $\beta \geq \beta_2$ because $1 - \sigma\sqrt{\beta_2 - 1} < 0$). $\beta \leq (1 + \frac{4}{step\_size})^l$, since $1 \leq \alpha_i \leq 2$.

$$(1 + \frac{4}{step\_size})^l \leq \beta_1 \iff step\_size \geq \frac{4}{\beta_1^{1/l} - 1}, \tag{10}$$

(10) is a sufficient condition of $\beta \leq \beta_1$. Furthermore, $4/(l\beta_1^{1/l} - l)$ is nearly a constant as ϵ and σ are fixed. For example, $4/(l\beta_1^{1/l} - l) \approx 1569.2 \leq 1600$ when $\epsilon = 0.2$, $\sigma = 1.96$. So $step\_size = 1600l$ keeps the range of 95 % confidence interval of $vol(P)$ less than 20 % of the exact value of $vol(P)$.

References

1. Dyer, M., Frieze, A.: On the complexity of computing the volume of a polyhedron. SIAM J. Comput. **17**(5), 967–974 (1988)
2. Khachiyan, L.G.: On the complexity of computing the volume of a polytope. Izvestia Akad. Nauk SSSR Tekhn. Kibernet **3**, 216–217 (1988)

3. Khachiyan, L.G.: The problem of computing the volume of polytopes is NP-hard. Uspekhi Mat. Nauk. **44**, 179–180 (1989). In Russian; translation in. Russian Math. Surv. **44**(3), 199–200
4. Büeler, B., Enge, A., Fukuda, K.: Exact volume computation for polytopes: a practical study. In: Kalai, G., Ziegler, G.M. (eds.) Polytopes—Combinatorics and Computation, pp. 131–154. Birkhäuser Verlag, Birkhäuser Basel (2000)
5. Dyer, M., Frieze, A., Kannan, R.: A random polynomial time algorithm for approximating the volume of convex bodies. In: 21st Annual ACM Symposium on Theory of Computing, pp. 375–381 (1989)
6. Lovász, L., Simonovits M.: Mixing rate of Markov chains, an isoperimetric inequality, and computing a the volume. In: 31st Annual Symposium on Foundations of Computer Science, Vol. I, II, pp. 346–354 (1990)
7. Kannan, R., Lovász, L., Simonovits, M.: Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. Random Struct. Algorithms **11**(1), 1–50 (1997)
8. Lovász, L.: Hit-and-Run mixes fast. Math. Prog. **86**(3), 443–461 (1999)
9. Lovász, L., Vempala, S.: Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. J. Comput. Syst. Sci. **72**(2), 392–417 (2006)
10. http://www.math.u-bordeaux1.fr/aenge/?category=software&page=vinci
11. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer, Heidelberg (1993)
12. Goldfarb, D., Todd, M.J.: Modifications and implementation of the ellipsoid algorithm for linear programming. Math. Program. **23**(1), 1–19 (1982)
13. Smith, R.L.: Efficient Monte-Carlo procedures for generating points uniformly distributed over bounded regions. Oper. Res. **32**, 1296–1308 (1984)
14. Berbee, H.C.P., Boender, C.G.E., Ran, A.R., Scheffer, C.L., Smith, R.L., Telgen, J.: Hit-and-run algorithms for the identification of nonredundant linear inequalities. Math. Program. **37**(2), 184–207 (1987)
15. Belisle, C.J.P., Romeijn, H.E., Smith, R.L.: Hit-and-run algorithms for generating multivariate distributions. Math. Oper. Res. **18**(2), 255–266 (1993)
16. Liu, S., Zhang, J., Zhu, B.: Volume computation using a direct Monte Carlo method. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 198–209. Springer, Heidelberg (2007)
17. Lovász, L., Deák, I.: Computational results of an $O^*(n^4)$ volume algorithm. Eur. J. Oper. Res. **216**, 152–161 (2012)