A Fast and Practical Method to Estimate Volumes of Convex Polytopes

CunJing Ge, Feifei Ma, Jian Zhang and Xingming Wu

{gecj,maff,zj,wuxm}@ios.ac.cn
Institute of Software, Chinese Academy of Sciences

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 a simplified version of hit-and-run method, and employs a technique of reutilizing sample points. 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 appications such as economics, computational complexity analysis, linear systems modeling, and statistics. It is also extremely difficult to solve. Dyer et.al. [2] and Khachiyan [3, 4] proved respectively that exact volume computation is #P-hard, even for explicitly described polytopes. Büeler et.al. [5] 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 [6]. 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. [7][8][9][10]. Despite the polynomial time results and reduced complexity, there is still a lack of practical implementation. In fact, there are some difficulties in applying the above volume estimation algorithms. First, in theoretical research of randomized volume algorithms, oracle is usually used to describe the convex bodies and the above time complexity results are measured in terms of oracle queries. However, oracle is too complex and oracle queries are time-consuming. Second, there exists a very large hidden constant coefficient in the theoretical complexity [9], 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.

[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

In this paper, we mainly 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 [9] indicates. We implement a volume estimation algorithm which is based on the Multiphase Monte-Carlo method [11]. We make two improvements: (1) We simplify the original hit-and-run method [9], and find that the simplified version not only runs faster, but also is more accurate; (2) We develop a new technique to reutilize sample points. As a result, the number of sample points can be further reduced. 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 section 2, we propose our method in detail. In section 3, we show experimental results and compare our method with the exact volume computation tool `VINCI`. Finally we conclude this paper in Section 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. A convex polytope P is represented as $P = \{Ax \leq b\}$, where A is an $(m \times n)$ matrix and $A = (a_{ij}) = (a_1, \ldots, a_m)^T$. 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.

Like the original multiphase Monte-Carlo algorithm [11], our algorithm consists of three parts: rounding, subdivision and sampling.

Rounding: Find an affine transformation T on polytope Q such that $B(0,1) \subseteq T(Q) \subseteq B(0,r)$ and a constant $\gamma = \frac{vol(Q)}{vol(T(Q))}$. We can achieve $r = n+1$ by the Shallow-Cut Ellipsoid Method [12]. 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. For more details about the rounding procedure, one can refer to Appendix A.

The other two parts are to be elaborated in the following subsections.

2.1 Subdivision

To avoid curse of dimensionality[2] (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, here we select 2 as the

[2] http://en.wikipedia.org/wiki/Curse_of_dimensionality

constant. Place $l = \lceil n \log_2(n+1) \rceil$ concentric balls $\{B_i\}$ between $B(0,1)$ and $B(0,n+1)$, 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 $K_{i+1} = B_{i+1}$ which equals to $B_{i+1} \subseteq P$. That is, $\alpha_i = 2 \Leftrightarrow B_{i+1} \subseteq P$. It shows that each α_i is bounded by constants. Hence sample size would not grow too fast. Actually it is sufficient to estimate α_i by generating a polynomial number of random points.

2.2 Hit-and-run

From (1) we know that, to estimate $vol(P)$, we only have to find the approximation of α_i. 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 $\alpha_i \approx step\_size/c_i$. 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 algorithm is a random walk which has been proposed and studied for a long time [9][13][14][15]. It can generate points with almost uniform distribution in polynomial time ("almost uniform" here means that the distribution of each point is at most a constant away from the uniform in total variation distance). Hit-and-run walk 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 all directions.

Step 2. Choose a point x' uniformly on the segment in K_{k+1} of line L.

In practice, we apply a simplified version of hit-and-run algorithm. In step 1, we select a direction uniformly from n fixed directions instead of all directions. The simplified hit-and-run method is:

Step 1'. Select a line L through x uniformly over n fixed directions, $e_1, \ldots e_n$, where $e_1 = (1, 0, \ldots, 0)$, $e_2 = (0, 1, 0, \ldots, 0)$, $\ldots$, $e_n = (0, \ldots, 0, 1)$.

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 $x' = x|_{x_d=x'_d} \in K_{k+1}$. It takes much less time to find intersections of L and ∂K_{k+1} than the original version.

Algorithm 1 Hit-And-Run Sampling Algorithm

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

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

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

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

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 Formula (2) and Formula (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. However, we cannot avoid divisions in the original hit-and-run method.

2.3 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 ($step\_size - count$) points by random walk. It's easy to find out that the expected number of reduced sample points with our algorithm is

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

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.

2.4 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.

Algorithm 2 The Framework of Volume Estimation Algorithm

```
1: function EstimateVol
2:     γ ← Preprocess( )
3:     x ← O
4:     for k ← l − 1, 0 do
5:         for i ← count, step_size do
6:             x ← Walk(x, k)
7:             if x ∈ B_0 then
8:                 t_0 ← t_0 + 1
9:             else if x ∈ B_k then
10:                m ← ⌈(n/2) log_2 |x|⌉
11:                t_m ← t_m + 1
12:            end if
13:        end for
14:        count ← Σ_{i=0}^{k} t_i
15:        α_k ← step_size/count
16:        x ← 2^{−1/n} x
17:    end for
18:    return γ · unit_ball(n) · Π_{i=0}^{l−1} α_i
19: end function
```

In Algorithm 2, the formula $\lceil \frac{n}{2} \log_2 |x| \rceil$ returns index i that $x \in K_i \setminus K_{i-1}$. We use t_i to record the number of sample points that hit $K_i \setminus K_{i-1}$. Furthermore, the sum *count* of $t_0, \ldots, t_{k+1}$ is the number of reusable sample points that generated inside K_{k+1}. Then we only have to generate the rest $(step\_size - count)$ points inside K_{k+1} in the k_{th} phase. At the end of the k_{th} phase, we multiply x by a constant $2^{-\frac{1}{n}}$ to keep $x \in K_k$ and employ it as the starting point of the next phase. Finally, according to equation (1) and $\gamma = \frac{vol(Q)}{vol(P)}$, we achieve the estimation of $vol(Q)$.

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 $1600l$ for the reason discussed in

Appendix B. The experiments are performed on a workstation with 3.40GHz Intel Core i7-2600 CPU and 8GB memory.

3.1 The Performance of PolyVest

Table 1 shows the results of comparison between PolyVest and VINCI [1]. 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. Both PolyVest and VINCI use a single core.

Table 1: Comparison between PolyVest and VINCI

			PolyVest		VINCI	
Instance	n	m	Result	Time(s)	Result	Time(s)
cube_10	10	20	1016.2	0.372	1024	0.004
cube_14	14	28	17759.8	1.232	16384	0.160
cube_20	20	40	1.08805e+6	4.484	—	—
cube_30	30	60	1.0902e+9	23.197	—	—
cube_40	40	80	1.02491e+12	72.933	—	—
rh_8_25	8	25	758.89	0.252	785.989	0.884
rh_10_20	10	20	13629.8	0.604	13882.7	0.284
rh_10_25	10	25	6149.25	0.576	5729.52	5.100
rh_10_30	10	30	1998.55	0.544	—	—
cross_7	7	128	0.0250086	0.324	0.0253968	0.088
fm_6	15	59	283708	2.396	—	—
cc_8_10	8	70	156062	0.584	156816	6.764
cc_8_11	8	88	1.42674e+6	0.636	1.39181e+6	34.430

Instances tested in Table 1 are the test cases from the website of VINCI [1]. The parameters n and m represent the number of dimensions and the number of facets of each instance respectively. "cube_n" is an n-dimentional cube with side length 2, that is, the volume of "cube_n" is 2^n. "rh_n_m" is an n-dimentional polytope randomly generated with m hyperplanes. Though most instances in Table 1 have ristrictions that $n \leq 15$ and $m \leq 60$, PolyVest can handle much larger problems in reasonable time. The dash symbol "—" indicates that when VINCI computed the instance in H-representation, the computer ran out of memory. Actually, it took over 3GB memory when computing "rh_10_25" and hundreds of MB memory when computing most of the other instances. From computation of VINCI with inputs in V-representation, we obtained the results of "rh_10_30" and "fm_6", which are 2015.58 and 286113 respectively.

From Table 1, we observe that VINCI usually takes much more time and space as the scale of the problem grows a bit. On the contrary, the complexity of our algorithm is polynomial about n and m, so the running times 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 1000 times with PolyVest for each instance in Table 2 and listed the statistical results.

Table 2: Statistical Results of `PolyVest`

Instance	Exact Volume v	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	1024	1024.91	41.7534	[943.077, 1106.75]	947	15.9695%
cube_14	16384	16382.3	688.571	[15032.7, 17731.9]	949	16.4763%
cube_20	1.04858e+6	1.04551e+6	49092.6	[9.49284e+5, 1.14173e+6]	942	18.4067%
cube_30*	1.07374e+9	1.06671e+9	5.95310e+7	[9.50024e+8, 1.18339e+9]	96	21.8769%
cube_40*	1.09951e+12	1.09328e+12	4.85772e+12	[9.98073e+11, 1.18850e+12]	95	17.4175%
rh_8_25	785.989	786.240	23.5826	[740.018, 832.462]	956	11.7577%
rh_10_20	13882.7	13876.3	473.224	[12948.8, 14803.9]	953	13.3683%
rh_10_25	5729.52	5736.83	193.715	[5357.15, 6116.51]	945	13.2366%
rh_10_30	2015.58	2013.08	62.1032	[1891.35, 2134.80]	944	12.0932%
cross_7	2.53968e-2	2.53961e-2	4.76958e-4	[2.44613e-2, 2.63309e-2]	944	7.36205%
cross_9	1.41093e-3	1.41064e-3	2.80373e-5	[1.35568e-3, 1.46559e-3]	948	7.79126%
fm_6	2.86113e+5	2.85248e+5	10550.5	[2.64569e+5, 3.05927e+5]	952	14.4990%
cc_8_10	1.56816e+5	1.56699e+5	5384.98	[1.46144e+5, 1.67253e+5]	953	13.4712%
cc_8_11	1.39181e+6	1.39065e+6	49676.9	[1.29327e+6, 1.48801e+6]	953	14.0031%
simplex_10	2.75573e-7	2.75595e-7	1.08614e-8	[2.54306e-7, 2.96883e-7]	944	15.4491%
simplex_15	7.64716e-13	7.64678e-13	3.17676e-14	[7.02413e-13, 8.26942e-13]	946	16.2852%

*: "cube_30" and "cube_40" only estimated 100 times with `PolyVest` .

From Table 2, we observe that $\overline{v} \approx v$ and 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 20% for all instances except "cube_30".

3.2 Result Checking

For arbitrary convex polytopes with more than 10 dimensions, there is no easy way to evaluate the performance of `PolyVest` since the exact volumes cannot be computed with tools like `VINCI`. 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	5.07809e+11	[4.58326e+11, 5.57292e+11]	9.43749e+10	4.14623e+11	5.08997e+11	0.234%	98

3.3 The Advantages of Simplified Hit-and-run Method

In Table 4, t_1 and t_2 represent the time consumption of simplified and original hit-and-run method which each method is excuted 10 million times. We observe that simplified hit-and-run method is faster than the original one. The reason is that the original hit-and-run 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 two versions of hit-and-run methods on accuracy. The results in Table 5 show that the relative error and standard deviation of the simplified version 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
cross_7	2.53968e-2	2.53961e-2	0.003%	4.76958e-4	2.55068e-2	0.433%	1.27379e-3
cross_9	1.41093e-3	1.41064e-3	0.021%	2.80373e-5	1.41109e-3	0.011%	6.85675e-5
fm_6	286113	285248	0.302%	10550.5	287685	0.549%	23792.7
cc_8_10	156816	156699	0.075%	5384.98	156683	0.085%	10053.0
cc_8_11	1.39181e+6	1.39065e+6	0.087%	49676.9	1.40268e+6	0.781%	91891.0
simplex_10	2.75573e-7	2.75595e-7	0.008%	1.08614e-8	2.74047e-7	0.554%	2.11586e-8
simplex_15	7.64716e-13	7.64678e-13	0.005%	3.17676e-14	7.65066e-13	0.046%	6.83931e-14

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 only 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 lastest work [17] is an implementation of the $O^*(n^4)$ volume algorithm in [10]. Some interesting techniques are also discussed in the paper. However, the algorithm is targeted for convex bodies, and only the computational results for instances within 10 dimensions are reported. The authors also claim 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 and implement a practical tool. 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. However, some of our results still lack theoretical proof. It will be our primary concern in the future.

References

[1] http://www.math.u-bordeaux1.fr/~aenge/index.php?category=software&page=vinci

[2] M. Dyer, A. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing* 967-974 (1988)

[3] L.G. Khachiyan. On the complexity of computing the volume of a polytope. *Izvestia Akad. Nauk SSSR Tekhn. Kibernet* 216217 (1988)

[4] L.G. Khachiyan. The problem of computing the volume of polytopes is NP-hard. *Uspekhi Mat. Nauk 44*, 179180. In Russian; translation in Russian Math. Surveys 44, no. 3, 199200 (1989)

[5] B. Büeler, A. Enge, K. Fukuda. Exact volume computation for polytopes: a practical study. *Polytopescombinatorics and computation*. Birkhäuser Basel, 131-154. (2000)

[6] M. Dyer, A. Frieze, R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. *21st Annual ACM Symposium on Theory of Computing* 375381 (1989)

[7] L. Lovász, M. Simonovits. Mixing rate of Markov chains, an isoperimetric inequality, and computing a the volume. *31st Annual Symposium on Foundations of Computer Science*, Vol. I, II, 346–354 (1990)

[8] R. Kannan, L. Lovász, M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Structures & Algorithms*, Volume 11, Issue 1, pages 150, August 1997 (1996)

[9] L. Lovász. Hit-and-Run mixes fast. *Mathematical Programming*, Volume 86, Issue 3, pp 443-461 (1999)

[10] L. Lovász, S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *Journal of Computer and System Sciences*, Volume 72, Issue 2, pp 392417 (2006)

[11] M. Simonovits. How to compute the volume in high dimension? *Mathematical Programming*, Volume 97, Issue 1-2, pp 337-374 (2003)

[12] M. Grötschel, L. Lovász, A. Schrijver. Geometric Algorithms and Combinatorial Optimization. *Springer Verlag* (1993)

[13] R.L. Smith. Efficient Monte-Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, Vol. 32, pp. 12961308, (1984).

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

[15] C.J.P. Belisle, H. Edwin Romeijn, R.L. Smith. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, Vol. 18, No. 2, pp. 255-266 (1993)

[16] S. Liu, J. Zhang, B. Zhu. Volume computation using a direct Monte Carlo method. *Computing and Combinatorics*. Springer Berlin Heidelberg 198-209 (2007)

[17] L. Lovász, I.Deák. Computational results of an $O^*(n^4)$ volume algorithm. *European Journal of Operational Research*, Vol 216, pp. 152-161 (2012)

A Rounding

The pseudocode of rounding procedure and other preprocessings is presented in Algorithm 3. We define ellipsoid $E = \{x \in \mathbb{R}^n | (x-a)^T A^{-1}(x-a) \leq 1\}$, where A is a symmetric positive definite matrix. In function $InitEllipsoid$, we maximize each of the $2n$ linear functions $x_1, -x_1, \ldots, x_n, -x_n$ subject to $Ax \leq b$. So we get bounds $UB_1, LB_1, \ldots, UB_n, LB_n$ of each dimension of P and $2n$ vertices $v_1, \ldots, v_{2n}$ (possible that $v_i = v_j, i \neq j$). Let $o_0 = \frac{1}{2n}\sum_{i=1}^{2n} v_i$ and $r = \sqrt{\sum_{i=1}^{n}(UB_i - LB_i)^2}$. Then we obtain the initial ellipsoid $E_0(r^2 I, o_0) = B(o_0, r)$ where $o_0 \in P$(notice that P is a convex body) and $P \subseteq E_0$.

Line 3–20 of Algorithm 3 is the implementation of Shallow-Cut Ellipsoid Method [12]. It is an iterative method that generates a series of ellipsoids $\{E_i(T_i, o_i)\}$ s.t. $P \subseteq E_i$, until we find an E_k such that $E_k((n+1)^{-2}T_k, o_k) \subseteq P$.

The affine transformation is described through Line 21-24. Function $Cholesky(T_k)$ returns the Cholesky factorization L of T_k (that is, $T_k = L^T L$ and L is an upper triangular matrix), since T_k is a symmetric positive definite matrix. Notice

$$E_k(T_k, o_k) = E_k(L^T L, o_k) = \{x \in \mathbb{R}^n | ((L^T)^{-1}(x - o_k))^T (L^T)^{-1}(x - o_k) \leq 1\}.$$

Let $y = (L^T)^{-1}(x - o_k)$, then $\{y \in \mathbb{R}^n | y^T y \leq 1\} = B(0, 1)$. Thus

$$E_k(T_k, o_k) = L^T B(0, 1) + o_k.$$

Substitute x in $P = \{Ax \leq b\}$ by $x = L^T y + o_k$, we get

$$P' = \{A(L^T y + o_k) \leq b\} = \{A'y \leq b'\},\ B(0, \frac{1}{n+1}) \subseteq P' \subseteq B(0, 1), \tag{5}$$

where $A' = AL^T$, $b' = b - Ao_k$.
Resize P' by multiplying n, so $B(0, 1) \subseteq P'' = (n+1)P' \subseteq B(0, n)$

$$where\ P'' = \{A''x \leq b''\},\ A'' = AL^T,\ b'' = (n+1)(b - Ao_k). \tag{6}$$

Algorithm 3 The Ellipsoid Method and the affine transformation

function PREPROCESS
  $InitEllipsoid(r, o_0)$
  $T_0 \leftarrow r^2 \cdot I$
  $k \leftarrow 0$
  loop
    $i \leftarrow -1$
    if $o_k \notin P$ **then**
      choose i *that* $a_i x \leq b_i$ *does not hold*
    else if $E((n+1)^{-2} T_k, o_k) \nsubseteq P$ **then**
      choose i *such that* $(n+1)^{-2} a_i T_k a_i^T \leq (b_i - a_i o_k)$ *does not hold*
    end if
    if $i \geq 0$ **then**
      $c \leftarrow \frac{T_k a_i^T}{\sqrt{a_i T_k a_i^T}}$
      $o_{k+1} \leftarrow o_k - \frac{c^T}{(n+1)^2}$
      $T_{k+1} \leftarrow (1 + \frac{1}{2n^2(n+1)^2}) \frac{n^2}{n^2-1} \frac{n^2+2n}{(n+1)^2} (T_k - \frac{2cc^T}{n(n+1)})$
    else
      break loop
    end if
    $k \leftarrow k + 1$
  end loop
  $L \leftarrow Cholesky(T_k)$
  $b \leftarrow (n+1)(b - Ao_k)$
  $A \leftarrow AL^T$
  return $det(L)/(n+1)^n$
end function

The formulas in (6) are that of line 22, 23 in Algorithm 3. From (5) and (6), it is obvious that

$$\gamma = \frac{vol(P)}{vol(P'')} = \frac{det(L)}{(n+1)^n}. \tag{7}$$

So in Algorithm 3, function *Preprocess* returns the ratio of γ.

B About the Number of Sample Points

From Formula (1) we have

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

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 $\{\xi_1, \dots, \xi_t\}$ 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

$$\frac{vol(B(0,1))step\_size^l}{E(c) + \sigma\sqrt{D(c)}} - \frac{vol(B(0,1))step\_size^l}{E(c) - \sigma\sqrt{D(c)}} \leq vol(P) \cdot \epsilon \tag{8}$$

$$\iff \frac{1}{E(c) - \sigma\sqrt{D(c)}} - \frac{1}{E(c) + \sigma\sqrt{D(c)}} \leq \frac{\epsilon}{E(c)} \tag{9}$$

$$\iff \frac{1}{1 - \sigma\sqrt{\beta - 1}} - \frac{1}{1 + \sigma\sqrt{\beta - 1}} \leq \epsilon \tag{10}$$

$$\iff 4\sigma^2(\beta - 1) \leq \epsilon^2(1 + \sigma^2 - \sigma^2\beta)^2 \tag{11}$$

$$\iff \epsilon^2\sigma^2\beta^2 - 2\epsilon^2(1 + \sigma^2)\beta - 4\beta + (\frac{1}{\sigma} + \sigma)^2 + 4 \geq 0. \tag{12}$$

Solve inequality (12), 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 $1 \leq \alpha_i \leq 2$.

$$(1 + \frac{4}{step\_size})^l \leq \beta_1 \iff step\_size \geq \frac{4}{\beta_1^{1/l} - 1}, \tag{13}$$

(13) 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 $vol(P)$.