Approximate Integer Solution Counts over Linear Arithmetic Constraints

Cunjing Ge[1, 2]

[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
gecunjing@nju.edu.cn

Abstract

Counting integer solutions of linear constraints has found interesting applications in various fields. It is equivalent to the problem of counting lattice points inside a polytope. However, state-of-the-art algorithms for this problem become too slow for even a modest number of variables. In this paper, we propose a new framework to approximate the lattice counts inside a polytope with a new random-walk sampling method. The counts computed by our approach has been proved approximately bounded by a (ϵ, δ)-bound. Experiments on extensive benchmarks show that our algorithm could solve polytopes with dozens of dimensions, which significantly outperforms state-of-the-art counters.

Introduction

As one of the most fundamental type of constraints, linear constraints (LCs) have been studied thoroughly in many areas. In this paper, we consider the problem of counting approximately the number of integer solutions of a set of LCs. This problem has many applications, such as counting-based search (Zanarini and Pesant 2007; Pesant 2016), simple temporal planning (Huang et al. 2018), probabilistic program analysis (Geldenhuys, Dwyer, and Visser 2012; Luckow et al. 2014), etc.. It also includes as a special case several combinatorial counting problems that have been studied, like that of estimating the permanent of a matrix (Jerrum and Sinclair 1989; Gamarnik and Katz 2010; Harviainen, Röyskö, and Koivisto 2021), the number of contingency tables (Cryan et al. 2002; Desalvo and Zhao 2020), solutions to knapsack problems (Dyer et al. 1993), etc.. Moreover, it can be incorporated as a subroutine for #SMT (LA) (Ge et al. 2018). Since a set of LCs represents a convex polytope, its integer solutions correspond to lattice points inside the polytope. Accordingly, we do not distinguish the concepts of polytopes and sets of LCs in this paper.

It is well-known that counting lattice points in a polytope is #P-hard (Valiant 1979). On the implementation front, the first practical tool for lattice counting is LATTE (Loera et al. 2004), which is an implementation of Barvinok's algorithm (Barvinok 1993, 1994). The tool BARVINOK (Verdoolaege et al. 2007) is the successor of LATTE with an in general better performance. In practice, it often still has difficulties when the number of variables is greater than 10 (preventing many applications). The relation between the number of lattice points inside a polytope and the volume of a polytope has been studied for approximate integer counting (Ge et al. 2019). However, it is inevitable that the approximation bounds may far off from exact counts. A more recent work (Ge and Biere 2021) introduced factorization preprocessing techniques to reduce polytopes dimensionality, which are orthogonal to lattice counting, they also require polytopes in specific forms.

An algorithm for sampling lattice points in a polytope was introduced in (Kannan and Vempala 1997), which can be used to approximate the integer solution count, though we are not aware of any implementation. Since then, there have been a lot of works about sampling real points, such as Hit-and-run method (Lovász 1999; Lovász and Vempala 2006a), and approximating polytopes' volume (Lovász and Deák 2012; Cousins and Vempala 2015, 2018). As a result, the state-of-the-art volume approximation algorithms could solve general polytopes around 100 dimensions. Naturally, we wonder if they could be extended to integer cases.

The primary contribution of this paper is a novel approximate lattice counting algorithm, in detail, it includes new methods with theoretical results as follows.

- A lattice sampling method is introduced, which is a combination of Hit-and-run random walk and rejection sampling. We proved that it generates samples in distribution limited by Hit-and-run method, which is nearly uniform.
- A dynamic stopping criterion is proposed, which could be calculated by variance of approximations while running. We proved that errors of outputs approximately lie in $[1-\epsilon, 1+\epsilon]$ with probability at least $1-\delta$, given ϵ, δ.

We evaluated our algorithm on an extensive set of random and application benchmarks. We not only compared our tool with integer counters, but also with #SAT counters by translating LCs into propositional logic formulas. Experimental results show that our approach scales to polytopes up to 80 dimensions, which significantly outperforms the state-of-the-art counters. We also observe that counts computed by our algorithm are bounded well by theoretical guarantees.

Background

In this section, we first present definitions of notations, and then briefly describe the sampling and volume approximation algorithms which inspired us.

Notations and Preliminaries

Definition 1. *A linear constraint is an inequality of the form* $a_1x_1 + \cdots + a_nx_n$ op b*, where* x_i *are numeric variables,* a_i *are constant coefficients, and* op $\in \{<, \leq, >, \geq, =\}$.

Without loss of generality, a set of linear constraints can be written in the form of: $\{A\vec{x} \leq \vec{b}\}$, where A is a $m \times n$ coefficient matrix and $\vec{b}$ is a $1 \times n$ constant vector. In the view of geometry, a linear constraint is a halfspace, and a set of linear constraints is an n-dimensional polytope.

Definition 2. *An* n*-dimensional polytope is in the form of*

$$P = \{\vec{x} \in \mathbb{R}^n : A\vec{x} \leq \vec{b}\}.$$

Naturally, $\mathbb{Z}^n$ represents the set of all integer points (points with all integer coordinates). Thus integer models of the linear constraints can be represented by $\{\vec{x} \in \mathbb{Z}^n : A\vec{x} \leq \vec{b}\}$. It is the same as the integer points inside the corresponding polytope, i.e.,

$$\{\vec{x} \in \mathbb{Z}^n : A\vec{x} \leq \vec{b}\} = P \cap \mathbb{Z}^n.$$

In this paper, we assume that the polytopes are bounded, i.e., finite number of integer solutions, otherwise, it can be easily detected via Integer Linear Programming (ILP). Note that in our experiments, the running time of ILP is usually negligible compared to that of the integer counting.

Definition 3. *More notations:*

- *Let* $A = (\vec{A_1}, ..., \vec{A_m})^T$ *and* $\mathbf{h_i} = \vec{A_i}\vec{x} \leq b_i$*, given* $P = \{A\vec{x} \leq \vec{b}\}$*, i.e.,* $P = \mathbf{h_1} \cap ... \cap \mathbf{h_m}$.
- *Let Vol*(K) *denote the volume of a given convex set* K*, which is the Lebesgue measure of* $|K|$ *in Euclidian space.*
- *Let* $C(\vec{x})$ *denote the unit cube centered at* $\vec{x}$*.*
- *Let* $B(\vec{x}, r)$ *denote the ball centered at* $\vec{x}$*, of radius* r*.*

Hit-and-run Method

Hit-and-run random walk method was first introduced in (Berbee et al. 1987), whose limiting distribution is proved to be uniform. It was employed and improved for volume approximation by (Lovász 1999; Lovász and Vempala 2006a). Experiments (Ge et al. 2018) showed that a variation called Coordinate Directions Hit-and-run is more efficient in practice. Thus we also adopt this variation, which is called Hit-and-run for short in the rest of paper. It samples a real point from $\vec{p}$ in a given convex body K by the following steps:

- Select a direction from n coordinates uniformly.
- Generate the line l through $\vec{p}$ with above direction.
- Pick a next point $\vec{p'}$ uniformly from $l \cap K$.
- Start from p' and repeat above steps w times.

Earlier works (Lovász and Vempala 2006a) proved that Hit-and-run method mixes in $w = O(n^2)$ steps for a random initial point and $O(n^3)$ steps for a fixed initial point. However, further numerical studies (Lovász and Deák 2012; Ge et al. 2018) reported that $w = n$ is sufficient for nearly uniformly sampling in polytopes with dozens of dimensions.

Multiphase Monte-Carlo Algorithm

Multiphase Monte-Carlo Algorithm (MMC) is a polynomial time randomized algorithm, which was first introduced in (Dyer, Frieze, and Kannan 1991). At first, the complexity is $O^*(n^{23})$, it was reduced to $O^*(n^3)$ by a series of works (Lovász 1999; Lovász and Vempala 2006b; Cousins and Vempala 2018). It consists of the following steps:

- Employ an Ellipsoid method to obtain an affine transformation T, s.t., $B(\vec{0}, 1) \subset T(P) \subset B(\vec{0}, \rho)$, given a $\rho > n$. Note that $\text{Vol}(P) = \text{Vol}(T(P)) \cdot \det(T)$.
- Construct a series of convex bodies $K_i = T(P) \cap B(\vec{0}, 2^{i/n})$, $i = 0, ..., l$, where $l = \lceil n \log_2 \rho \rceil$. Then

$$\text{Vol}(T(P)) = \text{Vol}(K_l) = \text{Vol}(K_0) \cdot \prod_{i=0}^{l-1} \frac{\text{Vol}(K_{i+1})}{\text{Vol}(K_i)}.$$

  Specifically, $K_0 = B(\vec{0}, 1)$ and $K_l = T(P)$.
- Generate a set S_i of sample points by Hit-and-run in K_{i+1}, where $|S_i| = f(l, \epsilon, \delta)$. Then count $|K_i \cap S_i|$ and use $r_i = \frac{|K_i \cap S_i|}{|S_i|}$ to approximate the ratio $\frac{\text{Vol}(K_{i+1})}{\text{Vol}(K_i)}$.
- At last, $\text{Vol}(P) \approx \text{Vol}(B(\vec{0}, 1)) \cdot \prod_{i=0}^{l-1} r_i \cdot \det(T)$.

Note that the function $f(l, \epsilon, \delta)$ determines the number of samples with given ϵ, δ, s.t., relative errors of outputs are bounded in $[1 - \epsilon, 1 + \epsilon]$ with probability at least $1 - \delta$.

Algorithm

To apply MMC framework and Hit-and-run random walk on lattice counting problem, there are some difficulties:

- How to efficiently sampling lattice points nearly uniformly inside a polytope?
- How to construct a chain of polytopes and then approximate ratios among them like MMC?
- How many sample points are sufficient, given ϵ, δ? Could relative errors be computed while algorithm running?

In this section, we will propose new algorithms to answer above questions, with theoretical analysis.

Lattice Sampling

To sampling lattice points in a given polytope P, we apply Hit-and-run random walk method with rejection sampling.

Intuitively, a real point $\vec{p} = (p_1, ..., p_n)$ corresponds to a lattice point $[\vec{p}] = ([p_1], ..., [p_n])$. So lattice points can be generated by Hit-and-run method and number rounding, noted $[\,.\,]$. However, the distribution of lattices generated by sampling real points directly in P is not uniform. Because the probability of sampling a lattice point $\vec{u}$ closed to polytopes' facets may be smaller than a point $\vec{v}$ which $C(\vec{v}) \subset P$.

Example 1. *In Figure 1, the probability of a blue point picked by sampling directly in* P*, is smaller than a red point. Now let us consider shifting* $\mathbf{c}$ *to* $\mathbf{l_1}$*,* $\mathbf{l_2}$ *and* $\mathbf{l_3}$*. Note that* $C(u_3) \subset \mathbf{a} \cap \mathbf{b} \cap \mathbf{l_2} \subset \mathbf{a} \cap \mathbf{b} \cap \mathbf{l_3}$*, but* $C(u_3) \not\subset \mathbf{a} \cap \mathbf{b} \cap \mathbf{l_1}$*. Then the probability of picking* u_3 *by sampling real points in* $\mathbf{a} \cap \mathbf{b} \cap \mathbf{l_2}$ *or* $\mathbf{a} \cap \mathbf{b} \cap \mathbf{l_3}$ *is the same as red points.*

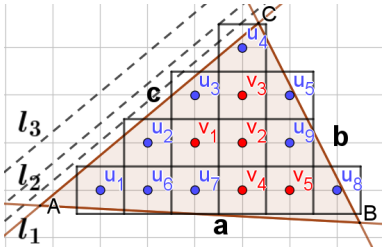


Figure 1: An illustration of shifting facets. Here $P = \triangle ABC = \mathbf{a} \cap \mathbf{b} \cap \mathbf{c}$, where $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are inequalities (half-spaces) correspond to AB, BC, AC, respectively. Inequalities $\mathbf{l_1}, \mathbf{l_2}, \mathbf{l_3}$ are parallel to $\mathbf{c}$. Red points $\{v_1, ..., v_5\}$ and blue points $\{u_1, ..., u_9\}$ are lattice points in P s.t. $C(v_i) \subset P$ and $C(u_i) \not\subset P$ respectively.

Therefore, our approach first enlarges P to P' by shifting facets of P. Then it repeatedly generates real points in P' and rejects samples whose corresponding lattice points outside P. Obviously, the larger P', the larger probability of rejection. Now we have a further question:

- How to obtain such P' as small as possible?

Naturally, P' should contain all unit cubes centered at lattice points in P, i.e.,

$$C(\vec{p}) \subset P', \forall \vec{p} \in P \cap \mathbb{Z}^n.$$

Without loss of generality, let us consider shifting the ith facet $\vec{A}_i\vec{x} \leq b_i$. The hyperplane shifting problem is equivalent to the following optimization problem

$$\begin{aligned} &\min b_i' \text{ s.t. } C(\vec{p}) \subset \vec{A}_i\vec{x} \leq b_i', \forall \vec{p} \in P \cap \mathbb{Z}^n. \\ \Leftrightarrow &\max b_i' \text{ s.t. } [C(\vec{p}) \cap \vec{A}_i\vec{x} = b_i'] \neq \emptyset, \exists \vec{p} \in P \cap \mathbb{Z}^n. \\ \Leftrightarrow &\max \vec{A}_i\vec{x} \text{ s.t. } \vec{x} \in \bigcup_{\vec{p} \in P \cap \mathbb{Z}^n} C(\vec{p}), \vec{x} \in \mathbb{R}^n. \end{aligned}$$

In the worst case, assume there is a lattice point $\vec{q}$ on the ith facet of P, i.e., $\vec{A}_i\vec{q} = b_i$. Then we have

$$\begin{aligned} &\Leftrightarrow \max \vec{A}_i\vec{x} \text{ s.t. } \vec{x} \in C(\vec{q}), \vec{x} \in \mathbb{R}^n. \\ &\Leftrightarrow \ b_i + \max \vec{A}_i\vec{x} \text{ s.t. } \vec{x} \in C(\vec{0}), \vec{x} \in \mathbb{R}^n. \end{aligned} \tag{1}$$

The optimization problem of Equation (1) can be solved by Linear Programming (LP), e.g., Simplex algorithm.

Algorithm 1 is the pseudocode of our sampling method. It first enlarges P to P' by the shifting method. Next it applies the Shallow-β-Cut Ellipsoid method on P' which is the same as MMC. It obtains an affine transformation T such that $B(0,1) \subset T(P') \subset B(0,2n)$. Then it samples a lattice point $\vec{q}$ by $[T^{-1}(\vec{p})]$, where $\vec{p}$ is a real sample point generated by Hit-and-run in $T(P')$, and T^{-1} is the inverse transformation of T. The algorithm only accepts samples inside P. At last it repeats above steps till $|S| = s$. The parameter w will be discussed later in Section .

Why we adopt an affine transformation T before random walks? Intuitively, it could transform a 'thin' polytope P' into is a well-rounded one $T(P')$. Thus it is easier for Hit-and-run walks to get out of corners.

Algorithm 1: Sample() – Sample s lattice points in P

Input: P, s
Parameter: w
Output: S

for each $\vec{A}_i\vec{x} \leq b_i$ in P **do**
  $v_i \leftarrow \text{Simplex}(\max \vec{A}_i\vec{x} \text{ s.t. } \{-\frac{1}{2} \leq x_i \leq \frac{1}{2}\})$
end for
$P' \leftarrow \{A\vec{x} \leq \vec{b} + \vec{v}\}$
$T \leftarrow \text{Ellipsoid}(P')$
$\vec{p} \leftarrow \vec{0}$, $S \leftarrow \emptyset$
repeat s **times**
  do
    $\vec{p} \leftarrow \text{HitAndRun}(T(P'), \vec{p}, w)$
    $\vec{q} \leftarrow [T^{-1}(\vec{p})]$
  while $\vec{q} \notin P$
  $S \leftarrow S \cup \{\vec{q}\}$
end repeat

The following results show that Algorithm 1 generates lattice sample points in nearly uniform.

Lemma 1. *The probability of acceptance is* $\frac{|P \cap \mathbb{Z}^n|}{Vol(P')}$, *if* Hit-and-run *is a uniform sampler.*

Proof. Assume $\vec{x}$ is generated by Hit-and-run method. Then

$$\begin{aligned} &\text{Prob}(\vec{x} \text{ accepted}) = \text{Prob}([T^{-1}(\vec{x})] \in P) \\ =&\text{Prob}(\vec{x} \in \cup_{\vec{p} \in P \cap \mathbb{Z}^n} T(C(\vec{p}))) \\ =&\frac{\text{Vol}(\cup_{\vec{p} \in P \cap \mathbb{Z}^n} T(C(\vec{p})))}{\text{Vol}(T(P'))} \\ =&\frac{\sum_{\vec{p} \in P \cap \mathbb{Z}^n} \text{Vol}(C(\vec{p}))/\det(T)}{\text{Vol}(P')/\det(T)} = \frac{|P \cap \mathbb{Z}^n|}{\text{Vol}(P')}. \end{aligned}$$

□

Theorem 1. *Each point* $\vec{x} \in P \cap \mathbb{Z}^n$ *gets picked with the same probability, if* Hit-and-run *is a uniform sampler.*

Proof. Consider an arbitrary point $\vec{x} \in P \cap \mathbb{Z}^n$. Let $\vec{p}$ represents a real point generated by Hit-and-run in $T(P')$. Then

$$\begin{aligned} \text{Prob}(\vec{x} \text{ picked}) &= \frac{\text{Prob}(\vec{p} \in T(C(\vec{x})))}{\text{Prob}(\vec{p} \text{ accepted})} \\ &= \frac{\text{Vol}(T(C(\vec{x})))}{\text{Vol}(T(P'))} \cdot \frac{\text{Vol}(P')}{|P \cap \mathbb{Z}^n|} = \frac{1}{|P \cap \mathbb{Z}^n|}. \end{aligned}$$

□

From Lemma 1, we observe that the acceptance could be very small when $|P \cap \mathbb{Z}^n| \ll \text{Vol}(P')$.

Polytopes Chain Generation

Now we consider a chain of polytopes $\{P_0, ..., P_l\}$ s.t.

$$\begin{aligned} |P \cap \mathbb{Z}^n| &= |P_0 \cap \mathbb{Z}^n| \cdot \prod_{i=0}^{l-1} \frac{|P_{i+1} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|}, \\ \frac{|P_{i+1} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|} &\in \begin{cases} [r_{min}, r_{max}] & i \leq l-2, \\ [r_{min}, 1) & i = l-1. \end{cases} \end{aligned} \tag{2}$$

Algorithm 2: Subdivision() – Obtain the polytopes chain

Input: P, s
Parameter: r_{max}, r_{min}, μ
Output: l, $\{P_i\}$, $\{S_i\}$

1: $P_0 \leftarrow$ GetRect(P)
2: $i \leftarrow 0$, $j \leftarrow 1$ and $S_0 \leftarrow \emptyset$
3: **while** $j \leq m$ **do**
4: $S_i \leftarrow$ Sample(P_i, s)
5: $H \leftarrow \mathbb{R}^n$
6: **while** $\frac{|S_i \cap H \cap \mathbf{h_j}|}{|S_i|} > r_{max}$ and $j \leq m$ **do**
7: $H \leftarrow H \cap \mathbf{h_j}$, $j \leftarrow j+1$
8: **end while**
9: $k \leftarrow \min(j, m)$
10: **if** $\frac{|S_i \cap H \cap \mathbf{h_k}|}{|S_i|} \geq r_{min}$ **then**
11: $H \leftarrow H \cap \mathbf{h_k}$, $j \leftarrow k+1$
12: **else**
13: **do**
14: $\vec{A'_k} \leftarrow \vec{A_k}$ or Disturb(A_k, μ) since second loop
15: find min b'_k s.t. $\frac{|S_i \cap H \cap \vec{A'_k}\vec{x} \leq b'_k|}{|S_i|} \geq r_{min}, b'_k \geq b_k$
16: **while** no feasible b'_k found
17: $H \leftarrow H \cap \vec{A'_k}\vec{x} \leq b'_k$
18: **end if**
19: $P_{i+1} \leftarrow P_i \cap H$
20: $i \leftarrow i+1$
21: **end while**
22: **return** i, $\{P_0, ..., P_i\}$, $\{S_0, ..., S_i\}$

where $[r_{min}, r_{max}]$ bounds ratios close to $\frac{1}{2}$, like $[0.4, 0.6]$. If ratios are close to 0, the computational cost of generating points in $P_{i+1} \cap \mathbb{Z}^n$ by sampling in $P_i \cap \mathbb{Z}^n$ will increase. On the other hand, l will be a large number when ratios are close to 1, which is also not computational-wise. Algorithm 2 presents our method for constructing such P_is.

Recall that in MMC, it eventually approximates the ratio between volume of P and an inner ball $B(\vec{0}, 1)$ whose exact volume is easy to compute. It constructs a series of convex body K_i inside P. Lemma 1 indicates that the smaller polytope, the more difficult to sampling lattice points, naturally, we would like to construct polytopes chain outside P. Our approach starts from an n-dimensional rectangle $P_0 = Rect(P) \supset P$, whose exact lattice count is also easy to obtain. Next it constructs $P_1 \supset P$ by adding new cutting constraints on P_0, s.t. $\frac{|P_1 \cap \mathbb{Z}^n|}{|P_0 \cap \mathbb{Z}^n|}$ close to $\frac{1}{2}$. Then it repeatedly generates $P_1 \supset P_2 \supset ...$ until a polytope $P_l = P$ found.

- How to find cutting constraints to halve P_is?

Example 2. *In Figure 2, given $P = \triangle ABC = \mathbf{a} \cap \mathbf{b} \cap \mathbf{c}$, and $P_0 = ADEF \supset P$. Now we try to cut P_0 with* **a**, **b** *and* **c**. *We observe that $\frac{|P_0 \cap \mathbf{a} \cap \mathbf{b} \cap \mathbb{Z}^n|}{|P_0 \cap \mathbb{Z}^n|} = \frac{10}{15} > r_{max}$ and $\frac{|P_0 \cap \mathbf{a} \cap \mathbf{b} \cap \mathbf{c} \cap \mathbb{Z}^n|}{|P_0 \cap \mathbb{Z}^n|} = \frac{4}{15} < r_{min}$. Then we find* **d** *parallel to* **c** *s.t. $\frac{|P_0 \cap \mathbf{a} \cap \mathbf{b} \cap \mathbf{d} \cap \mathbb{Z}^n|}{|P_0 \cap \mathbb{Z}^n|} = \frac{8}{15}$. Thus $P_1 = P_0 \cap \mathbf{a} \cap \mathbf{b} \cap \mathbf{d}$.*

Suppose that we already have $P_0 \supset ... \supset P_i$ which $P_i \subset \mathbf{h_1} \cap ... \cap \mathbf{h_{j-1}}$ and $P_i \not\subset \mathbf{h_j}$. Then cutting constraints for constructing P_{i+1} are found by the following steps:

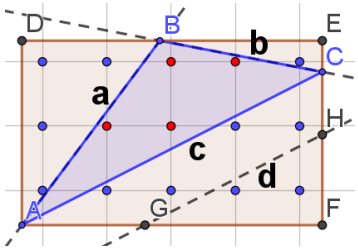


Figure 2: An example of constructing $P_0 = ADEF$, $P_1 = ABCHG$ and $P_2 = \triangle ABC = P$.

- **Step 1.** Add constraints $\mathbf{h_j}$, $\mathbf{h_{j+1}}$,... repeatedly until a k is found s.t. $\frac{|P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_k} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|} \leq r_{max}$ or $k = m$.
- **Step 2.** If $\frac{|P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_k} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|} \geq r_{min}$, then $P_{i+1} = P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_k}$ has been found. Note that $P_{i+1} = P_l = P$ when $k = m$.
- **Step 3.** Otherwise, it indicates that $\mathbf{h_k}$ over-cuts the solution space. Then we find an $\mathbf{h'_k} = \vec{A'_k}\vec{x} \leq b'_k$ (almost) parallel to $\mathbf{h_k}$ s.t. $r_{min} \leq \frac{|P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h'_k} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|} \leq r_{max}$. At last, let $P_{i+1} = P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_{k-1}} \cap \mathbf{h'_k}$.

About above steps, we may naturally ask:

- How to determine the value of $\frac{|P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_k} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|}$?

Algorithm 2 samples lattice points S_i in P_i and then approximates $\frac{|P_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_k} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|}$ via $\frac{|S_i \cap \mathbf{h_j} \cap ... \cap \mathbf{h_k}|}{|S_i|}$. Since we aim to obtain P_{i+1} s.t. $\frac{|P_{i+1} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|}$ close to $\frac{1}{2}$, it is not necessary to approximate very accurately with a mass of samples.

- How to find $\mathbf{h'_k}$ in Step 3?

Line 13 to 16 in Algorithm 2 is the loop of finding $\mathbf{h'_k}$. At the first time of loop, it sets $\vec{A'_k} = \vec{A_j}$ and searches the minimum $b'_k \geq b_k$ s.t. $\frac{|S_i \cap H \cap \vec{A'_k}\vec{x} \leq b'_k|}{|S_i|} \geq r_{min}$. We then compute and sort $D = \{d : d = \vec{A_k}\vec{p},\ \forall \vec{p} \in S_i \cap H\}$. Thus searching b'_k is equivalent to scanning D, whose time complexity is $O(|D|) = O(|S_i \cap H|) = O(s)$.

Note that there may be no feasible b'_k, as for certain y, $\frac{|S_i \cap H \cap \vec{A_k}\vec{x} \leq y|}{|S_i|} > r_{max}$, $\frac{|S_i \cap H \cap \vec{A_k}\vec{x} < y|}{|S_i|} < r_{min}$. For example, $|x_1 + x_2 = 0.99 \cap \mathbb{Z}^2| = 0$ and $|x_1 + x_2 = 1 \cap \mathbb{Z}^2| = \infty$. Therefore, if our algorithm fails to find a feasible b'_k once, it will generate $\vec{A'_k} = \{a'_{k1}, ..., a'_{kn}\}$ by disturbing $\vec{A_k}$, i.e., $a'_{ki} \in [a_{ki} - \mu, a_{ki} + \mu]$, where $\mu \in \mathbb{R}$ is a small constant. In practice, the loop in line 13–16 (Algorithm 2) usually finds a feasible b'_k by disturbing $\vec{A'_k}$ once, occasionally twice, though the loop may not stop in theory in worst cases.

With respect to the size of l, it is easy to find the following result as every P_{i+1} is constructed by nearly halve P_i.

Theorem 2. *The length l of the chain $P_0, .., P_l$ constructed by Algorithm 2 is in $O(\log_2 |P_0 \cap \mathbb{Z}^n|)$ in the worst case.*

Dynamic Stopping Criterion

Approximating $|P \cap \mathbb{Z}^n|$ is factorized into approximating a series ratios $\frac{|P_{i+1} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|}$ by Equation (2). Naturally, we could approximate ratios via $\frac{|P_{i+1} \cap S_i|}{|S_i|}$, where S_i is a set of lattice points sampled in P_i by Algorithm 1. A key question rises:

- How many sample points is sufficient to approximate $|P \cap \mathbb{Z}^n|$ with certain guarantees, like an (ϵ, δ)-bound?

Let R_i denote the random variable of $\frac{|P_{i+1} \cap S_i|}{|S_i|}$, and $R = \prod_{i=0}^{l-1} R_i$. Note that R_is are mutually independent, since for each S_i, the random walk starts from origin $\vec{0} \in T(P')$ (see Algorithm 1). Thus we have the variance of R:

$$\begin{aligned} \mathrm{Var}(R) &= \mathrm{Var}(\prod R_i) = \mathrm{E}((\prod R_i)^2) - [\mathrm{E}(\prod R_i)]^2 \\ &= \prod \mathrm{E}(R_i^2) - [\prod \mathrm{E}(R_i)]^2 \\ &= \prod [\mathrm{Var}(R_i) + \mathrm{E}(R_i)^2] - \prod [\mathrm{E}(R_i)]^2. \end{aligned} \quad (3)$$

From Chebyshev inequality, we have:

$$\begin{aligned} &\mathrm{Prob}(|\frac{R - \mathrm{E}(R)}{E(R)}| \geq \epsilon) \leq \frac{\mathrm{Var}(R)}{\epsilon^2 \cdot E(R)^2} \leq \delta \\ &\Rightarrow \mathrm{Var}(R) \leq \delta \cdot \epsilon^2 \cdot E(R)^2. \end{aligned} \quad (4)$$

Equation (4) shows when the approximate result lies in $[(1-\epsilon)|P \cap \mathbb{Z}^n|, (1+\epsilon)|P \cap \mathbb{Z}^n|]$ with probability at least $1 - \delta$, i.e., satisfies an (ϵ, δ)-bound. Thus we adopt Equation (4) as the stopping criterion of approximation.

Given a set of sample S_i, let $r_i = |P_{i+1} \cap S_i|/|S_i|$ and $r = \prod_{i=0}^{l-1} r_i$. We use r and r_is to approximately represent $E(R_i)$s and $E(R)$ respectively (Lemma 3 shows that such substitutions are safe). Then we split S_i into N groups $\{S_{ij}\}$ with the same size s/γ, where $N = |S_i| \cdot \gamma / s$ is the number of groups. Let $r_{ij} = |P_{i+1} \cap S_{ij}|/|S_{ij}|$ and R_{ij} denote the random variable of r_{ij}. If R_{ij}s are mutually independent and follow the same distribution, we have

$$\begin{aligned} \mathrm{Var}(R_i) &= \mathrm{Var}(\frac{\sum_{j=1}^{N} R_{ij}}{N}) = \frac{1}{N^2} \sum_{j=1}^{N} \mathrm{Var}(R_{ij}) \\ &= \frac{\mathrm{Var}(R_{i1})}{N} \approx \frac{1}{N} \sum_{j=1}^{N} \frac{(r_{ij} - r_i)^2}{N-1}. \end{aligned}$$

Note that R_{ij} can be exactly mutually independent if random walks start from a fixed point, however, it is not actually necessary. Let $v_i = \sum \frac{(r_{ij} - r_i)^2}{N(N-1)}$. As a result, an approximate stopping criterion is obtained

$$\mathrm{Var}(R) \approx \prod (v_i + r_i^2) - r^2 \leq \delta \cdot \epsilon^2 \cdot r^2. \quad (5)$$

The pseudocode of the main framework is presented as Algorithm 3. It first generates s sample points for each P_i and then computes r_is, v_is, r and v. If Equation (5) satisfies, it returns $|P_0 \cap \mathbb{Z}^n| \cdot r$, otherwise, it repeats above steps.

Lemma 2. $\lim_{|S_i| \to \infty} r_i = \frac{|P_{i+1} \cap \mathbb{Z}^n|}{|P_i \cap \mathbb{Z}^n|}$ *and* $\lim_{|S| \to \infty} r = \frac{|P \cap \mathbb{Z}^n|}{|P_0 \cap \mathbb{Z}^n|}$*, if Hit-and-run is a uniform sampler.*

Algorithm 3: Approximate Lattice Counts

Input: P
Parameter: ϵ, δ, s, γ
Output: $|P \cap \mathbb{Z}^n|$

$(l, \{P_i\}, \{S_i\}) \leftarrow$ Subdivision(P, s)
$N \leftarrow 0$
do
  $N \leftarrow N + \gamma$
  for $i = 0$ to $l - 1$ **do**
    $S_i \leftarrow S_i \cup$ Sample(P_i, s)
    $r_i \leftarrow |P_{i+1} \cap S_i|/|S_i|$
    Split S_i into N groups $S_{i1}, ..., S_{iN}$
    $r_{ij} \leftarrow |P_{i+1} \cap S_{ij}|/|S_{ij}|, \quad j \in \{1, ..., N\}$
    $v_i \leftarrow \sum_{j=1}^{N} \frac{(r_{ij} - r_i)^2}{N(N-1)}$
  end for
  $r \leftarrow \prod_{i=0}^{l-1} r_i$
  $v \leftarrow \prod_{i=0}^{l-1} (v_i + r_i)^2 - r^2$
while $v \leq \delta \cdot \epsilon^2 \cdot r^2$
return $|P_0 \cap \mathbb{Z}^n| \cdot r$

Proof. Note that sampling uniform in P_i and then count the number of samples in P_{i+1} is a Bernoulli trial. □

Lemma 3. *Equation (4) and (5) are approximately equivalent, regardless of the difference between r and $E(R)$.*

Proof. Let $c = r/E(R)$ and $c_i = r_i/E(R_i)$ represent the differences. Since $r_i = c_i \cdot E(R_i) = E(c_i R_i)$, we have

$$v_i \approx \mathrm{Var}(c_i R_i) = c_i^2 \cdot \mathrm{Var}(R_i).$$

Equation (4) can be transformed into

$$\begin{aligned} \delta \cdot \epsilon^2 &\geq \frac{\prod(\mathrm{Var}(R_i) + E(R_i)^2)}{E(R)^2} - 1 \\ &\approx \frac{c^2}{E(R)^2} \cdot \prod \frac{\mathrm{Var}(R_i) + E(R_i)^2}{c_i^2} - 1 \\ &= \frac{\prod(v_i + r_i^2)}{r^2} - 1. \end{aligned} \quad (6)$$

Note that Equation (6) is the same as Equation (5). □

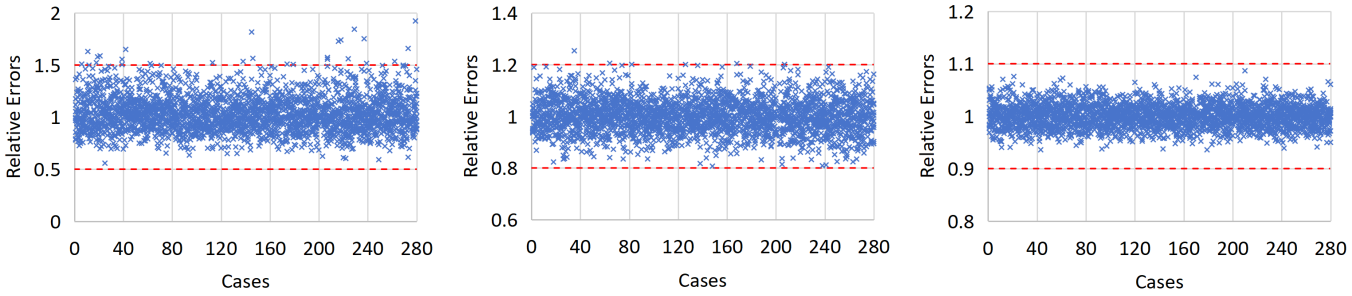
From Lemma 2, 3 and Equation (5), we have

Theorem 3. *The output of Algorithm 3 is approximately bounded in an (ϵ, δ)-bound.*

Implementation Details

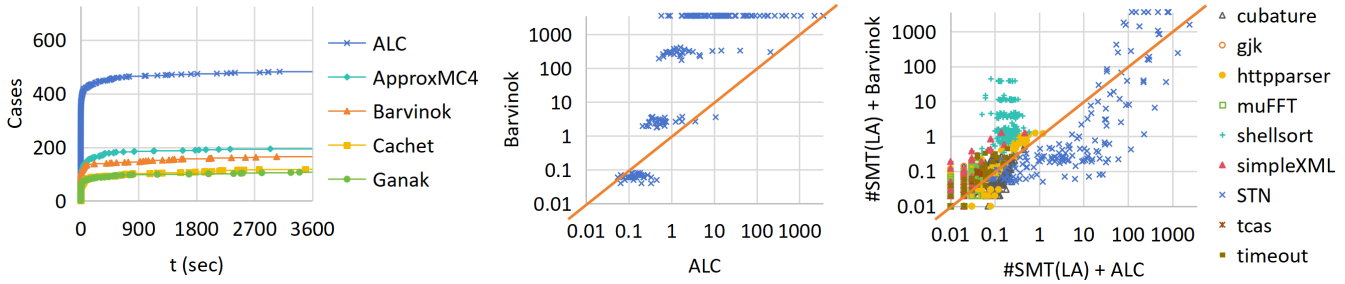
The setting of parameters in Algorithm 1, 2 and 3 are listed with explanations as the following:

- $\epsilon = 0.2$ and $\delta = 0.1$. They determine the bounds of counts computed by our approach. Experimental results with more pairs of values, such as $(0.5, 0.1)$ and $(0.1, 0.05)$, can be found in Section .
- $w = n$. It controls the number of Hit-and-run walks per real sample point. Earlier theoretical results (Lovász and Vempala 2006a) showed the upper bounds on w in the Markov chain is $O(n^2)$ for a random initial point and



(a) $\epsilon = 0.5, \delta = 0.1$ (b) $\epsilon = 0.2, \delta = 0.1$ (c) $\epsilon = 0.1, \delta = 0.05$

Figure 3: Quality of counts computed by ALC with different ϵ and δ on cases whose exact counts are available. Each case was experimented 10 times, i.e., 10 data points per case. The average running times of experiments in (a) (b) (c) are 0.19s ($\epsilon = 0.5$, $\delta = 0.1$), 0.81s ($\epsilon = 0.2$, $\delta = 0.1$), 5.73s ($\epsilon = 0.1$, $\delta = 0.05$) respectively.



(a) Random polytopes. (b) Rotated thin rectangles. (c) Application instances.

Figure 4: Performance comparisons among tools on different families of benchmarks.

$O(n^3)$ for a fixed initial point. However, further numerical studies (Lovász and Deák 2012; Ge et al. 2018) reported that $w = n$ is sufficient on polytopes with dozens of dimensions. They also tried $w = 2n$ and $w = n \ln n$, but no visible improvement. Thus we adopt $w = n$.

- $s = 2/(\delta \cdot \epsilon^2)$: It controls the number of samples in one round. We select this value, as $1/(\delta \cdot \epsilon^2)$ uniform samples are sufficient to approximate r_i in (ϵ, δ)-bound. Note that the total number of samples is determined by the stopping criterion instead of s.
- $r_{min} = 0.4$, $r_{max} = 0.6$, $\mu = 0.005$ and $\gamma = 10$.

In Algorithm 2, $P_0 = Rect(P)$ can be easily computed by LP or ILP. Naturally, LP is cheaper than ILP, but the rectangle generated by ILP is smaller. In practice, the cost of ILP is usually negligible compared to entire counting algorithm.

In Algorithm 2 and 3, samples in $S_i \cap P_{i+1}$ can be reutilized in S_{i+1}. Thus we only have to generate $s - |S_i \cap P_{i+1}|$ new samples for S_{i+1}. (Ge et al. 2018) proved that this technique has no side-effect on errors for approximating ratios.

Evaluation

We implemented a prototype tool called ApproxLatCount (ALC) [1] in C++. Furthermore, we integrated ALC into a DPLL(T)-based #SMT(LA) counter (Ge et al. 2018). Experiments were conducted on Intel(R) Xeon(R) Gold 6248 @ 2.50GHz CPUs with a time limit of 3600 seconds and memory limit of 4 GB per benchmark. The setting of parameters of ALC has already been presented and discussed in Section . The benchmark set consists of three parts:

- Random Polytopes: We generated 726 random polytopes with three parameters (m, n, λ), where n ranges from 3 to 100, $m \in \{n/2, n, 2n\}$ and $\lambda \in \{2^0, 2^1, ..., 2^{10}\}$. A benchmark is in the form of $\{A\vec{x} \leq \vec{b}, -\lambda \leq x_i \leq \lambda\}$, where $a_{ij} \in [-10, 10] \cap \mathbb{Z}$ and $b_i \in [-\lambda, \lambda] \cap \mathbb{Z}$.
- Rotated Thin Rectangles: To evaluate the quality of approximations on "thin" polytopes, 180 n-dimensional rectangles $\{-1000 \leq x_1 \leq 1000, -\tau \leq x_i \leq \tau, i \geq 2\}$ were generated and then rotated randomly, where $n \in \{3, ..., 8\}$ and $\tau \in \{0.1, 0.2, ..., 2.9, 3.0\}$.
- Application Instances: We adopted 4131 benchmarks (Ge and Biere 2021) from program analysis and simple temporal planning. The domain of variables is $[-32, 31]$.

We compared our tool ALC with the state-of-the-art integer counter Barvinok (Verdoolaege et al. 2007). On random polytopes, we further compared our approach with the state-of-the-art propositional model counters ApproxMC4 (Soos and Meel 2019), Cachet (Sang et al. 2004), and Ganak (Sharma et al. 2019). We used the default settings of ApproxMC4 ($\epsilon = 0.8$, $\delta = 0.2$) and Ganak ($\delta = 0.05$). Note that they require CNF formulas as inputs. Thus we first translated linear constraints into bit-vector formulas, and then translated into propositional CNF

[1] Source code of ALC and experimental data including benchmarks can be found at https://github.com/bearben/ALC.

	Dim n	3	4	5	6	7	8	10	12	14	15	20	30	40	50	60	70	80
ALC	#solve	**33**	**33**	**33**	**33**	**33**	**33**	**33**	**30**	**29**	**28**	**22**	**14**	**11**	**10**	**5**	**4**	**1**
	avg. $\bar{t}$	**0.03**	**0.05**	**0.07**	**0.19**	**0.65**	**0.50**	**85.6**	**42.6**	**48.8**	**151**	**156**	**286**	**249**	**1057**	**515**	**1684**	**3090**
	avg. $\bar{l}$	1.6	3.2	4.2	6.1	7.4	8.2	12.3	13.4	15.8	17.6	21.6	24.7	31.6	40.4	31.4	40.3	44
Barvinok	#solve	**33**	**33**	**33**	**33**	22	11	0	0	0	0	0	0	0	0	0	0	0
	avg. $\bar{t}$	0.22	0.24	2.72	105	1052	1158	—	—	—	—	—	—	—	—	—	—	—
Cachet	#solve	27	18	17	13	11	9	6	3	2	0	0	0	0	0	0	0	0
	avg. $\bar{t}$	161	71.8	537	396	291	434	601	483	2159	—	—	—	—	—	—	—	—
Ganak	#solve	25	17	13	11	9	8	5	3	3	0	0	0	0	0	0	0	0
	avg. $\bar{t}$	68.7	256	9.4	187	27.6	198	169	390	2909	—	—	—	—	—	—	—	—
ApproxMC4	#solve	**33**	**33**	32	22	19	13	10	5	4	3	0	0	0	0	0	0	0
	avg. $\bar{t}$	1.16	9.78	81.3	98.4	136	121	580	608	673	1001	—	—	—	—	—	—	—

Table 1: More statistics of performance on random polytopes with respect to n (33 cases for each n, experiment once per case).

with BOOLECTOR (Niemetz et al. 2018). Translation times are not included in the running times.

Figures 3 (a) (b) (c) show the relative errors (y-axis) of counts computed by ALC with different (ϵ, δ) settings. The experiments were conducted on random polytopes (case $91 \sim 280$) and rotated thin rectangles (case $1 \sim 90$) whose exact counts could be obtained by BARVINOK and CACHET. We run ALC 10 times on each cases. So there are 10 data points per case, 2800 data points per figure. We observe that the counts computed by ALC are bounded well. For example, in Figure 3 (b), relative errors should lie in $[0.8, 1.2]$ with probability at least 90% with $\epsilon = 0.2$, $\delta = 0.1$.

Figures 4 (a) (b) (c) compare running times among tools on different families of benchmarks. In general, ALC significantly outperforms other tools. On random polytopes, more results with respect to n are listed in Table 1, which will be discussed later. Figure 4 (b) present the results on rotated thin rectangles. Note that none of cases in this family was solved in timeout by APPROXMC4, CACHET or GANAK, due to larger coefficients and variable domains. We observe jumps regarding running times of BARVINOK, as n increases. Figure 4 (c) presents the results of comparisons over application instances which are all SMT(LA) formulas. Since we only integrated ALC and BARVINOK into the #SMT(LA) counter, we did not compare with other tools. Note that 'STN' is the family of simple temporal planning benchmarks, others are all generated by analyzing C++ programs. We find that most benchmarks were solved in one second by both tools, except 'shellsort' and 'STN' instances. On 'shellsort' instances, ALC significantly outperforms BARVINOK. On 'STN' instances, ALC eventually gains upper hand as the dimensionality increases.

Table 1 lists the number of solved cases and average running times (exclude timeout cases) with respect to n. For each n, there are 33 benchmarks. We find that ALC could handle random polytopes up to 80 dimensions. "Avg. $\bar{l}$" means the average length of polytopes chain (exclude timeout cases), which grows nearly linear. Note that APPROXMC4, CACHET and GANAK could solve cases with more variables (max to 15) than BARVINOK here, due to benchmarks with $\lambda = 1$, i.e., $-1 \leq x_i \leq 1$, which are in favor of propositional model counters.

Related Works

There are a few related works which also investigate approximate integer solution counting problem. In (Kannan and Vempala 1997), an algorithm for sampling lattice points in a polytope was introduced. Similar to Algorithm 1, it considers an enlarged polytope P'' for real points sampling and then rejects samples outside P, where

$$P'' = \{\vec{x} : \vec{A_i}\vec{x} \leq b_i + (c + \sqrt{2 \log m})|\vec{A_i}|)\},$$

$c = \sqrt{\ln \frac{4}{\varepsilon}}$ and ε is the variational difference between the uniform density and the probability density of real points sampling. As a result, they proved that there exists a polynomial time algorithm for nearly uniform lattice sampling if $b_i \in \Omega(n\sqrt{m}|\vec{A_i}|)$. However, in practice, such condition is often too loose. For example, benchmarks considered in Section are usually smaller, i.e., $b_i < n\sqrt{m}|\vec{A_i}|$, especially when $n \geq 10$, which has a higher difficulty in sampling. Also note that P' computed by our approach is tighter than P''. Thus the probability of rejection by sampling in P' is lower than in P''. In addition, back to the time of this work published, the best real points sampler is only with time complexity of $O^*(n^5)$. Nowadays, the state-of-the-art real points sampler is in $O^*(n^3)$.

A more recent work (Ge and Biere 2021) introduced factorization preprocessing techniques to reduce polytopes dimensionality. Suppose a polytope P has been factorized into $F_1, ..., F_k$, and $|P \cap \mathbb{Z}^n| = \prod_{i=1}^{k} |F_i \cap \mathbb{Z}^{n_i}|$, where n_i represents the dimensionality of F_i. To approximate $|P \cap \mathbb{Z}^n|$ with given ϵ, δ, we have to approximate counts in F_i with smaller ϵ', δ'. It indicates that factorization techniques integrated with ALC may not as effective as with exact counters.

Conclusion

In this paper, a new approximate lattice counting framework is introduced, with a new lattice sampling method and dynamic stopping criterion. Experimental results show that our algorithm significantly outperforms the state-of-the-art counters, with low errors. Since our sampling method is limited by the Hit-and-run random walk, which is only a nearly uniform sampler, we are interested in an efficient method to test the uniformity of samplers in the future.

Acknowledgments

This work is supported by National Key R&D Program of China (2022ZD0116600). Cunjing Ge is supported by the National Natural Science Foundation of China (62202218), and is sponsored by CCF-Huawei Populus Grove Fund (CCF-HuaweiFM202309).

References

Barvinok, A. I. 1993. Computing the Volume, Counting Integral Points, and Exponential Sums. *Discrete & Computational Geometry*, 10: 123–141.

Barvinok, A. I. 1994. Computing the Ehrhart Polynomial of a Convex Lattice Polytope. *Discrete & Computational Geometry*, 12: 35–48.

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

Cousins, B.; and Vempala, S. S. 2015. Bypassing KLS: Gaussian Cooling and an O*(n3) Volume Algorithm. In Servedio, R. A.; and Rubinfeld, R., eds., *Proc. STOC*, 539–548. ACM.

Cousins, B.; and Vempala, S. S. 2018. Gaussian Cooling and O*(n3) Algorithms for Volume and Gaussian Volume. *SIAM J. Comput.*, 47(3): 1237–1273.

Cryan, M.; Dyer, M. E.; Goldberg, L. A.; Jerrum, M.; and Martin, R. A. 2002. Rapidly Mixing Markov Chains for Sampling Contingency Tables with a Constant Number of Rows. In *Proc. FOCS*, 711–720. IEEE Computer Society.

Desalvo, S.; and Zhao, J. 2020. Random sampling of contingency tables via probabilistic divide-and-conquer. *Comput. Stat.*, 35(2): 837–869.

Dyer, M. E.; Frieze, A. M.; and Kannan, R. 1991. A Random Polynomial Time Algorithm for Approximating the Volume of Convex Bodies. *J. ACM*, 38(1): 1–17.

Dyer, M. E.; Frieze, A. M.; Kannan, R.; Kapoor, A.; Perkovic, L.; and Vazirani, U. V. 1993. A Mildly Exponential Time Algorithm for Approximating the Number of Solutions to a Multidimensional Knapsack Problem. *Comb. Probab. Comput.*, 2: 271–284.

Gamarnik, D.; and Katz, D. 2010. A deterministic approximation algorithm for computing the permanent of a 0, 1 matrix. *J. Comput. Syst. Sci.*, 76(8): 879–883.

Ge, C.; and Biere, A. 2021. Decomposition Strategies to Count Integer Solutions over Linear Constraints. In Zhou, Z., ed., *Proc. of IJCAI*, 1389–1395. ijcai.org.

Ge, C.; Ma, F.; Ma, X.; Zhang, F.; Huang, P.; and Zhang, J. 2019. Approximating Integer Solution Counting via Space Quantification for Linear Constraints. In Kraus, S., ed., *Proc. of IJCAI*, 1697–1703. ijcai.org.

Ge, C.; Ma, F.; Zhang, P.; and Zhang, J. 2018. Computing and estimating the volume of the solution space of SMT(LA) constraints. *Theor. Comput. Sci.*, 743: 110–129.

Geldenhuys, J.; Dwyer, M. B.; and Visser, W. 2012. Probabilistic symbolic execution. In *Proc. of ISSTA*, 166–176.

Harviainen, J.; Röyskö, A.; and Koivisto, M. 2021. Approximating the Permanent with Deep Rejection Sampling. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Proc. of NeurIPS*, 213–224.

Huang, A.; Lloyd, L.; Omar, M.; and Boerkoel, J. C. 2018. New Perspectives on Flexibility in Simple Temporal Planning. In *Proc. of ICAPS*, 123–131.

Jerrum, M.; and Sinclair, A. 1989. Approximating the Permanent. *SIAM J. Comput.*, 18(6): 1149–1178.

Kannan, R.; and Vempala, S. 1997. Sampling Lattice Points. In *Proc. of STOC*, 696–700.

Loera, J. A. D.; Hemmecke, R.; Tauzer, J.; and Yoshida, R. 2004. Effective lattice point counting in rational convex polytopes. *J. Symb. Comput.*, 38(4): 1273–1302.

Lovász, L. 1999. Hit-and-run mixes fast. *Math. Program.*, 86(3): 443–461.

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

Lovász, L.; and Vempala, S. 2006a. Hit-and-Run from a Corner. *SIAM J. Comput.*, 35(4): 985–1005.

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

Luckow, K. S.; Pasareanu, C. S.; Dwyer, M. B.; Filieri, A.; and Visser, W. 2014. Exact and approximate probabilistic symbolic execution for nondeterministic programs. In *Proc. of ASE*, 575–586.

Niemetz, A.; Preiner, M.; Wolf, C.; and Biere, A. 2018. Btor2, BtorMC and Boolector 3.0. In Chockler, H.; and Weissenbacher, G., eds., *Proc. of CAV*, volume 10981 of *Lecture Notes in Computer Science*, 587–595. Springer.

Pesant, G. 2016. Counting-Based Search for Constraint Optimization Problems. In *Proc. of AAAI*, 3441–3448.

Sang, T.; Bacchus, F.; Beame, P.; Kautz, H. A.; and Pitassi, T. 2004. Combining Component Caching and Clause Learning for Effective Model Counting. In *Proc. of SAT*.

Sharma, S.; Roy, S.; Soos, M.; and Meel, K. S. 2019. GANAK: A Scalable Probabilistic Exact Model Counter. In Kraus, S., ed., *Proc. of IJCAI*, 1169–1176. ijcai.org.

Soos, M.; and Meel, K. S. 2019. BIRD: Engineering an Efficient CNF-XOR SAT Solver and Its Applications to Approximate Model Counting. In *Proc. of AAAI*, 1592–1599. AAAI Press.

Valiant, L. G. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, 8(3): 410–421.

Verdoolaege, S.; Seghir, R.; Beyls, K.; Loechner, V.; and Bruynooghe, M. 2007. Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions. *Algorithmica*, 48(1): 37–66.

Zanarini, A.; and Pesant, G. 2007. Solution Counting Algorithms for Constraint-Centered Search Heuristics. In *Proc. of CP*, 743–757.