Theoretical Computer Science 743 (2018) 110–129

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Computing and estimating the volume of the solution space of SMT(LA) constraints

Cunjing Ge [a,c], Feifei Ma [a,b,c,*], Peng Zhang [d], Jian Zhang [a,d]

[a] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[b] Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, China
[c] University of Chinese Academy of Sciences, China
[d] School of Computer Science and Technology, Shandong University, China

ARTICLE INFO

Article history:
Received 4 November 2015
Received in revised form 16 October 2016
Accepted 28 October 2016
Available online 15 November 2016

Keywords:
SMT
Volume
Counting
Convex polytope

ABSTRACT

The satisfiability modulo theories (SMT) problem is a decision problem, i.e., deciding the satisfiability of logical formulas with respect to combinations of background theories (like reals, integers, arrays, bit-vectors). In this paper, we study the counting version of SMT with respect to linear arithmetic – SMT(LA), which generalizes both model counting and volume computation of convex polytopes. We describe a method for estimating the volume of convex polytopes based on the Multiphase Monte-Carlo method. It employs a new technique to reutilize random points, so that the number of random points can be significantly reduced. We prove that the reutilization technique has no side-effect on the error. We also investigate a simplified version of hit-and-run method: the coordinate directions method. Based on volume estimation method for polytopes, we present an approach to estimating the volume of the solution space of SMT(LA) formulas. It employs a heuristic strategy to accelerate the volume estimation procedure. In addition, we devise some specific techniques for instances that arise from program analysis.

1. Introduction

The satisfiability (SAT) problem in the propositional logic is a fundamental problem in computer science. But in practice, many problems cannot be expressed by propositional formulas directly or naturally. In recent years, there have been a lot of works on solving the Satisfiability Modulo Theories (SMT) problem, which try to decide the satisfiability of logical formulas with respect to combinations of background theories (like reals, integers, arrays, bit-vectors). SMT can be regarded as an extension to SAT, as well as a kind of constraint satisfaction problem (CSP). Quite efficient SMT solvers have been developed, such as CVC3/CVC4, Z3 and Yices [1,10,11].

The counting version of CSP, i.e., #CSP, has been studied by various researchers [5,6]. There has also been much work on the model counting problem in the propositional logic, i.e., counting the number of models of a propositional formula. It is closely related to approximate reasoning [33,8].

On the other hand, the counting version of SMT, i.e., #SMT, has not been studied much. In this paper, we focus on the #SMT problem with respect to the theory of linear arithmetic – #SMT(LA). Given a set of SMT(LA) constraints, we would like to know how many solutions there are. Or, in other words, how large the solution space is. The problem can be regarded

* Corresponding author.
E-mail addresses: gecj@ios.ac.cn (C. Ge), maff@ios.ac.cn (F. Ma), algzhang@sdu.edu.cn (P. Zhang), zj@ios.ac.cn (J. Zhang).

http://dx.doi.org/10.1016/j.tcs.2016.10.019
0304-3975/

as an extension to SMT solving, and also a generalization of both the model counting problem in the propositional logic and volume computation of convex polytopes. It has recently gained some attention in the software engineering community [23,16].

An SMT(LA) formula is satisfiable if and only if there exists a Boolean assignment to its linear inequalities such that the SMT formula is evaluated to true in Boolean level, and the conjunction of inequalities is also consistent. Such Boolean assignment is called feasible assignment. The linear system corresponding to a feasible assignment forms a convex polytope. Ma et al. [32] proposed an exact approach for #SMT(LA) problem which integrates SMT solving with volume computation for convex polytopes. However, exact volume computation in general is a difficult problem. It has been proved to be #P-hard, even for explicitly described polytopes [12,21,22]. Yet, in many applications, it suffices to have an approximate value of the volume of the solution space. Therefore, it is desirable to study highly efficient methods for *estimating* the volume of the solution space.

Volume computation for convex polytopes is a classical problem in mathematics. The high complexity of exact volume computation procedure for convex polytopes is the bottleneck of the approach in [32]. On the other hand, volume estimation methods for convex bodies have been extensively studied in theory. The Monte-Carlo method is a straightforward way to estimate the volume of a convex body. However, it suffers from the curse of dimensionality, which means the possibility of sampling inside a certain space in the target object decreases very quickly while the dimension increases. As a result, the sample size has to grow exponentially to achieve a reasonable estimation. To avoid the curse of dimensionality, Dyer et al. proposed a polynomial time randomized approximation algorithm (called Multiphase Monte-Carlo Algorithm) [13]. At first, the theoretical complexity of this algorithm is $O^*(n^{23})$,[1] it was reduced to $O^*(n^4)$ at last by Lovász, Kannan et al. [28,19,26,31]. Despite the polynomial time results and reduced complexity, there is still a lack of practical implementation.

In this paper, we first describe an algorithm for estimating the volume of convex polytopes which is based on the Multiphase Monte-Carlo method. The algorithm is augmented with a new technique to reutilize random points, so that the number of random points can be significantly reduced. We prove that the reutilization technique has no side-effect on the error. We also investigate a simplified version of hit-and-run method: the coordinate directions method, which has never been employed in volume estimation before. Then we integrate our volume estimation method for convex polytopes into the framework of solving #SMT(LA) problems. We propose a heuristic improvement called two-round strategy, which automatically adjusts the number of random points for each invocation of polytope volume estimation. Besides, for instances arise from program analysis, we also introduce some effective techniques.

The rest of this paper is organized as follows. We first describe some basic concepts and notations, as well as some essential techniques and tools in Section 2. Then Section 3 reviews some related works. In Section 4, we present our volume estimation method for convex polytopes, with theoretical analysis. Section 5 presents our approach to volume computation and estimation for SMT(LA) formulas. In Section 6, we further discuss how to improve our approach for the instances generated from program analysis. Section 7 presents some experimental results. Finally, we conclude in Section 8.

This article is an extension of a conference paper [15] presented at the 9th International Workshop of Frontiers in Algorithmics.

2. Preliminaries

This section describes some basic concepts and notations. We also mention some existing techniques and tools that will be used later.

2.1. SMT(LA) formulas

Definition 1. A **linear arithmetic (LA)** constraint is an expression that may be written in the form $a_1x_1 + a_2x_2 + \cdots + a_nx_n \ op \ a_0$. Here $x_1, x_2, \ldots, x_n$ are numeric variables, $a_0, a_1, a_2, \ldots, a_n$ are constant coefficients, and $op \in \{<, \leq, >, \geq, =, \neq\}$.

Definition 2. An SMT formula ϕ over LA constraints, i.e., an **SMT(LA) formula**, can be represented as a Boolean formula $PS_\phi(b_1, \ldots, b_n)$ together with definitions in the form: $b_i \equiv c_i$. Here c_is are LA constraints. PS_ϕ is the *propositional skeleton* of the formula ϕ.

The propositional skeleton contains logical operators, like AND, OR, NOT. A simple example of SMT(LA) formulas is

$$(x + y < 1 \text{ OR } x \geq y) \text{ AND } (x + y < 1 \text{ OR } x < y \text{ OR } b).$$

Let the Boolean variables b_1 and b_2 represent the linear inequalities $x + y < 1$ and $x < y$ respectively. Then we obtain the propositional skeleton

$$(b_1 \text{ OR } (\text{NOT } b_2)) \text{ AND } (b_1 \text{ OR } b_2 \text{ OR } b).$$

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

Definition 3. An SMT(LA) formula ϕ is satisfiable if there is an assignment α to the Boolean variables in PS_ϕ such that:

1. α propositionally satisfies ϕ, or formally $\alpha \models PS_\phi$;
2. The conjunction of LA constraints under the assignment α is consistent.

The assignment α is called a **feasible assignment**. We denote the set of all feasible assignments of ϕ by $Model(\phi)$.

Let us consider two specific types of numeric variables, the integers and reals.

Definition 4. A **linear integer arithmetic (LIA)** constraint is an LA constraint with integer type variables. Analogously, we define the **linear real arithmetic (LRA)** constraint for real type variables.

Accordingly, there are **SMT(LIA)** formulas and **SMT(LRA)** formulas. For an SMT(LIA) formula, we count the number of solutions. For an SMT(LRA) formula, we compute the volume of the solution space instead.

2.2. Convex polytopes

The assignment of the propositional skeleton of the SMT(LA) formula corresponds to a conjunction of linear constraints which can be regarded as a convex polytope.

Definition 5. A **convex polytope** P is a bounded subset of $\mathbb{R}^d$ which is the intersection of a finite set of half spaces (inequalities).

Formally, it is usually described using the H-representation $\{x|Ax \leq b\}$, where A is a matrix of dimension $m \times d$ and b is a vector of dimension m. a_{ij} represents the element at the i-th row and the j-th column of A, and a_i represents the i-th row vector of A.

There are already some tools available to compute the exact volume of a convex polytope. For example, `Vinci` [4] is such a tool, whose input is a set of linear inequalities. Sometimes we are interested in the number of *integer* points in the solution space for LIA constraints. `LattE` [25] is a tool dedicated to the counting of lattice points inside convex polytopes and the solution of integer programs. But all the parameters in the matrix A and vector b should be integers.

In this paper, we use $vol(K)$ to denote the volume of a body K. For an assignment α of an SMT(LA) formula ϕ, we use $vol(\alpha)$ to denote the volume of the corresponding polytope. The volume of ϕ, denoted by $vol(\phi)$, is formally defined as follows:

$$vol(\phi) = \sum_{\alpha \in Model(\phi)} vol(\alpha).$$

3. Related works

3.1. Volume approximation for convex bodies

Liu et al. [24] developed a tool to estimate the volume of a convex body with a direct Monte-Carlo method. It can also deal with non-convex cases. Suffered from the curse of dimensionality, it can hardly solve high-dimensional problem instances. A more recent work [27] is an implementation of the $O^*(n^4)$ volume algorithm in [31]. This algorithm is designed for convex bodies. However, there are no experimental results except cubes within 10 dimensions, because the oracle describing the convex bodies takes too long to run. Furthermore, it takes hours to approximate the volume of an 8-dimension cube. In Section 7.1.1, we present the experimental results about comparison between our approach and the method used in [27].

3.2. Model counting for SMT formulas

There was little work on the counting of SMT solutions, until quite recently.

Fredrikson and Jha [14] relate a set of privacy and confidentiality verification problems to the so-called *model-counting satisfiability* problem, and present an abstract decision procedure for it. They implemented this procedure for linear-integer arithmetic. Their tool is called `countersat`.

Zhou et al. [36] propose a BDD-based search algorithm which reduces the number of conjunctions. For each conjunction, they propose a Monte-Carlo integration with a ray-based sampling strategy, which approximates the volume. Their tool is named `RVC`. It can handle formulas with up to 18 variables. But the running time is dozens of minutes.

A different approach is described in [9]. It is a bit-level hashing-based model counter. Their approach propositionalizes the solution space and uses XOR-based bit-level hash functions to obtain a randomized subset of the solution space. Then it calls an SMT solver repeatedly to count the subset and estimates the volume of the whole solution space. Note that this approach does not need to modify existing SMT solvers. Their work focuses on the problem of approximate model counting

for a space projected from the solution space of a mixed integer SMT(LA) formula. For continuous problems, though [9] proposed a discretization procedure, it is not so practical, since it introduces too many discrete variables that may be even beyond the limit of Z3's XOR reasoning.

More recently, Chakraborty et al. [7] proposed a hashing-based approximate model counter. It benefits from state-of-the-art word-level SMT solvers. It also approximates the volume of the whole solution space instead of a projection space. For discrete problems without projection, [7] outperforms the previous approximate counter that employs XOR-based hash functions [9], especially, over benchmarks with word-level constraints (e.g., arithmetic constraints). In Section 7.2.3, we present the comparison between our approach and [7].

4. Volume estimation for convex polytopes

In this section, we present our algorithm for estimating the volume of convex polytopes which is based on the Multiphase Monte-Carlo method [13]. We propose two improvements over the original Multiphase Monte-Carlo method. Firstly, we develop a new technique to reutilize random points, so that the number of random points can be significantly reduced. Secondly, we use the coordinate directions hit-and-run method instead of hypersphere directions method. We implemented the new method in a tool called `PolyVest` [15] (Polytope Volume Estimation).

We assume that P is a full-dimensional and nonempty convex polytope. We use $B(x, R)$ to denote the ball with radius R and center x. And 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, i.e., for every non-zero column vector z, the scalar $z^T Az$ is positive.

The basic procedure of `PolyVest` consists of the following three steps: rounding, subdivision and random point generation.

4.1. Rounding

Given a convex polytope Q, the rounding procedure is to find an affine transformation T on Q such that $B(0,1) \subseteq T(Q) \subseteq B(0,r)$, with a constant $\gamma = \frac{vol(Q)}{vol(T(Q))}$. If $r > n$, T can be found by the Shallow-β-Cut Ellipsoid Method [17] (Chapter 3), where $\beta = \frac{1}{r}$. It is an iterative method that generates a series of ellipsoids $\{E_i = E(T_i, o_i)\}$ s.t. $Q \subseteq E_i$, until we find an E_k such that $E(\beta^2 T_k, o_k) \subseteq Q$. Then we transform the ellipsoid E_k into $B(0,r)$. Intuitively, rounding can transform a very "thin" polytope, which cannot be subdivided directly, into a well-bounded one.

This procedure could take much time when r is close to n, e.g. $r = n+1$. There is a tradeoff between rounding procedure and random point generation, since the smaller r is, the more iterations for rounding and the fewer points have to be generated. We used $r = 2n$ in our implementation, so that the overhead of rounding is usually negligible compared to the whole estimation method for polytopes. In the sequel, we use P to represent the new polytope $T(Q)$, and we only consider the polytope P instead of Q.

4.2. Subdivision

Then we divide P into a sequence of convex bodies. The high-level idea of the subdivision step is illustrated in Fig. 1. We place l concentric balls $\{B_i\}$ between $B(0,1)$ and $B(0,r)$. Set $K_i = B_i \cap P$, then $K_0 = B(0,1)$, $K_l = P$ and

$$vol(P) = vol(K_0) \prod_{i=0}^{l-1} \frac{vol(K_{i+1})}{vol(K_i)}.$$

Let α_i denote the ratio $vol(K_{i+1})/vol(K_i)$, then

$$vol(P) = vol(K_0) \prod_{i=0}^{l-1} \alpha_i. \tag{1}$$

Hence the volume of the polytope P is transformed to the product of a series of ratios and the volume of K_0. Note that $K_0 = B(0,1)$, whose volume can be easily computed. So, we only have to estimate the value of α_i.

Of course, one would like to choose the number of concentric balls, l, to be small. However, from Theorem 5, one needs about $O(l^2)$ random points to get a sufficiently good approximation for α_i. It follows that the α_i must not be too large. In `PolyVest`, we set $l = \lceil n \log_2 r \rceil$ and $B_i = B(0, 2^{i/n})$ to construct the convex bodies $\{K_i\}$.

Proposition 1. *If $l = \lceil n \log_2 r \rceil$ and $B_i = B(0, 2^{i/n})$, then $1 \leq \alpha_i \leq 2$.*

Proof. Let r_i denote the radius of ball $B(0, 2^{i/n})$, i.e., $r_i = 2^{i/n}$. 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, since P contains the origin after rounding procedure, K_is also contain the origin. Note that K_is are convex bodies, so

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

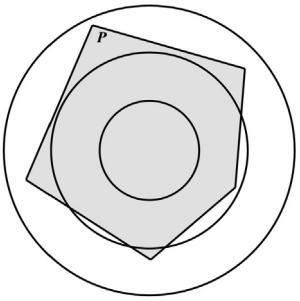

Fig. 1. Multiphase Monte-Carlo.

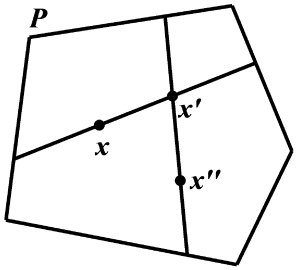

Fig. 2. Hit-and-run.

we have

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

That is, $1 \leq \alpha_i \leq 2$. □

4.3. 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. The value of the parameter *step_size* will be discussed in Section 4.7. Then there is

$$\alpha_i \approx \frac{step\_size}{c_i}.$$

It is easy to generate points in uniform distributions on cubes or ellipsoids but not easy on K_is. So we consider the random walk method. Hit-and-run method is a random walk which has been studied for a long time [34,3,2]. There are two versions: the hypersphere directions method (HDHR), and the coordinate directions method (CDHR). See Fig. 2 for an illustration. HDHR starts from a point x in a convex body K, and generates the next point x' in K by two steps: (i) select a line L through x uniformly on a hypersphere, and (ii) then choose a point x' uniformly on the segment of line L in K. The CDHR is similar to HDHR, but it chooses directions with equal probability from the coordinate direction vectors and their opposites. Berbee et al. [3] proved the following theorems.

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

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

Note that coordinate directions are special cases of directions generated on a hypersphere, hence the previous theoretical research about volume approximation algorithm with hit-and-run methods mainly focuses on HDHR. In this paper, we investigate CDHR and apply it to the volume approximation algorithm. In our algorithm, CDHR 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, $e_1, \ldots, e_n$.
Step 2. Choose a point x' uniformly on the segment of line L in K_{k+1}.

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 $x' = x|_{x_d=x'_d} \in K_{k+1}$.

Since $r_i = 2^{i/n}$ is the radius of B_i and $K_{k+1} = B_{k+1} \cap P$, so $x' \in B_{k+1}$ and $x' \in P$, we have

$$x' \in B_{k+1} \Leftrightarrow |x'| \leq r_{k+1} \Leftrightarrow {x'_d}^2 \leq r_{k+1}^2 - \sum_{i \neq d} x_i^2$$

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

Let

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

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

then interval $[u, v]$ is the range of x'_d that satisfies Formula (2) and Formula (3).

4.4. Reutilization of random 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 generating 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: Random 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 random points generated in previous phases. Suppose that we have already generated a set of points $\mathcal{S}$ by random walk with almost uniform distribution in K_{i+1}, and some of them also hit the convex body K_i, denoted by $\mathcal{S}'$. The ratio α_i is thus estimated with $\frac{|\mathcal{S}|}{|\mathcal{S}'|}$. However, these random points can reveal more information than just the ratio α_i. Since K_i is a sub-region of K_{i+1}, the points in $\mathcal{S}'$ are also almost uniformly distributed in K_i. Therefore, $\mathcal{S}'$ can serve as part of the random points in K_i. Furthermore, for any K_j $(0 \leq j \leq i)$ inside K_{i+1}, the points in K_{i+1} that hit K_j can serve as random points to approximate α_j as well.

Algorithm 1 Volume estimation algorithm with reutilization technique.

```
1: function EstimateVol(step_size)
2:     γ ← Rounding()
3:     x ← Origin
4:     l ← ⌈n log2 r⌉
5:     count, t_0, ..., t_{l-1} ← 0
6:     for i ← l − 1 downto 0 do
7:         for j ← count to step_size do
8:             x ← Walk(x, i, w) /* perform w steps of random walk in K_{i+1}*/
9:             if x ∈ K_i then
10:                calculate a value m such that x ∈ K_m and x ∉ K_{m−1}
11:                t_m ← t_m + 1
12:            end if
13:        end for
14:        count ← Σ_{i'=0}^{i} t_{i'}
15:        α_i ← 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
```

Based on this insight, we devise a different direction, i.e., generate from outside to inside. At the i-th phase which approximates ratio α_i, the algorithm first calculates the number *count* of the former points that are also in K_{i+1}, then generates the rest $(step\_size - count)$ points by random walk. The framework of our volume estimation algorithm with reutilization technique is presented in Algorithm 1. The parameter w is the number of sufficient steps for hit-and-run algorithm mixing. We discuss the value of w in Section 4.5.

Unlike generating random points in natural order, choosing the starting point for each phase in reverse order is a bit complex. The whole generating process in reverse order also starts from the origin. At the end of the i-th phase, we select a point x in K_{i+1} and employ $x' = 2^{-\frac{1}{n}} x$ as the starting point of the next phase (the $(i-1)$-th phase) since $2^{-\frac{1}{n}} x \in K_i$. It is easy to find out that the expected number of saved random points with our algorithm is

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

Since $\alpha_i \leq 2$, we only have to generate less than half random points with this technique. Actually, it can save over 70% time consumption on a large set of benchmarks (see Section 7.1.4). In addition, we shall prove that the reutilization technique has no effect on the error of the estimation result (see Section 4.6).

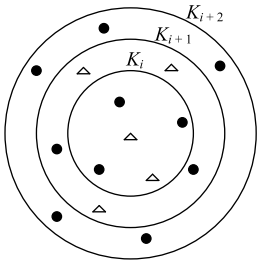

Fig. 3. An illustration of the i-th phase, which aims to estimate $\alpha_i = \frac{k_{i+1}}{k_i}$. The black round points are generated in and before the $(i+1)$-th phase. The white triangle points are generated in the i-th phase.

4.5. About the mixing time

When generating the next point x' with the previous random point x, we have to make some steps from x to achieve stationarity and make x' independent of x. However, the number of sufficient steps w for hit-and-run algorithm mixing is hard to decide. The previous theoretical research [30,29] presented the upper bounds on w in the Markov chain which are of the form:

$w = O(n^2)$ for a random initial point, and

$w = O(n^3)$ for a fixed initial point.

Note that the hiding constant factors in $O(n^2)$ and $O(n^3)$ are 10^{30} and 10^{11} respectively. Lovász et al. [27] reported that the upper bound for HDHR method is much higher than actually required according to the numerical experiences. And they used to set $w = n+1$. They also tried $w = 2(n+1)$ and $w = n\ln n$ without visible improvement. We investigate the value of w for the CDHR method in a similar way, since obtaining the theoretical upper bound is hard. We conducted experiments with linear size of w, i.e., $w = n$, $w = 2n$, and $w = 3n$. Based on the observation in Section 7.1.2, we choose $w = n$.

4.6. Analysis of the reutilization technique

In the following analysis, we assume that our algorithm generates points in uniform distribution. For simplicity, we use k_i to represent $vol(K_i)$ in this section. Let $f(k; n, p)$ represent the probability mass function of the binomial distribution $\mathbb{B}(n, p)$, i.e., $f(k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$.

Recall that in our volume estimation procedure, there are l phases in total. With the reutilization technique, in each phase i $(0 \le i \le l-1)$, to estimate the value of $\alpha_i = \frac{k_{i+1}}{k_i}$, we reuse all points in K_{i+1} generated in the earlier phases, and generate enough new points in K_{i+1}, so that the total number of already existing points and newly generated points in K_{i+1} is equal to *step_size*. See Fig. 3 for an illustration.

We introduce some new notations for the analysis.

For every $0 \le i \le l-1$, define

$G_i = \{$the points newly generated in the i-th phase$\}$

and

$C_i = \{$the points dropped in K_i at the end of the i-th phase$\}$.

Note that $C_i \subseteq K_i$ and $G_i \subseteq K_{i+1}$ for every $0 \le i \le l-1$. Let $c_i = |C_i|$ and $g_i = |G_i|$.

At the end of the i-th phase, Algorithm 1 counts the number of points lying in K_i, and use

$$\frac{\#\text{ of random points in } K_{i+1}}{\#\text{ of random points in } K_i} = \frac{step\_size}{c_i}$$

as the estimation of α_i $(0 \le i \le l-1)$.

Recall that the first phase of Algorithm 1 is the $(l-1)$-th phase. We additionally define $C_l = \emptyset$ (and thus $c_l = 0$) to indicate the fact that at the beginning of the algorithm, there is no random point in K_l.

By definition, we have

Lemma 1. *For every* $0 \le i \le l-1$,

$$c_{i+1} + g_i = step\_size.$$

Lemma 1 says that for every phase i, the sum of the number of random points that are generated in and before the $(i+1)$-th phase (in Fig. 3, these points are the black round points in K_{i+1}), and the number of random points newly generated in the i-th (this) phase (in Fig. 3, these points are the triangle points in K_{i+1}), is equal to $step\_size$.

Furthermore, we define

$$C_{i,j} = C_i \cap K_j, \quad 0 \le i \le l-1, 0 \le j \le i,$$
$$G_{i,j} = G_i \cap K_j, \quad 0 \le i \le l-1, 0 \le j \le i+1.$$

By definition, we have $C_{i,i} = C_i$ and $G_{i,i+1} = G_i$.

Let $c_{i,j} = |C_{i,j}|$ and $g_{i,j} = |G_{i,j}|$.

Lemma 2. *For each $i = 0, \ldots, l-1$, we have*

1. $C_i = C_{i+1,i} \cup G_{i,i}$, *and*
2. $c_i = c_{i+1,i} + g_{i,i}$.

Proof. By definition, C_i is the set of random points that lie in K_i at the end of the i-th phase of Algorithm 1. These points consist of two parts as illustrated in Fig. 3. One part is the set of random points that lie in K_i and are generated *before* the i-th phase. This set can be denoted by $C_{i+1} \cap K_i = C_{i+1,i}$ using our notation. In Fig. 3, $C_{i+1,i}$ is the set of black round points in K_i. The other part is the set of random points that lie in K_i and are newly generated *in* the i-th phase. This set can be denoted by $G_i \cap K_i = G_{i,i}$ using our notation. In Fig. 3, $G_{i,i}$ is the set of triangle points in K_i.

The above analysis shows that

$$C_i = (C_{i+1} \cap K_i) \cup (G_i \cap K_i) = C_{i+1,i} \cup G_{i,i}.$$

Since $C_{i+1,i}$ and $G_{i,i}$ are disjoint, we naturally have $c_i = c_{i+1,i} + g_{i,i}$. The lemma follows. □

Lemma 3. *$c_{i+1,i}$ and $g_{i,i}$ are conditionally independent given c_{i+1} or g_i ($i = 0, \ldots, l-1$).*

Proof. Lemma 1 indicates that the value of g_i is determined by c_{i+1}, and vice versa. So, we only need to show that $c_{i+1,i}$ and $g_{i,i}$ are conditionally independent given one of c_{i+1} and g_i, say, g_i.

By definition, $c_{i+1,1} = |C_{i+1} \cap K_i|$, and $g_{i,i} = |G_i \cap K_i|$. See Fig. 3 for example. In Fig. 3, $c_{i+1,i}$ is the number of black round points in K_i, and $g_{i,i}$ is the number of triangle points in K_i. Although the total number of random points in K_{i+1} is equal to $step\_size$ (which contains both $c_{i+1,i}$ and $g_{i,i}$), given the value $g_i = |G_i|$, the value $g_{i,i}$ is only related to the shape of K_i, since all the points in G_i are distributed uniformly at random in K_{i+1}, which is a superset of K_i. Therefore, $c_{i+1,i}$ and $g_{i,i}$ are conditionally independent given the value g_i. □

Lemma 3 is an important observation of our algorithm with reutilization technique. Then, with this observation, we introduce the following theorem which concerns the correctness of our algorithm.

Theorem 3. *For each $i = 0, \ldots, l-1$, $c_i \sim \mathbb{B}(step\_size, \frac{k_i}{k_{i+1}})$.*

Proof. For simplicity, let

$$p = \frac{k_i}{k_{i+1}}$$

and

$$s = step\_size.$$

We consider the probability $\Pr(c_i = x)$ for arbitrary $0 \le x \le s$. We have

$$\Pr(c_i = x) = \sum_{y=0}^{s} \Pr(c_i = x, c_{i+1} = y) = \sum_{y=0}^{s} \Pr(c_i = x \mid c_{i+1} = y) \Pr(c_{i+1} = y). \tag{5}$$

The conditional probability $\Pr(c_i = x \mid c_{i+1} = y)$ can be calculated as

$$\Pr(c_i = x \mid c_{i+1} = y)$$
$$\underset{\text{LM 2}}{=} \Pr(c_{i+1,i} + g_{i,i} = x \mid c_{i+1} = y)$$

$$= \sum_{a=0}^{x} \Pr(c_{i+1,i} = a, g_{i,i} = x - a \mid c_{i+1} = y)$$

$$\underset{\text{LM 3}}{=} \sum_{a=0}^{x} [\Pr(c_{i+1,i} = a \mid c_{i+1} = y) \cdot \Pr(g_{i,i} = x - a \mid g_i = s - y)]. \tag{6}$$

Since the points in C_{i+1} are generated uniformly at random in K_{i+1}, and $C_{i+1,i} = C_{i+1} \cap K_i$ (that is, $C_{i+1,i}$ is the set of points in C_{i+1} that are dropped in K_i), we have

$$\Pr(c_{i+1,i} = a \mid c_{i+1} = y) = f(a; y, \frac{k_i}{k_{i+1}}) = f(a; y, p). \tag{7}$$

Similarly, since the points in G_i are generated uniformly at random in K_{i+1}, and $G_{i,i} = G_i \cap K_i$, we have

$$\Pr(g_{i,i} = x - a \mid g_i = s - y) = f(x - a; s - y, p). \tag{8}$$

Combining Equations (6), (7) and (8), we have

$$\begin{aligned}
\Pr(c_i = x \mid c_{i+1} = y) &= \sum_{a=0}^{x} f(a; y, p) f(x - a; s - y, p) \\
&= \sum_{a=0}^{x} \binom{y}{a} \binom{s-y}{x-a} p^x (1-p)^{s-x} \\
&= p^x (1-p)^{s-x} \sum_{a=0}^{x} \binom{y}{a} \binom{s-y}{x-a} \\
&= p^x (1-p)^{s-x} \binom{s}{x} \\
&= f(x; s, p).
\end{aligned} \tag{9}$$

Finally, combining Equations (5) and (9), we have

$$\Pr(c_i = x) = \sum_{y=0}^{s} \Pr(c_{i+1} = y) f(x; s, p) = f(x; s, p). \quad \square \tag{10}$$

The proof of Theorem 3 also shows that c_i and c_{i+1} are independent (see Equations (9) and (10)). Actually, we have stronger statement, which says that $c_0, \ldots, c_{l-1}$ are mutually independent.

Theorem 4. *The random variables $c_0, \ldots, c_{l-1}$ are mutually independent.*

Proof. We prove the theorem by induction on the indices of c_i's, from $l-2$ down to 0.

Basic Step: When $i = l - 2$, we have already proved that c_{l-2} and c_{l-1} are independent in the proof of Theorem 3 (see Equations (9) and (10)).

Induction Hypothesis: Suppose that $c_{i+1}, c_{i+2}, \ldots, c_{l-1}$ are mutually independent, i.e.,

$$\Pr(c_{i+1} = x_{i+1}, \ldots, c_{l-1} = x_{l-1}) = \prod_{j=i+1}^{l-1} \Pr(c_j = x_j). \tag{11}$$

Induction Step: In the following we show that $c_i, c_{i+1}, \ldots, c_{l-1}$ are mutually independent. For arbitrary $0 \le x_i, \ldots, x_{l-1} \le$ *step_size*, we consider the probability

$$\begin{aligned}
&\Pr(c_i = x_i, c_{i+1} = x_{i+1}, \ldots, c_{l-1} = x_{l-1}) \\
&= \Pr(c_i = x_i \mid c_{i+1} = x_{i+1}, \ldots, c_{l-1} = x_{l-1}) \cdot \Pr(c_{i+1} = x_{i+1}, \ldots, c_{l-1} = x_{l-1}).
\end{aligned} \tag{12}$$

By definition, the set C_i consists of points in C_{i+1} and G_i which also lie in K_i. Though the set C_{i+1} consists of points generated in and before the $(i+1)$-phase (i.e., $C_{i+1} = \bigcup_{j=i+1}^{l-1} (G_j \cap K_{i+1})$), the points in C_{i+1} are still distributed uniformly in K_{i+1}. In addition, although g_i is determined by c_{i+1}, the points in G_i are generated independently with the points in C_j's ($j \ge i+1$). Therefore, c_i is only affected by the shape of K_i for any value of c_{i+1}, since in the i-th phase, Algorithm 1 must generate *step_size* uniformly distributed random points in K_{i+1}. In other words, the procedure to count c_i in the i-th phase

of Algorithm 1 is equivalent to generating *step_size* random points in K_{i+1} uniformly at random from scratch and then counting the number of random points dropped in K_i. Furthermore, when generating random points in K_{i+1}, Algorithm 1 does not check or count the points in $C_{i+2}, \ldots, C_{l-1}$. So, we have

$$\begin{aligned}\Pr(c_i = x_i \mid c_{i+1} = x_{i+1}, \ldots, c_{l-1} = x_{l-1}) &= \Pr(c_i = x_i \mid c_{i+1} = x_{i+1}) \\ &= \Pr(c_i = x_i). \end{aligned} \tag{13}$$

Finally, combining Equations (11), (12) and (13), we have

$$\Pr(c_i = x_i, c_{i+1} = x_{i+1}, \ldots, c_{l-1} = x_{l-1}) = \prod_{j=i}^{l-1} \Pr(c_j = x_j). \quad \square$$

4.7. Analysis on the number of random points

In this section, we provide some analysis about the number of random points for estimating each α_i.

Theorem 5. *Given $\epsilon > 0$, $\delta \in (0, 1)$, if we let $step\_size = \left\lceil (\frac{z_{1-\delta/2} \cdot l}{\ln(1+\epsilon)} + z)^2 \right\rceil$, then the estimation result of Algorithm 1 lies in $[(1+\epsilon)^{-1} vol(P), (1+\epsilon) vol(P)]$ with probability at least $1-\delta$, where $z_{1-\delta/2}$ is the $1-\delta/2$ quantile of a standard normal distribution.*

Proof. For simplicity, let

$$s = step\_size$$

and

$$z = z_{1-\delta/2}.$$

We use v to represent the output estimation of Algorithm 1. Let p_i represent $\frac{c_i}{s}$. From Theorem 3, the value of p_i is the proportion of successes in a Bernoulli trail process which follows binomial distribution $\mathbb{B}(s, \frac{vol(K_{i+1})}{vol(K_{i+2})})$. So, we apply the approximate formula of a binomial proportion confidence interval $p_i \pm z\sqrt{\frac{p_i(1-p_i)}{s}}$, i.e.,

$$\Pr\left(p_i - z\sqrt{\frac{p_i(1-p_i)}{s}} \le \frac{vol(K_{i+1})}{vol(K_{i+2})} \le p_i + z\sqrt{\frac{p_i(1-p_i)}{s}} \right) \ge 1 - \delta.$$

Recall that Algorithm 1 uses $\frac{s}{c_i}$ to estimate α_i. There is $v = vol(K_0) \prod_{i=0}^{l-1} \frac{1}{p_i}$. Then the $1-\delta$ confidence interval of v shall be

$$\left[\frac{vol(K_0)}{\prod_{i=0}^{l-1} \left(p_i + z\sqrt{\frac{p_i(1-p_i)}{s}} \right)}, \frac{vol(K_0)}{\prod_{i=0}^{l-1} \left(p_i - z\sqrt{\frac{p_i(1-p_i)}{s}} \right)} \right].$$

To prove this theorem, it suffices to prove exact volume $vol(P)$ lies in interval $[(1+\epsilon)^{-1} v, (1+\epsilon) v]$ with probability at least $1-\delta$. Therefore, we only have to prove the following two inequalities,

$$\frac{vol(K_0)}{\prod_{i=0}^{l-1} \left(p_i + z\sqrt{\frac{p_i(1-p_i)}{s}} \right)} \ge (1+\epsilon)^{-1} v, \tag{14}$$

$$\frac{vol(K_0)}{\prod_{i=0}^{l-1} \left(p_i - z\sqrt{\frac{p_i(1-p_i)}{s}} \right)} \le (1+\epsilon) v. \tag{15}$$

Consider Equation (14), it is equivalent to

$$\prod_{i=0}^{l-1} \left(1 + z\sqrt{\frac{(1-p_i)}{s \cdot p_i}} \right) \le 1 + \epsilon.$$

Since Proposition 1 indicates $\frac{1}{2} \le p_i \le 1$, it is easy to see that $(1-p_i)/p_i \le 1$. That is, we only have to prove

$$\left(1+\frac{z}{\sqrt{s}}\right)^{l} \leq 1+\epsilon. \tag{16}$$

Note that for arbitrary constant β, $(1+\beta/l)^l$ is monotonically increasing with respect to l, and $\lim_{l\to\infty}(1+\beta/l)^l = e^{\beta}$, where e is the base of the natural logarithm. So, $s = (\frac{z\cdot l}{\ln(1+\epsilon)} + z)^2 \geq (\frac{z\cdot l}{\ln(1+\epsilon)})^2$ guarantees Equation (16).

In a similar way, we prove Equation (15). It is equivalent to

$$\prod_{i=0}^{l-1} \frac{1}{1-z\sqrt{\frac{(1-p_i)}{s\cdot p_i}}} \leq 1+\epsilon. \tag{17}$$

Consider the left-hand-side of Equation (17), there is

$$LHS \leq \left(\frac{1}{1-\frac{z}{\sqrt{s}}}\right)^{l} = \left(1+\frac{z}{\sqrt{s}-z}\right)^{l}.$$

Note that $(1+\frac{\ln(1+\epsilon)}{l})^l \leq 1+\epsilon$. So, it is easy to see that $s \geq (\frac{z\cdot l}{\ln(1+\epsilon)} + z)^2$ guarantees $(1+\frac{z}{\sqrt{s}-z})^l \leq 1+\epsilon$. □

5. Integrating polytope volume estimation into #SMT(LA) solving

In this section, we first review the exact volume computation approach for SMT(LA) formulas in [32], then describe how to extend it to volume estimation of SMT(LA) Formulas. We also propose a two-round strategy to accelerate the volume estimation procedure.

5.1. From computation to estimation for #SMT(LRA) problems

Given an SMT(LA) formula, the sum of volumes of all feasible assignments is the volume of the whole formula. Ma et al. [32] presented an exact approach to solving #SMT(LA) problem which integrates SMT solving with volume computation for convex polytopes. The basic idea is to enumerate feasible assignments by solving the SMT(LA) formula and accumulate the volumes of these assignments. Polytope volume computation serves as a subroutine which produces the volume of each feasible assignment. To reduce the number of calls of polytope volume computation, we also proposed a strategy that combines the feasible assignments into "bunches". Each time a feasible assignment is obtained, we search the neighborhood of this assignment by negating its literals. We can combine the original assignment with one of its feasible neighbor assignments. Then we obtain a partial assignment that still propositionally satisfies the formula. The resulting assignment may cover a bunch of feasible assignments, hence is called a "bunch". For example, given a feasible assignment $\{b_1, \neg b_2, \neg b_3, b_4\}$ of formula $PS_\phi(b_1, b_2, b_3, b_4)$, we search its neighborhood. Assume $\{b_1, \neg b_2, \neg b_3, \neg b_4\}$ is also feasible, then we could obtain a partial feasible assignment $\{b_1, \neg b_2, \neg b_3\}$ such that $vol(\{b_1, \neg b_2, \neg b_3\}) = vol(\{b_1, \neg b_2, \neg b_3, b_4\}) + vol(\{b_1, \neg b_2, \neg b_3, \neg b_4\})$. And the volume computation subroutine is called for the polytope corresponding to each bunch rather than each feasible assignment, so that the number of calls is reduced.

Although the number of calls of polytope volume computation is considerably reduced by the "bunch" strategy, polytope volume computation is still the bottleneck of our previous approach because of its high complexity. To overcome this obstacle, we can substitute the polytope volume computation subroutine with the volume estimation method in Section 4, thereby generalize our previous approach to estimate the volume of the solution space of SMT(LRA) formulas. The basic procedure is quite similar to that of volume computation as described in [32]. Each time we obtain a bunch of feasible (partial) assignments, we call `PolyVest` to estimate the volume of the polytope corresponding to this bunch. The sum of the estimated volumes of all bunches is approximately the volume for the whole formula.

5.2. Two-round strategy

In the Multiphase Monte-Carlo method, the number of random points at each phase, i.e., *step_size*, is a key parameter. To control parameter *step_size* easily, we introduce a weight S for *step_size* so that the weighted version is $step\_size = S \cdot \left\lceil \frac{z_{1-\delta/2}\cdot l^2}{\ln(1+\epsilon)} \right\rceil$.

As the number of random points increases, the accuracy of estimation improves, and the estimation process also takes more time. It is important to balance the accuracy and the running time since the estimation subroutine is usually called many times. Therefore, we employ a **two-round strategy** that can dynamically determine a proper weight for each feasible (partial) assignment. At the first round of estimation, each feasible assignment is generated with a fixed small weight to get a quick and rough estimation. Since the volumes of feasible assignments may vary a lot, intuitively a feasible assignment with relatively larger volume should be estimated with higher accuracy. Hence at the second round, the weight for each assignment is determined according to its estimated volume from the first round. More specifically, we use the following rule to decide the weights in the second round:

- Suppose the fixed small weight in the first round is S_{min}, and the largest weight in the second round is set to S_{max}. Let V_{max} denote the largest estimated volume in the first round, and V_i denote the volume of the ith feasible assignment estimated in the first round. Then the weight S_i for the ith feasible assignment in the second round is:

$$S_i = \frac{2 \times S_{max} \times V_i}{V_{max}}.$$

If $S_i \leq S_{min}$, the ith feasible assignment is neglected at the second round, and we use the result from the first round as its estimated volume. If $S_i > S_{max}$, then set S_i to S_{max}.

We choose $S_{min} = 0.01$ and $S_{max} = 1$ in practice. It usually saves more than 95% points for random instances. The experimental results and further discussions are presented at Section 7.2.1.

6. Handling practical instances from program analysis

SMT solvers are the core engine of many tools for program analysis, testing, and verification. These tools may generate a large number of SMT(LA) formulas. It is important to improve the efficiency of our approach in these scenarios, especially when handling large instances.

6.1. The difficulties

There almost always exist integer variables in the SMT(LA) formulas generated from program analysis. For such formulas, lattice counting with `LattE` can only handle instances with about 10 variables within a reasonable amount of time. However, even when analyzing just one function in a program, we might obtain problem instances with dozens of integer variables. Yet it is risky to use volume estimation (as described in the previous section) to approximate the number of lattice points, since there is no bound of the relative error of such an approximation. For example, volume estimation or computation will return zero directly if it encounters an equality constraint. But there may be many lattice points for such cases. In the following we present divide-and-conquer methods to deal with some large formulas.

6.2. Several observations

We have made several observations on instances generated from program analysis.

- Usually there are just a few variables in each linear inequality.
- There exist groups of independent variables in the set of linear inequalities. So variables in different groups do not appear in the same inequality.
- In particular, there may exist some linear inequalities with only one variable.
- Due to the bunch technique, it is common that the inequalities in a bunch only contain part of variables.

The following SMT(LA) formula is generated from the analysis of a space management program:

```
(a > 0) AND (t >= a) AND (t <= a + 16) AND
(NOT (b < c)) AND (NOT (d + 0 >= r)) AND
((e = 0) OR (e = 3) OR (e = 5) OR (e = 10) OR (e = 15)) AND
((y > p) OR (z > p) OR (x > q)) AND (f = 1)
```

It contains 13 variables and 14 linear inequalities. There are at most two variables in each inequality (our first observation). Then we consider one of the bunches:

```
(a > 0) AND (t >= a) AND (t <= a + 16) AND (b >= c) AND
(d + 0 < r) AND (e != 0) AND (e != 3) AND (e != 5) AND
(e != 10) AND (e = 15) AND (y <= p) AND (z > p) AND (f = 1)
```

There are 11 variables and 13 linear inequalities in this bunch (assume `x > q` is reduced by bunch techniques).
We can manually subdivide this bunch into 6 mutually independent groups of variables, as well as the inequalities:

```
G1. (a > 0) AND (t >= a) AND (t <= a + 16)
G2. (b >= c)
G3. (d + 0 < r)
G4. (e != 0) AND (e != 3) AND (e != 5) AND (e != 10) AND (e = 15)
G5. (y <= p) AND (z > p)
G6. (f = 1)
```

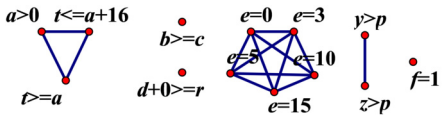

Fig. 4. IRG of the Example in Section 6.2.

There are at most three different variables in each group. Specifically, there are only single-variable inequalities in group 4 and 6. We can compute their solution space easily. And the number of lattice points equals to the multiplication of the number of lattice points of each group. So we reduce this bunch into several one to two dimensional problems which are much easier to solve.

6.3. Reduction and division for linear inequalities

Based on the aforementioned observations, we propose several preprocessing techniques to reduce and divide the set of inequalities and variables. In our implementation, we use $m \times n$ matrix A and vector b to represent the set of linear inequalities $Ax \leq b$, where m is the number of inequalities and n is the number of variables. For convenience, we use $vol(Ax \leq b)$ to denote the volume of solution space of constraints $Ax \leq b$.

6.3.1. Variable reduction

We call a variable **active** if it appears in at least one inequality in $Ax \leq b$. Due to the bunch technique, some variables are not always active in the inequalities of the bunch. So it is necessary to identify active variables and reduce the number of inactive variables. We observe that such reduction procedure is quite useful. This will be discussed in the evaluation section. In the example of Section 6.2, the bunch eliminates the inequality `(x > q)` and variables x and q. So we could shrink the matrix A by deleting columns corresponding to x and q.

6.3.2. Graph-based division

We introduce the **Inequality Relation Graph (IRG)** to divide a bunch into mutually independent groups of inequalities. It is constructed by two rules: (i) map each inequality into a vertex $v \in V$, (ii) add an edge $e = (u, v)$ into E if and only if there exists a variable in both inequalities represented by u and v. The time complexity of the direct construction for IRG is $O(m^2n)$. Because there is no edge between vertices in different strongly connected components (SCC), the inequalities represented by these vertices are also independent. So each SCC in the IRG can represent a group. To obtain SCCs, one can use Tarjan's algorithm [35] which is linear time. The overhead of this division procedure is negligible compared to other parts of our approach for #SMT solving. Note that different groups don't share inequalities or variables, so we have the following proposition.

Proposition 2. *$vol(Ax \leq b) = \prod_i vol(A_i x \leq b)$, where $A_i x \leq b$ represents the ith SCC.*

Consider the example in Section 6.2. The IRG of the bunch is illustrated in Fig. 4. There are 6 SCCs in the graph which correspond to the groups listed in Section 6.2.

6.3.3. Special case of one dimension

`LattE` is based on the Barvinok's algorithm which is very sophisticated. For quite simple circumstances, such as one-dimensional problems, there is not any special treatment for them in `LattE`. The initialization takes so much time that division procedure becomes almost useless on small problems. And so does `Vinci`. Therefore, we handle one-dimensional problems directly and do not call `LattE` or `Vinci`. For a one-dimensional problem, it is an interval. So we only have to calculate the upper and lower bounds by checking the corresponding linear constraints.

7. Experimental results

In this section, we first present the evaluation of `PolyVest`[2] in section 7.1. We then present the results of our approach for #SMT(LA) problems in section 7.2. By default, we used $\epsilon = 0.45$, $\delta = 0.1$, and a timeout of 1 hour. Every experiment was conducted on a workstation with 3.40 GHz Intel Core i7-2600 CPU and 8 GB memory. In the following tables, "—" means that the instance takes more than one hour to solve (or the tool runs out of memory).

[2] The tool and benchmarks are available at http://lcs.ios.ac.cn/~zj/polyvest.html.

Table 1
Comparison between PolyVest and LVM.

Instance	n	v	LVM					PolyVest		
			Time (s)	Result	Err. e	e/v	a_0	ϵ	Time (s)	Result
cube_2	2	4	643.4	4.013	0.051	0.013	6n	0.013	0.134	4.006
cube_3	3	8	1008	8.109	0.279	0.035	6n	0.035	0.128	8.071
cube_4	4	16	1419	15.48	0.452	0.028	6n	0.028	0.984	16.11
cube_5	5	32	1910	31.70	3.250	0.102	6n	0.102	0.291	32.43
cube_6	6	64	2583	62.34	2.8	0.044	2n	0.044	3.913	64.42
cube_7	7	128	3210	128.7	11.6	0.091	2n	0.091	2.442	129.2
cube_8	8	256	–	–	–	–	2n	0.45	0.413	245.8

7.1. Evaluation of PolyVest

In this subsection, we first compare our approach with the Lovasz–Vempala method [27] and Vinci. Next we present evaluations of the accuracy of our approach. Then we discuss the size of w. After that we compare CDHR and HDHR methods. Finally, we show the effectiveness of reutilization technique. Our test cases include:

- "cube_n": Hypercubes with side length 2. The volume of "cube_n" is 2^n.
- "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 product of permutation matrices $\{P_i\}$ that put rows and columns of Q in random orders. This mapping preserves the volume.
- "rh_n_m": An n-dimensional polytope constructed by randomly choosing m hyperplanes tangent to sphere.
- "cuboid_n(S)": Scaling "cube_n" by 100 in one dimension, and then apply random shear mapping to it once. We use this instance to approximate a "thin stick" which is not parallel to any axis.

7.1.1. The performance of PolyVest

Table 1 presents the result of comparing the performance of PolyVest vis-a-vis Lovasz–Vempala Method (LVM), which is also a volume estimation algorithm based on the Multiphase Monte-Carlo method. We only conducted experiments on cubes as the implementation of LVM presented in [27] could not handle instances other than cubes. In Table 1, column 1 gives the instance name, column 2 gives the number of dimensions n, and column 3 gives the exact volume v. The running times and estimating results for LVM are presented in columns 4 and 5. Column 6 gives the total error e which is estimated by LVM. It indicates that the estimating result lies in the interval $[v-e, v+e]$ with high probability.[3] Column 7 gives the ratio e/v which is the relative error estimated by LVM. Column 8 gives the settings of parameter a_0 of LVM. In [27], the authors used $a_0 = 6n$ for *cube*_2 and *cube*_5, and $a_0 = 2n$ for *cube*_8. Column 9 gives the parameter ϵ of our approach. The running times and estimating results for PolyVest are presented in columns 10 and 11. In these experiments, we specify the value of ϵ exactly the same as e/v. It means that LVM estimates with similar size of error to PolyVest. The results show that our approach is significantly faster LVM. For the 8-dimensional cube, LVM could not solve in one hour. We could not obtain the value of e/v as well. So, we used the value of parameter ϵ by default.

Table 2 presents the result of comparing the performance of PolyVest vis-a-vis Vinci. Vinci is a well-known package which implements the state-of-the-art algorithms for exact volume computation of convex polytopes. It consisted of several methods. In Table 2, T_{rlass}, T_{hot} and T_{lawnd} represent the running times of three methods in Vinci respectively. The "rlass" uses Lasserre's method, which needs input of H-representation. The "hot" uses a Cohen&Hikey-like face enumeration scheme, which needs input of V-representation. The "lawnd" uses Lawrence's formula, which is the fastest method in Vinci and both descriptions are needed. In Table 2, the running times of Vinci do not contain transformation from H-representation to V-representation. 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". Given 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 it is the dual problem of constructing the convex hull by the vertices. This process is both time-consuming and space-consuming. On the other hand, the running times of PolyVest appear to be more 'stable'.

Recall that there are $O(l)$ phases, $O(l^2)$ random points in each phase, and $O(w)$ steps for one random point. Since $l = n\log_2 2n$ and $w = n$, our algorithm generates $O(n^4(\log_2 2n)^3)$ steps of random walk. Note that $10^4(\log_2 20)^3 : 20^4(\log_2 40)^3 : 30^4(\log_2 60)^3 \approx 1 : 30 : 207$. Consider the running time growth of a walk with respect to n (for details, see Table 5), the overall running times of instances "cube_10", "cube_20" and "cube_30" accord with the complexity. Besides, there are two factors which are also related to the running time: (i) the reutilizing ratio and (ii) the shape of the polytope after rounding procedure. These lead to the differences of running times for instances with the same scale, e.g., "cube_10" and "cube_10(S)". It is difficult to predict these factors. However, our analysis covers the worst cases.

[3] Note that the authors of [27] did not specify the probability, but only reported it with "high" probability.

Table 2
Comparison between `PolyVest` and `Vinci`.

Instance	n	m	`PolyVest`		`Vinci`			
			Result	Time (s)	Result	T_{rlass} (s)	T_{hot} (s)	T_{lawnd} (s)
cube_10	10	20	1037.9	2.063	1024	0.004	0.044	0.008
cube_10(S)	10	20	990.5	0.939	1023.86	0.008	0.124	0.024
cube_15	15	30	32729.3	22.583	3.28e+4	0.300	212.8	0.156
cube_20	20	40	1.04e+6	126.1	1.05e+6	–	–	8.085
cube_30	30	60	1.08e+9	1672.6	–	–	###	###
rh_8_25	8	25	815.5	0.460	785.989	0.864	0.160	0.016
rh_10_20	10	20	14002	1.327	13882.7	0.284	0.340	0.012
rh_10_25	10	25	5705.48	1.434	5729.52	5.100	1.932	0.072
rh_10_30	10	30	2016.57	1.420	2015.58	660.4[a]	5.772	0.144
rh_8_25(S)	8	25	796.329	0.452	785.984	1.268	0.156	0.032
rh_10_20(S)	10	20	14062.9	1.278	13883.8	0.832	0.284	0.032
rh_10_25(S)	10	25	5507.95	1.443	5729.18	11.949	1.960	0.104
rh_10_30(S)	10	30	2043.33	1.489	2015.87	1251.1[a]	6.356	0.248

###: We did not test "cube_30" by "hot" and "lawnd", because there are too many vertices in these polytopes.
[a] Enable the `Vinci` option to restrict memory storage, so as to avoid running out of memory.

Table 3
More statistical results of `PolyVest`.

Instance	v	$[1.45^{-1}v, 1.45v]$	$[v_{min}, v_{max}]$	Avg.	Std Dev.
cube_5	32	[22.07, 46.4]	[28.99, 33.86]	31.93	1.05
cube_10	1024	[953.6, 1485]	[953.6, 1089]	1019	25.98
cube_10(S)	1024	[953.6, 1485]	[980.4, 1064]	1023	15.20
cube_15	32768	[22598, 47513]	[31169, 34508]	32866	588.5
cube_20	1.05e+6	[7.23e+5, 1.52e+6]	[1.00e+6, 1.09e+6]	1.05e+6	16889
cube_20(S)	1.05e+6	[7.23e+5, 1.52e+6]	[1.02e+6, 1.08e+6]	1.05e+6	10739
cuboid_10(S)	1.02e+5	[7.06e+4, 1.48e+5]	[9.83e+4, 1.07e+5]	1.02e+5	1626
cuboid_20(S)	1.05e+8	[7.23e+7, 1.52e+8]	[1.02e+8, 1.07e+8]	1.05e+8	1.14e+6
rh_8_25	786.0	[542.1, 1139.7]	[751.0, 822.9]	784.8	16.78
rh_10_20	13883	[9574.3, 20130]	[13220, 14535]	13817	275.0
rh_10_25	5730	[3951, 8308]	[5422, 5980]	5714	109.8
rh_10_30	2016	[1390, 2922]	[1937, 2104]	2016	32.19

Table 4
Comparison about different sizes of w.

Instance	$w = 1$		$w = n$		$w = 2n$		$w = 3n$	
	Avg.	Std Dev.	Avg.	Std Dev.	Avg.	Std Dev.	Avg.	Std Dev.
cube_2	3.96	0.258	3.97	0.205	4.01	0.166	4.01	0.163
cube_5	32.07	2.51	32.08	1.28	32.18	1.09	32.05	1.10
cube_10	1027	63.03	1025	28.97	1022	23.36	1024	24.85
cube_15	32658	1520	32809	680.9	32685	605.2	32800	643.6
cube_20	1.05e+6	42703	1.05e+6	17080	1.05e+6	18516	1.05e+6	17108
rh_8_25	785.1	39.60	785.5	17.66	789.3	17.52	785.8	17.23
rh_10_20	13790	676.1	13882	298.4	13849	271.8	13881	247.6
rh_10_25	5724	272.3	5734	99.07	5731	92.26	5729	95.59
rh_10_30	2030	82.29	2015	37.73	2013	37.20	2017	34.60
rh_20_40	107.9	4.34	108.4	1.38	108.2	1.39	108.0	1.48

We did more tests on our approach to see how accurate it is. We executed `PolyVest` 100 times for each instance. In Table 3, column 1 gives the instance name, column 2 gives the exact volume v, column 3 gives the interval $[1.45^{-1}v, 1.45v]$, column 4 gives the minimum value v_{min} and maximum value v_{max} over 100 times of experiment, column 5 and column 6 give the average values and standard deviations respectively. Since we used $\epsilon = 0.45$ and $\delta = 0.1$, from Theorem 5, the estimating result should lie in interval $[1.45^{-1}v, 1.45v]$ with probability at least 90%. Table 3 shows that $[v_{min}, v_{max}] \subset [1.45^{-1}v, 1.45v]$ for each instance. In other words, it means that the frequency on interval $[1.45^{-1}v, 1.45v]$ is 100 over 100 times of experiments, which follows our analysis. In addition, we observe that the interval $[v_{min}, v_{max}]$ is significantly smaller than $[1.45^{-1}v, 1.45v]$. Actually, there is $[v_{min}, v_{max}] \subset [1.1^{-1}v, 1.1v]$ for most instances.

7.1.2. The experiments on mixing time

To achieve a proper mixing time, we experimented different sizes of w: $w = 1$, $w = n$, $w = 2n$ and $w = 3n$. We executed `PolyVest` 100 times for each instance. In Table 4, "Avg." and "Std Dev." represent the average values and the standard deviations respectively.

Table 5
Comparison about speed between CDHR and HDHR.

n	m	CDHR (s)	HDHR (s)
10	20	3.572	13.761
20	40	7.095	24.502
30	60	13.85	40.455
40	80	22.13	61.484

Table 6
Comparison about accuracy between CDHR and HDHR.

Instance	Exact.	CDHR			HDHR		
		Avg.	Std Dev.	ERR.	Avg.	Std Dev.	ERR.
cube_5	32	31.93	1.05	3.28%	31.88	1.51	4.73%
cube_10	1024	1019	25.98	2.55%	1032	32.04	3.10%
cube_15	3.28e+4	32866	588.5	1.79%	32973	766.4	2.32%
cube_20	1.05e+6	1.05e+6	16889	1.61%	1.05e+6	22240	2.12%
cuboid_10(S)	1.02e+5	1.02e+5	1626	1.59%	1.03e+5	2161	2.10%
cuboid_20(S)	1.05e+8	1.05e+8	1.14e+6	1.02%	1.05e+8	1.27e+6	1.21%
rh_8_25	785.99	784.8	16.78	2.14%	786.4	26.06	3.31%
rh_10_20	13883	13817	275.0	1.99%	14051	378.5	2.69%
rh_10_30	2016	2016	32.19	1.60%	2006	61.04	3.04%

Table 7
Effectiveness of reutilizing random points.

Instance	Without reusing			With reusing			$\overline{n_2}/\overline{n_1}$
	$\overline{n_1}$	Avg.	Std Dev.	$\overline{n_2}$	Avg.	Std Dev.	
cube_5	95324	32.18	1.19	33617	31.93	1.05	35.27%
cube_10	1.57e+6	1021	24.99	5.83e+5	1019	25.98	37.08%
cube_15	7.48e+6	32806	625.8	2.87e+6	32886	588.5	38.41%
cube_20	2.24e+7	1.05e+6	19388	8.84e+6	1.05e+6	16889	39.39%
cuboid_10	8.07e+5	1.03e+5	1488	2.52e+5	1.02e+5	1626	31.19%
cuboid_20	1.06e+7	1.05e+8	1.23e+6	3.66e+6	1.05e+8	1.14e+6	34.87%
rh_8_25	4.79e+5	786.0	16.90	1.43e+5	784.8	16.78	29.74%
rh_10_20	1.22e+6	13876	280.9	3.44e+5	13817	275.0	28.19%
rh_10_30	1.04e+6	2014	34.67	3.25e+5	2016	32.19	31.15%

Theorem 2 indicates that the standard deviation converges as w increases. In intuition, with sufficiently many times of experiments, the variance should be monotonically decreasing as w increases. The results in Table 4 also show such tendencies. Since we could only experiment with finite times (100 times), there exist errors, e.g., some standard deviations for $w = 3n$ are larger than the ones for $w = 2n$ or the ones for $w = n$. This phenomenon also indicates that the standard deviations are close to the convergence. There is a tradeoff of speed and accuracy, since the larger w, the smaller variance but larger number of steps. We observe that the overall differences between the standard deviations for $w = n$ and $w = 3n$ are small. So, we choose $w = n$ at last.

7.1.3. The comparison of two hit-and-run methods

Table 5 illustrates the running times of 10 million steps of CDHR and HDHR in the intersection of a cube and a ball. This experiment is irrelevant to the procedure of volume estimation. Table 5 shows that CDHR is faster than its rival. The reason is that HDHR has to do more vector multiplications to find intersection points and $m \times n$ more divisions during each step of walk.

In addition, we also compare the two hit-and-run methods on accuracy. We set $w = n$ for both methods and executed 100 times for each instance. The results are listed in Table 6. The column of "ERR." gives the ratios of standard deviations and average values. It clearly shows that the standard deviations of the volume estimated by CDHR method are smaller than HDHR method.

7.1.4. The advantage of reutilization of random points

We conducted experiments to demonstrate the effectiveness of the reutilization technique. We executed `PolyVest` 100 times for each instance. $\overline{n_1}$ ($\overline{n_2}$) represents the average number of newly generated random points without (with) this technique. Table 7 shows that the reutilization technique can save 60% to 70% random points, yet it has no visible effect on the average value and the variance.

Table 8
Comparison between estimation and computation methods for #SMT(LRA).

Instance	NV.	BC.	Estimation		Computation	
			Result	Time (s)	Result	Time (s)
ran_7_15_45	7	116	1.79e+15	2.817	1.84e+15	10.4
ran_7_20_60	7	253	6.85e+14	4.015	6.74e+14	73.2
ran_7_30_90	7	385	4.78e+13	7.994	4.58e+13	872
ran_8_15_45	8	220	3.49e+17	4.311	3.50e+17	71.8
ran_8_20_60	8	456	1.07e+17	11.07	1.09e+17	327
ran_8_30_90	8	1209	6.95e+16	28.69	–	–
ran_9_20_60	9	439	1.21e+19	24.79	–	–
ran_10_20_60	10	949	2.57e+22	58.44	–	–

7.2. Evaluation of `VolCE`

We implement our volume estimation algorithm and preprocessing techniques in a tool called `VolCE`,[4] which is described in [32]. It has the following three functions:

- Estimate volume for SMT(LRA) formulas with `PolyVest`.
- Compute volume for SMT(LRA) formulas with `Vinci` [4].
- Count the number of lattice points for SMT(LIA) formulas with `LattE` [25].

For all experiments in this subsection, we used $S_{min} = 0.01$ and $S_{max} = 1$. The test cases include:

- Random instances `ran_n_l_c`: which have n numeric variables, l LACs and c clauses. They are generated by randomly choosing coefficients of LACs and literals of clauses. The length of each clause is between 3 and 5.
- Instances generated from static program analysis. We analyzed the following programs: (i) `abs`: a function which calculates absolute values; (ii) `findmiddle`: a function which finds the middle number among 3 numbers; (iii) `Space_manage`: a program related to space technology; (iv) `tritype`: a program which determines the type of a triangle; (v) `calDate`: a function which converts the special date into a Julian date; (vi) `tcas`: a program about the traffic collision avoidance system; (vii) `FINDpath`: a selection program FIND [18]; (viii) `getopPath`: a program function called *getop()* [20].
- Instances from SMT-Lib, including the QF_LIA benchmarks: `CAV_2009`, `bignum`, `int_incompleteness`, `pigeon-hole`, `fischer`, `prime_cone`.

The QF_LIA benchmark set is a huge and broad collection of benchmarks, which can be found in the SMT-LIB and is also part of the SMT Competition. It is the standard reference for measuring the performance of linear integer arithmetic solvers. Since `VolCE` is a counter instead of a solver, these benchmarks are usually too difficult for `VolCE`. We scanned this benchmark set and filtered out the complicated instances which cannot be handled by `VolCE` in one hour. At last, we selected 6 families and 112 instances from this benchmark set.

7.2.1. Volume estimation for SMT(LRA) formulas

In this subsection, we experimented our tool `VolCE` on randomly generated SMT(LRA) formulas to evaluate the capability of our volume estimation routine, i.e., `VolCE` with `PolyVest`. Note that the linear constraints in random instances contain almost all variables, the reduction and division preprocessing techniques are not effective for these instances.

Table 8 presents the result of comparing the performance of volume estimation method (`VolCE` with `PolyVest`) and volume computation method (`VolCE` with `Vinci`). In Table 8, column 1 gives the name of instances, column 2 and 3 give the number of numeric variables and partial feasible assignments respectively. The outputs and running times for estimation routine and computation routine are presented in column 4 to column 7. The results show that the volume estimation method for #SMT(LRA) is very efficient and the relative errors of approximation are small. When the dimension of instance grows to 8 or larger, volume computation method often fails to give an answer in one hour or depletes memory. Though `Vinci` has an option to restrict its memory storage, as a tradeoff it will take much more time to solve, and still cannot solve instances within the time limit.

Table 9 presents the results about the effectiveness of our two-round strategy. Columns "n_1" and "n_2" present the number of total random points generated by volume estimation method without and with the two-round strategy respectively. Table 9 shows that the two-round strategy saves 90% to 98% random points and more than 90% of running time. At the same time, the difference of the output results between the original and the two-round strategy is usually less than 5%.

[4] The tool and benchmarks are available at http://lcs.ios.ac.cn/~zj/vc.html.

Table 9
Effectiveness of the two-round strategy.

Instance	BC.	Original			Two-round			n_2/n_1
		Result	Time (s)	n_1	Result	Time (s)	n_2	
ran_7_15_45	116	1.86e+15	26.63	1.03e+7	1.79e+15	2.817	9.34e+5	9.07%
ran_7_20_60	253	6.65e+14	59.68	2.09e+7	6.85e+14	4.015	8.70e+5	4.17%
ran_7_30_90	385	4.53e+13	111.4	3.49e+7	4.78e+13	7.994	1.07e+6	3.06%
ran_8_15_45	220	3.56e+17	108.0	3.36e+7	3.49e+17	4.311	1.09e+6	3.23%
ran_8_20_60	456	1.09e+17	236.4	7.04e+7	1.07e+17	11.07	2.63e+6	3.73%
ran_8_30_90	1209	6.96e+16	723.6	1.91e+8	6.95e+16	28.69	3.58e+6	1.89%
ran_10_15_45	228	1.18e+23	399.4	8.41e+7	1.19e+23	11.77	2.28e+6	2.71%
ran_10_20_60	949	2.55e+22	1801	3.49e+8	2.57e+22	58.44	1.02e+7	2.92%
ran_10_30_90	8039	–	–	–	1.35e+21	361.6	4.56e+7	–
ran_15_40_200	1726	–	–	–	7.88e+27	619.2	3.88e+7	–
ran_15_50_250	495	–	–	–	1.25e+23	217.1	1.18e+7	–
ran_20_60_400	700	–	–	–	6.62e+32	1666	5.19e+7	–

Table 10
Comparison of the tool with and without preprocessing techniques.

Instance	Scale		Original		Reduction		Reduc&Div	
	NV.	Ineq.	Solved (total)	Time (s)	Solved (total)	Time (s)	Solved (total)	Time (s)
abs	1	1	2 (2)	0.061	2 (2)	0.011	2 (2)	0.01
findmiddle	3	6.8	10 (10)	0.998	10 (10)	0.632	10 (10)	0.622
getopPath	2	15	2 (2)	0.392	2 (2)	0.199	2 (2)	0.063
tritype	4	15.0	54 (54)	9.93	54 (54)	5.27	54 (54)	5.39
calDate	6	6.67	21 (21)	1.18	21 (21)	0.378	21 (21)	0.379
FINDpath	8	15.5	2 (2)	0.139	2 (2)	0.115	2 (2)	0.123
Space_manage	17	12.5	56 (56)	–	56 (56)	190	56 (56)	14.8
tcas	24	24.2	0 (1801)	–	1801 (1801)	70.45	1801 (1801)	42.3
CAV_2009	9.17	17.5	2 (6)	0.138	2 (6)	0.753	2 (6)	0.752
bignum	6	13	2 (2)	0.095	2 (2)	0.061	2 (2)	0.059
int_incompletness	3.33	4.67	3 (3)	0.091	3 (3)	0.014	3 (3)	0.014
pigeon-hole	162	347	19 (19)	0.56	19 (19)	1.38	19 (19)	1.38
fischer	28.0	184	47 (49)	1605	47 (49)	1786	47 (49)	1747
prime_cone	11.2	24.7	15 (37)	–	13 (37)	1103	14 (37)	–

7.2.2. Reduction and division techniques

We experimented our reduction and division techniques over instances generated from program analysis and QF_LIA benchmarks that are both SMT(LIA) formulas. Table 10 shows the experimental results about the comparison of the tool with and without the improvements introduced in Section 6. Column "Scale" presents the average scale of the instances in the instance family. Column "Original" presents the results of the original tool. Column "Reduction" presents the results of our tool with the reduction technique. Column "Reduc&Div" presents the results of the improved tool with both of the techniques. For each configuration, the experimental results consist of the number of solved instances and the running times.

Table 10 shows that our preprocessing techniques work well for the instances generated from program analysis. There is no significant improvements for the QF_LIA benchmarks, which are not surprising. In some circumstances, our tool is even slower with preprocessing techniques, since the division may cause overhead. For example, a set of 6-dimensional constraints is divided into three groups of 2-dimensional problems. Then the lattice counting has to be initialized three times for these subproblems.

7.2.3. Performance comparison for SMT(LIA) formulas

To further evaluate the performance of `VolCE` for solution counting, we compare it with `SMTApproxMC` [7] which is a hashing-based approximate counter for SMT(BV) formulas. For comparison, we transformed SMT(LIA) formulas into SMT(BV) formulas manually by replacing integer variables with fixed-length variables, bit-vector constants and bit operations. We experimented `SMTApproxMC` with parameters $\epsilon = 0.8$ and $\delta = 0.2$. It guarantees the output lying in interval $[1.8^{-1}R_F, 1.8R_F]$ with probability at least 80%, where R_F is the real count of a given formula F.

Table 11 presents the result of comparing the performance of `VolCE` with `SMTApproxMC` on a subset of our benchmarks. In these experiments, `VolCE` calls `LattE` for integer solution counting inside a polytope, so our tool returns the exact counts instead of approximations. Table 11 shows that our approach significantly outperforms `SMTApproxMC` for a large class of benchmarks. We observe that the running time of `SMTApproxMC` is closely related to the number of the

Table 11
Comparison between VolCE and SMTApproxMC over SMT(LIA) formulas.

Instance	NV.	BV.	VolCE			SMT approx MC		
			BC.	Result	Time (s)	TB.	Result	Time (s)
FINDpath_1	8	0	1	4.08e+6	0.07	32	3.98e+6	1021
FINDpath_2	8	0	1	87516	0.05	32	90000	116.8
getopPath_1	1	0	6	242	0.02	8	245	2.568
getopPath_2	3	0	18	8085	0.06	24	8381	14.96
findmiddle_4	3	0	2	5527040	0.02	24	–	–
findmiddle_6	3	0	4	130560	0.04	24	1.33e+5	151.5
findmiddle_8	3	0	2	65280	0.02	24	62135	207.9
Space_manage_38	7	0	7	5.41e+14	0.11	56	–	–
Space_manage_49	13	0	7	2.51e+27	0.85	104	–	–
tcas_1200	5	0	6	2.81e+14	0.02	80	–	–
tcas_1201	7	0	34	1.21e+24	0.05	112	–	–
tcas_1214	7	0	10	1.84e+19	0.02	112	–	–
prime_cone_sat_2	2	0	1	4159	0.02	32	3855	5.950
prime_cone_sat_3	3	0	1	25777	0.02	48	24672	284.3
prime_cone_sat_4	4	0	1	75662	0.12	64	65535	1134
prime_cone_sat_5	5	0	1	48505	1.23	80	–	–
prime_cone_sat_6	6	0	1	55143	6.85	96	–	–
prime_cone_sat_7	7	0	1	17823	76.37	112	–	–
FISCHER1-1-fair	4	20	1	256	0.03	40	253	5.825
FISCHER2-7-fair	24	193	35	30135	4.94	240	28749	2463
FISCHER3-8-fair	36	320	565	120540	243.11	360	–	–

solutions rather than the number of variables, i.e., the larger number of solutions, the more difficult for `SMTApproxMC` to handle.

8. Concluding remarks

In contrast to various kinds of decision problems, counting problems have received less attention. We lack practical methods for solving them. This paper studies the counting problem for SMT(LA) constraints. Given a formula/constraint which is a Boolean combination of linear arithmetic inequalities, we would like to know the size of the solution space. Previous exact methods are not scalable.

In this paper, we have described a practical method for estimating the volume of convex polytopes, based on the Multiphase Monte-Carlo method. It employs a new technique to reutilize random points, so that the number of random points can be significantly reduced. We proved that the reutilization technique has no side-effect on the error. We also investigated a simplified version of hit-and-run method: the coordinate directions method. Based on the volume estimation method for polytopes, we presented an approach for estimating the volume of the solution space of SMT(LA) formulas, which is augmented with a heuristic called two round strategy to accelerate the procedure. We also devised some specific techniques for instances that arise from program analysis. The proposed methods have been evaluated on various benchmarks, and the results are promising.

Acknowledgements

We are very grateful to the anonymous reviewers for their helpful comments and suggestions. This work has been supported by the National 973 Program under grant No. 2014CB340701. Peng Zhang is partly supported by the National Natural Science Foundation of China (61672323), the Natural Science Foundation of Shandong Province (ZR2015FM008), and the Fundamental Research Funds of Shandong University (2015JC006).

References

[1] C. Barrett, C.L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, C. Tinelli, CVC4, in: CAV 2011. Proceedings, 2011, pp. 171–177.
[2] C.J.P. Bélisle, H.E. Romeijn, R.L. Smith, Hit-and-run algorithms for generating multivariate distributions, Math. Oper. Res. 18 (2) (1993) 255–266.
[3] H.C.P. Berbee, C.G.E. Boender, A.H.G. Rinnooy Kan, C.L. Scheffer, R.L. Smith, J. Telgen, Hit-and-run algorithms for the identification of nonredundant linear inequalities, Math. Program. 37 (2) (1987) 184–207.
[4] B. Büeler, A. Enge, K. Fukuda, Exact Volume Computation for Polytopes: A Practical Study, 2000, pp. 131–154.
[5] J. Cai, S. Huang, P. Lu, From Holant to #CSP and back: dichotomy for Holant^c problems, Algorithmica 64 (3) (2012) 511–533.
[6] J. Cai, P. Lu, M. Xia, The complexity of complex weighted Boolean #CSP, J. Comput. Syst. Sci. 80 (1) (2014) 217–236.
[7] S. Chakraborty, K.S. Meel, R. Mistry, M.Y. Vardi, Approximate probabilistic inference via word-level counting, in: AAAI 2016. Proceedings, 2016, pp. 3218–3224.
[8] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (6–7) (2008) 772–799.
[9] D. Chistikov, R. Dimitrova, R. Majumdar, Approximate counting in SMT and value estimation for probabilistic programs, in: TACAS 2015. Proceedings, 2015, pp. 320–334.

[10] L.M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: TACAS 2008. Proceedings, 2008, pp. 337–340.
[11] B. Dutertre, Yices 2.2, in: CAV 2014. Proceedings, 2014, pp. 737–744.
[12] M.E. Dyer, A.M. Frieze, On the complexity of computing the volume of a polyhedron, SIAM J. Comput. 17 (5) (1988) 967–974.
[13] M.E. Dyer, A.M. Frieze, R. Kannan, A random polynomial time algorithm for approximating the volume of convex bodies, in: Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 1989, 1989, pp. 375–381.
[14] M. Fredrikson, S. Jha, Satisfiability modulo counting: a new approach for analyzing privacy properties, in: CSL-LICS 2014. Proceedings, 2014, pp. 42:1–42:10.
[15] C. Ge, F. Ma, A fast and practical method to estimate volumes of convex polytopes, in: FAW 2015. Proceedings, 2015, pp. 52–65.
[16] J. Geldenhuys, M.B. Dwyer, W. Visser, Probabilistic symbolic execution, in: ISSTA 2012. Proceedings, 2012, pp. 166–176.
[17] M. Grötschel, L. Lovász, A. Schrijver, Geometric algorithms and combinatorial optimization, Combinatorica (1988).
[18] C.A.R. Hoare, Proof of a program: FIND, Commun. ACM 14 (1) (1971) 39–45.
[19] R. Kannan, L. Lovász, M. Simonovits, Random walks and an $O^*(n^5)$ volume algorithm for convex bodies, Random Structures Algorithms 11 (1) (1997) 1–50.
[20] B.W. Kernighan, D.M. Ritchie, The C Programming Language, 1978.
[21] L.G. Khachiyan, On the complexity of computing the volume of a polytope, Izv. Akad. Nauk SSSR, Eng. Cybern. 3 (1988) 216–217.
[22] L.G. Khachiyan, The problem of computing the volume of polytopes is NP-hard, Uspekhi Mat. Nauk 44 (3) (1989) 199–200.
[23] S. Liu, J. Zhang, Program analysis: from qualitative analysis to quantitative analysis, in: ICSE 2011. Proceedings, 2011, pp. 956–959.
[24] S. Liu, J. Zhang, B. Zhu, Volume computation using a direct Monte Carlo method, in: COCOON 2007. Proceedings, 2007, pp. 198–209.
[25] J.A. De Loera, R. Hemmecke, J. Tauzer, R. Yoshida, Effective lattice point counting in rational convex polytopes, J. Symbolic Comput. 38 (4) (2004) 1273–1302.
[26] L. Lovász, Hit-and-run mixes fast, Math. Program. 86 (3) (1999) 443–461.
[27] L. Lovász, I. Deák, Computational results of an $O^*(n^4)$ volume algorithm, European J. Oper. Res. 216 (1) (2012) 152–161.
[28] L. Lovász, M. Simonovits, The mixing rate of Markov chains, an isoperimetric inequality, and computing the volume, in: 31st Annual Symposium on Foundations of Computer Science, 1990, Vol. I, 1990, pp. 346–354.
[29] L. Lovász, S. Vempala, Where to Start a Geometric Random Walk, 2003.
[30] L. Lovász, S. Vempala, Hit-and-run from a corner, SIAM J. Comput. 35 (4) (2006) 985–1005.
[31] L. Lovász, S. Vempala, Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm, J. Comput. Syst. Sci. 72 (2) (2006) 392–417.
[32] F. Ma, S. Liu, J. Zhang, Volume computation for Boolean combination of linear arithmetic constraints, in: CADE-22, 2009. Proceedings, 2009, pp. 453–468.
[33] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1–2) (1996) 273–302.
[34] R.L. Smith, Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions, Oper. Res. 32 (6) (1984) 1296–1308.
[35] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.
[36] M. Zhou, F. He, X. Song, S. He, G. Chen, M. Gu, Estimating the volume of solution space for satisfiability modulo linear real arithmetic, Theory Comput. Syst. 56 (2) (2015) 347–371.