Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Efficient Volume Computation for SMT Formulas*

Arijit Shaw[1,2], **Uddalok Sarkar**[3], **Kuldeep S. Meel**[4,5]
[1]Chennai Mathematical Institute
[2]IAI, TCG CREST, Kolkata
[3]Indian Statistical Institute, Kolkata
[4]Georgia Institute of Technology
[5]University of Toronto

Abstract

Satisfiability Modulo Theory (SMT) has recently emerged as a powerful tool for solving various automated reasoning problems across diverse domains. Unlike traditional satisfiability methods confined to Boolean variables, SMT can reason on real-life variables like bitvectors, integers, and reals. A natural extension in this context is to ask quantitative questions. One such query in the SMT theory of Linear Real Arithmetic (LRA) is computing the volume of the entire satisfiable region defined by SMT formulas. This problem is important in solving different quantitative verification queries in software verification, cyber-physical systems, and neural networks, to mention a few.

We introduce ttc, an efficient algorithm that extends the capabilities of SMT solvers to volume computation. Our method decomposes the solution space of SMT Linear Real Arithmetic formulas into a union of overlapping convex polytopes, then computes their volumes and calculates their union. Our algorithm builds on recent developments in streaming-mode set unions, volume computation algorithms, and AllSAT techniques. Experimental evaluations demonstrate significant performance improvements over existing state-of-the-art approaches.

1 Introduction

Satisfiability Modulo Theories (SMT) has revolutionized automated reasoning, serving as the foundational technology for diverse problems (Kroening and Strichman 2016). The power of SMT stems from its ability to reason over diverse theories, including bitvectors, reals, and integers, extending well beyond the capabilities of traditional SAT solvers (Barbosa et al. 2022; Barrett et al. 2021; Brummayer and Biere 2009; Cimatti et al. 2013; Niemetz and Preiner 2023). This versatility has established SMT as the de facto decision procedure not only in formal verification of software and hardware workflows (Hajdu and Jovanović 2020; Mattarei et al. 2018), but across numerous domains requiring sophisticated logical reasoning, including security (Backes et al. 2020), test-case generation, synthesis, planning (Cashmore, Magazzeni, and Zehtabi 2020), and optimization (Schkufza, Sharma, and Aiken 2016).

Meanwhile, quantitative reasoning has emerged as a critical advancement in satisfiability solving. Rather than merely determining whether a Boolean formula can be satisfied, model counting (Chakraborty, Meel, and Vardi 2021; Gomes, Sabharwal, and Selman 2021) techniques calculate the number of satisfying assignments — establishing a robust framework for addressing quantitative challenges like probabilistic inference (Chavira and Darwiche 2008), software verification (Teuber and Weigl 2021), network reliability (Duenas-Osorio et al. 2017), neural network verification (Baluta et al. 2019), and numerous other problems (Shaw and Meel 2024).

The natural evolution of these parallel developments leads to the compelling extension to effectively handle quantitative queries within SMT frameworks. This challenge is nuanced by the diversity of the underlying theories, each demanding different approaches. In discrete domains like bitvectors and linear integer arithmetic, the problem manifests as model counting. For linear real arithmetic, it transforms into volume computation or counting distinct regions. Recent years have witnessed remarkable progress across these domains, yielding both theoretical insights and practical algorithms for bitvectors (Chakraborty et al. 2016; Kim and McCamant 2018), linear integers (Ge and Biere 2021; Ge 2024a; Ge et al. 2019), reals (Ge et al. 2018; Ma, Liu, and Zhang 2009), strings (Aydin, Bang, and Bultan 2015), and projected counting over large classes of SMT formulas (Chistikov, Dimitrova, and Majumdar 2017; Shaw and Meel 2025). Apart from (Ge et al. 2018; Ma, Liu, and Zhang 2009), these approaches are mostly limited to discrete domains, and hardly extended to continuous domains. In this work, we address the question: *Given an SMT LRA formula, can we design an efficient volume computation algorithm?*

Our primary contribution is the development of ttc, a novel algorithmic framework that provides an affirmative answer to this question. The ttc algorithm approximates the volume of SMT LRA formula solution spaces with provable theoretical guarantees.

We start with a very related and well-studied problem to SMT volume computation: the problem of volume computation of bounded convex bodies. Sophisticated exact and approximate methods to solve the problem have been developed in the last few decades. Although the exact vol-

*Full version of the paper: https://arxiv.org/abs/2508.09934

ume problem is #P-hard, the seminal work by Dyer, Frieze, and Kannan (1991) showed a polynomial-time randomized approximation algorithm (FPRAS) for this problem. Furthermore, advancements have not only improved the asymptotic running times of these algorithms (Lovász and Simonovits 1990; Applegate and Kannan 1991; Lovász 1991; Lovász and Simonovits 1992; Lovász and Simonovits 1993; Kannan, Lovász, and Simonovits 1997; Lovász and Deák 2012; Cousins and Vempala 2018) but also yielded practically efficient methods that forgo certain theoretical guarantees to eliminate prohibitive hidden constants in their runtime (Cousins and Vempala 2016).

A central challenge in applying these ideas to SMT lies in the non-convex nature of the solution spaces generated by SMT formulas. Unlike convex bodies, non-convex regions are more complex to analyze due to their irregular shapes and potential discontinuities. A natural strategy to overcome this hurdle is to partition the non-convex space into a union of convex bodies, where each convex piece can be more easily managed with existing techniques. Decomposing a non-convex SMT solution space into convex components introduces its own set of challenges. Current state-of-the-art techniques can't handle the union of non-disjoint components. In many cases, the decomposition yields an excessive number of disjoint components, which may not accurately reflect the underlying structure of the solution space. In practice, solution spaces manifest as unions of overlapping, non-disjoint polytopic regions with boundaries and intersections that encode critical constraint information. This overlapping structure is not merely theoretical—our empirical analysis reveals cases where state-of-the-art decomposition techniques transform a natural representation of 7 overlapping polytopes into an unwieldy collection of 20,595 disjoint components, creating unnecessary computational complexity.

Our approach builds upon recent advancements in counting distinct elements across set unions in streaming models by Meel, Vinodchandran and Chakraborty (MVC,2021). The fundamental challenge in adapting the MVC algorithm to volume computation arises from the inherent difference between discrete and continuous domains. The MVC algorithm was specifically engineered for discrete settings, while volume computation operates in continuous space. We overcome this obstacle through a principled discretization approach, effectively reducing continuous volume computation to the problem of counting lattice points within a carefully constructed fine-grained lattice space.

We have implemented ttc and evaluated it on a comprehensive benchmark suite. The results demonstrate significant gains in scalability and accuracy. Out of a benchmark set of 1131 instances, ttc solved 1112, while the current state of the art can solve only 145.

Applications. The theory of linear real arithmetic has significant applications in the formal verification of systems with real variables. These include hybrid systems such as cyber-physical systems (Koley et al. 2023), and control systems (Cimatti, Mover, and Tonetta 2012) and timed systems (Cimatti et al. 2013). Advanced verification tools like Reluplex (Katz et al. 2017) extend SMT solving to neural networks by encoding real-valued variables for network inputs. Extending these approaches to quantitative verification would require LRA solvers with efficient volume computation capabilities. This follows the established pattern where model counting tools have enabled quantitative verification advances in software (Girol, Farinier, and Bardin 2021; Teuber and Weigl 2021) and binarized neural networks (Baluta et al. 2019).

2 Notation and Preliminaries

An SMT (Satisfiability Modulo Theories) formula F is a quantifier-free logical formula over a background theory $\mathcal{T}$ that may contain both theory atoms (e.g., linear arithmetic predicates) and pure Boolean variables. We focus on formulas over Linear Real Arithmetic (LRA), where theory atoms are of the form $a_1x_1 + \cdots + a_nx_n \circ b$, with $a_i, b \in \mathbb{R}$ and $\circ \in \{<, \leq, =, \geq, >\}$.

The Boolean abstraction F_B of F is obtained by replacing each theory atom with a fresh Boolean variable while leaving the original Boolean variables unchanged. We assume that F_B is expressed in Disjunctive Normal Form (DNF) as $F_B = \bigvee_{i=1}^{m} c_i$, where each cube is given by $c_i = \bigwedge_{j=1}^{n_i} \ell_{ij}$, with each ℓ_{ij} being a Boolean literal (either a variable or its negation). And-Inverter Graphs (AIGs) are used as a compact representation for Boolean functions as a circuit, where each node corresponds to a two-input AND gate or an inverter.

We define a mapping M on the Boolean literals corresponding to theory atoms such that for each such literal ℓ, $M(\ell)$ is its corresponding linear inequality (or its negation). Pure Boolean variables are not mapped, as they do not contribute geometric constraints. For each cube c_i, the conjunction $\bigwedge_{j=1}^{n_i} M(\ell_{ij})$ (with the understanding that M is applied only to theory literals) defines a (possibly empty) convex polytope, which we denote by $\mathcal{P}(c_i) = \{x \in \mathbb{R}^n \mid \forall \ell \in c_i$ (theory literal), $M(\ell)(x)$ holds$\}$.

A polyhedron is defined as the intersection of a finite number of half-spaces in $\mathbb{R}^n$, and a polytope is a bounded polyhedron whose volume (measured via the Lebesgue measure) can be computed. Concretely, any polytope K can be written in the form $\{x \in \mathbb{R}^n \mid Ax \leq b\}$, where A is a $m \times n$ matrix and b is a $n \times 1$ vector. Each row of the system $Ax \leq b$ defines one of the m half-spaces (the facets of K). We denote $\mathsf{facet}(K) = m$, $\mathsf{Sol}(K) = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, and $\mathsf{Volume}(K)$ as the number of facets, the feasible region, and the volume of K, respectively.

```
(set−logic QF_LRA)

(declare−const x Real)
(declare−const y Real)

(assert (or
   (and (> x 20) (< x 40) (> y 20) (< y 40))
   (and (> x 10) (< x 30) (> y 10) (< y 30))))

(check−sat)
```

Figure 1: SMT file.

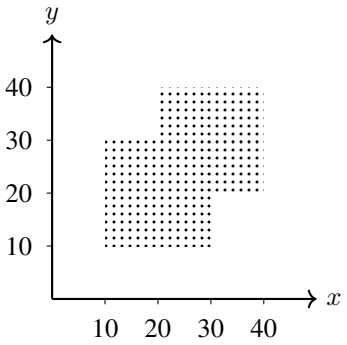


Figure 2: Solution space of SMT formula.

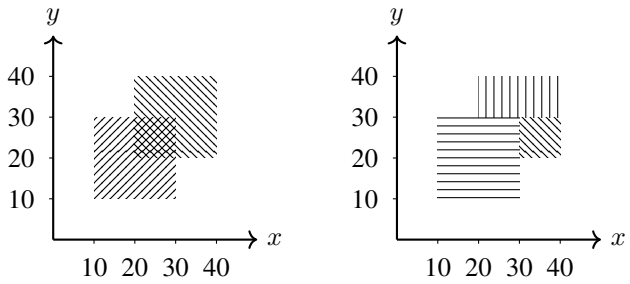


Figure 3: Disjoint (left) and non-disjoint decomposition of the solution space of the formula.

Since we are working over the domain $\mathbb{R}$, precision is crucial in our work. In our setting, we define precision as the number of digits after the decimal point. For example, the number 0.123456789 has a precision of 9.

Illustrative Example. Figure 1 shows the QF_LRA formula over two real variables x and y, which defines two overlapping square regions in the xy-plane. As depicted in Figure 3, each square has an area of 400, with an overlapping area of 100. Thus, the union of the two regions has an area of 700. Internal nodes represent logical conjunctions, and any inverted edges (dots) indicate negations. Figure 3 (left) illustrates a disjoint decomposition of the solution space, where each region is separated so that no two subsets overlap. This often simplifies volume computations but may require more partitions. By contrast, Figure 3 (right) shows a non-disjoint decomposition, allowing subsets to overlap. Figure 4 compares the number of cubes generated by disjoint decomposition and non-disjoint decomposition on our benchmark set.

Problem Statement Let K be the whole solution space of the given formula F, which has d dimensions. Let $\mathcal{B} \subset \mathbb{R}^n$ be an n-dimensional measurable set. The volume of $\mathcal{B}$ is defined as $\mathsf{Volume}(\mathcal{B}) = \int_{\mathcal{B}} d\mathbf{x}$, where $d\mathbf{x}$ denotes the differential volume element. In Cartesian coordinates, this element is expressed as $d\mathbf{x} = dx_1\, dx_2 \cdots dx_n$.

3 Related Work

SMT Volume Computation was first addressed by Ma, Liu, and Zhang, who developed the vinci tool (Ma, Liu, and Zhang 2009) for exact volume computation. Their approach performs intelligent disjoint decomposition of the solution space into convex polytopes using a *bunching* strategy, followed by exact computation of individual polytope volumes. Later, Ge et al. (2018) designed polyvest, which extended vinci by incorporating MCMC-based techniques for polytope volume computation. Recently, Ge introduced the SharpSMT tool (Ge 2024b), which integrates and optimizes various techniques from previous work of polyvest and vinci (Ge et al. 2018; Ma, Liu, and Zhang 2009) to create a more comprehensive solution for SMT volume computation.

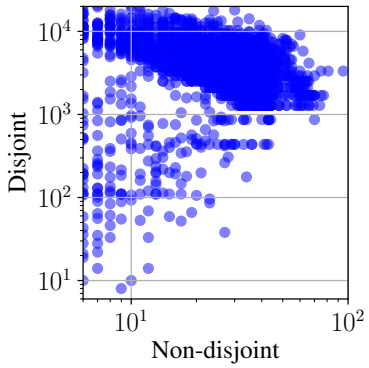


Figure 4: Comparison of the number of polytopes in disjoint and non-disjoint decomposition (notice the non-equal axis).

Weighted Model Integration (WMI) (Belle, Passerini, and Van den Broeck 2015) represents a closely related problem in the hybrid domain of Boolean and rational variables, involving the computation of volume given weight density over the entire domain. This area has seen significant research advances through diverse approaches, including predicate abstraction and All-SMT techniques (Morettin, Passerini, and Sebastiani 2017; Morettin, Passerini, and Sebastiani 2019), knowledge compilation methods (Kolb et al. 2018), and structurally-aware algorithms (Spallitta et al. 2022; Spallitta et al. 2024). However, while WMI focuses primarily on computing weighted integration across domains, our work specifically addresses the fundamental problem of determining the exact volume of the solution space.

Polytope Volume Computation has been a central focus in computational geometry since Dyer and Frieze (1988) proved it #P-hard, followed by Dyer, Frieze, and Kannan's FPRAS development (1991). Rigorous algorithmic advances have improved complexity bounds from $\widetilde{\mathcal{O}}(n^{23})$ to $\widetilde{\mathcal{O}}(n^3)$ (Applegate and Kannan 1991; Kannan, Lovász, and Simonovits 1997; Lovász and Vempala 2006; Lovász and Deák 2012; Cousins and Vempala 2018)[1]. Though these algorithms contain large hidden constants, practical implementations have been achieved by carefully relaxing theoretical guarantees (Cousins and Vempala 2016; Chalkis and Fisikopoulos 2021).

[1] $\widetilde{\mathcal{O}}(\cdot)$ hides the polylogarithmic factors in $\mathcal{O}(\cdot)$.

Estimating the size of a set union has been a well-studied problem since the work of Karp and Luby (1983), who proposed a $\mathcal{O}(m \log^2 |\Omega|)$-time algorithm, where m denotes the number of sets and $|\Omega|$ represents the universe size. Subsequent research has extensively explored streaming settings, where sets arrive sequentially over time (Flajolet and Martin 1985; Gibbons and Tirthapura 2001; Kane, Nelson, and Woodruff 2010). More recent methods employ sampling-based strategies in the context of streaming algorithms (Meel, Vinodchandran, and Chakraborty 2021), which have also been adapted for DNF counting (Soos et al. 2024). These approaches achieve a $\widetilde{\mathcal{O}}(nm)$-time complexity for estimating the solution count of a DNF formula, where m is the number of clauses and n is the number of variables in the DNF formula. Our work builds upon and extends techniques developed in the DNF counting framework.

Volume of a union of polytopes. Abboud, Ceylan, and Dimitrov (2020; 2022) study this problem under a DNF representation of the underlying Boolean formula and compute the volume of the union. Their method builds on the union algorithm of Karp, Luby, and Madras (1989). Our approach is based on MVC, which has shown practical performance advantages over KLM (Soos et al. 2024). Unlike Abboud, Ceylan, and Dimitrov (2020), we permit $\mathcal{O}(\varepsilon)$ error in sampling and volume estimation (vs. $\mathcal{O}(\varepsilon^2/n)$), where ε is the volume-approximation factor and n is the dimension.

4 Algorithm and Analysis

This section presents the core contribution of our work: the ttc algorithm along with its theoretical analysis.

4.1 Algorithm

Given an SMT LRA formula F, the ttc algorithm returns an estimate of $\mathsf{Volume}(F)$. Initially, the algorithm decomposes the solution space of the SMT formula into non-disjoint polytopes and computes the volume for each polytope. Subsequently, it estimates the volume of the union of the polytopes' solution space using a sampling-based approach.

Since we only have a volume computation algorithm for convex polytopes, but the solution space of an SMT formula may be non-convex, we decompose the solution space as a union of convex polytopes. First, we observe that using C, the Boolean abstraction of F in DNF form, we can capture the solution space of F as a union of polytopes. Let $c_i = \bigwedge_{j=1}^{n_i} \ell_{ij}$ be a cube in the DNF. Then, the conjunction $\bigwedge_{j=1}^{n_i} M(\ell_{ij})$ of the corresponding linear inequalities defines a (possibly empty) convex polytope, which we denote by $\mathsf{Polytope}(c_i)$. We can show that $\bigcup_{i=1}^{m} \mathsf{Polytope}(c_i) = \mathsf{Sol}(F)$ (Lemma 1). This relation shows that the DNF representation of the Boolean abstraction of an SMT formula can be used to decompose its solution space into convex polytopes.

To efficiently compute the union of these polytopes, we leverage recent breakthroughs in streaming algorithms for set union operations. The central idea is to compute the union of volumes by maintaining a representative set of points that approximates the total volume. The main caveat of this approach is that the algorithm requires the underlying sets to be finite, while the volume of a polytope cannot be measured directly with a finite number of points. To overcome this issue, we consider an axis-parallel lattice in $\mathbb{R}^n$ with cell side length 10^{-b}, defined as $\mathbb{L}^n = 10^{-b}\mathbb{Z}^n = \{(10^{-b}k_1, 10^{-b}k_2, \ldots, 10^{-b}k_n) \mid k_i \in \mathbb{Z} \text{ for } i = 1, \ldots, n\}$. If b is sufficiently large, we can use this lattice to establish a relationship between the number of lattice points contained within a polytope, $|K \cap \mathbb{L}^n|$, and the volume of the polytope, $\mathsf{Volume}(K)$.

Algorithm 1 $\mathsf{ttc}(F, \varepsilon, \delta)$

1: $F_B^A, M \leftarrow \mathsf{BooleanAbstraction}(F)$
2: $C \leftarrow \mathsf{toDNF}(F_B^A)$
3: $b \leftarrow \mathsf{GetPrecision}(C, M, \frac{\varepsilon}{8})$
4: $\varepsilon' \leftarrow \frac{\varepsilon}{12}$, $\delta' \leftarrow \frac{\delta}{2m}$
5: $\mathsf{Thresh} \leftarrow \max\left(24 \cdot \frac{\ln(24/\delta)}{(1-\varepsilon')\varepsilon'^2}, 6(\ln\frac{6}{\delta} + \ln m)\right)$
6: $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
7: **for** $i = 1$ to m **do**
8: $\quad t \leftarrow \mathsf{ComputeVolume}(\mathsf{Polytope}(c_i), \varepsilon', \delta')$
9: $\quad$ **for** $s \in \mathcal{X}$ **do**
10: $\quad\quad$ **if** $s \in \mathsf{Polytope}(c_i)$ **then** remove s from $\mathcal{X}$
11: $\quad$ **while** $p \geq \frac{\mathsf{Thresh}}{t}$ **do**
12: $\quad\quad$ Remove every element of $\mathcal{X}$ with prob. $1/2$
13: $\quad\quad p \leftarrow p/2$
14: $\quad N_i \leftarrow \mathsf{Poisson}(t \cdot p)$
15: $\quad$ **while** $N_i + |\mathcal{X}| > \mathsf{Thresh}$ **do**
16: $\quad\quad$ Remove every element of $\mathcal{X}$ with prob. $1/2$
17: $\quad\quad N_i \leftarrow \mathsf{Poisson}(t \cdot p/2)$ and $p \leftarrow p/2$
18: $\quad S \leftarrow \mathsf{GenerateSamples}(\mathsf{Polytope}(c_i), N_i, b)$
19: $\quad \mathcal{X}.\mathsf{Append(S)}$
20: Output $|\mathcal{X}|/p$

We present our algorithm, ttc, in Algorithm 1, and in the subsequent part, we describe the algorithm in detail.

In line 1 of ttc, we parse the formula F and construct its Boolean abstraction as a circuit, specifically as an And-Inverter Graph (AIG). Then, in line 2, ttc converts the circuit to DNF. The circuit representation enables us to avoid introducing any auxiliary variables. In essence, converting the circuit to DNF is equivalent to solving a circuit AllSAT problem, for which we leverage recent advances in the literature.

Based on the desired accuracy ε of the volume estimate, we begin by computing the precision parameter b using GetPrecision in line 3, and threshold value Thresh in line 5, which determines approximately how many points will be maintained during the algorithm's execution. In the main loop (lines 7 to 19), we process each cube $\mathsf{Polytope}(c_i)$ sequentially. For each cube, we first compute the (approximate) volume of its corresponding polytope using a polytope volume computation algorithm ComputeVolume (line 8). Next, in line 10, the algorithm removes from the current set $\mathcal{X}$ all solutions that are already accounted for. We then determine the number of solutions N_i that would be sampled from $\mathsf{Polytope}(c_i)$ if each solution were independently sampled with probability p; here, N_i is modeled by a Poisson distribution. Since we wish to keep the size of $\mathcal{X}$ bounded by Thresh, if the sum $|\mathcal{X}| + N_i$ exceeds Thresh, we decrease p and adjust

N_i accordingly—this adjustment is performed by resampling N_i from a Poisson distribution with the revised parameter (p) and by removing elements from $\mathcal{X}$ with probability $1/2$ (line 12). Next, we sample N_i solutions from the cube c_i uniformly at random and add to $\mathcal{X}$—this is essentially done by uniformly sampling a lattice point from $\mathsf{Polytope}(c_i) \cap \mathbb{L}^n$. Finally, the algorithm outputs the final volume estimate $\frac{|\mathcal{X}|}{p}$.

Getting precision. From Kannan and Vempala (1997), we know that if an n-dimensional ℓ_2-ball of radius $\gamma = \Omega(n\sqrt{\log f})$ can be inscribed in an n-dimensional polytope with f facets, then there exists a relationship between the number of integer lattice points inside the polytope and its volume. When working with the lattice $\mathbb{L}^n$ instead of the standard integer lattice $\mathbb{Z}^n$, the effective radius decreases, but the same relationship still applies. Furthermore, as shown in Lemma 3, the same relationship holds for a union of polytopes, provided each polytope contains an ℓ_2-ball of radius $\gamma = \Omega\left(n\sqrt{\log \sum_i \mathsf{facet}(P_i)}\right)$. In Algorithm 2, we process each polytope sequentially (lines 5–9). For each polytope P^i, we compute $\hat{\gamma}$, the maximum radius of a ball that can be inscribed in P^i, using an LP algorithm (line 6). The required precision for this polytope is then determined as $\lceil \log_{10}(\gamma/\hat{\gamma}) \rceil$. After processing all polytopes, we take the maximum of these precision values and return it.

Algorithm 2 $\mathsf{GetPrecision}(C, M, \eta)$

1: $\mathsf{maxRatio} \leftarrow 1,\ r \leftarrow 0$
2: **for** c_i in C **do**
3: $\quad P^i \leftarrow \mathsf{Polytope}(c_i)$
4: $\quad r \leftarrow r + \mathsf{facet}(P^i)$
5: **for** $i = 1$ to m **do**
6: $\quad \hat{\gamma} \leftarrow \mathsf{MaxInscribedBall}(P^i)$
7: $\quad \gamma \leftarrow \frac{16n}{\eta}\sqrt{\log \frac{4r}{\eta}}$
8: $\quad$ **if** $\frac{\gamma}{\hat{\gamma}} > \mathsf{maxRatio}$ **then**
9: $\quad\quad \mathsf{maxRatio} \leftarrow \frac{\gamma}{\hat{\gamma}}$
10: **return** $\lceil \log_{10}(\mathsf{maxRatio}) \rceil$

4.2 Analysis

Theorem 1. *Given a set F, and parameters $\varepsilon, \delta \in (0, 1]$,* ttc *returns an estimate* Est *of* $\mathsf{Volume}(F)$ *such that with probability at least $1 - \delta$,*

$$\mathsf{Est} \in [(1-\varepsilon) \cdot \mathsf{Volume}(F),\ (1+\varepsilon) \cdot \mathsf{Volume}(F)].$$

Theorem 2. *The* ttc *algorithm takes exponential time w.r.t. v in decomposing the polytope, where v is the number of variables in F_B^A. The number of cubes m can be exponential of v as well.*

Proof. The runtime complexity follows from the dual-rail algorithm of circuit AllSAT algorithm. There exists CNF formulas, for which DNF representation is exponential sized, resulting in the exponential blowup of m. □

Theorem 3. *Let m denote the number of decomposed polytopes. Then the* ttc *algorithm runs in time $\mathcal{O}(mn^4)$, where n is the number of dimensions and the $\mathcal{O}(\cdot)$ notation suppresses polylogarithmic factors in ε and δ.*

Proof. The algorithm's main loop (lines 7–19) executes exactly m iterations. In each iteration, the volume computation (line 8) requires $\mathcal{O}(n^4)$ time, while the sampling step (line 18) consumes $\mathcal{O}(n^3)$ time per sample. Since these two operations dominate the computational cost in each iteration, the total runtime is given by $m \times \mathcal{O}(n^4) = \mathcal{O}(mn^4)$, where the polylogarithmic dependencies on ε and δ are hidden in the $\mathcal{O}(\cdot)$ notation. □

Now we prove the theorem 1 using the following lemmas.

Lemma 1. *Let $C = \bigvee_{i=1}^m c_i$ be the DNF abstraction of the SMT formula F. Then,*

$$\bigcup_{i=1}^{m} \mathsf{Polytope}(c_i) = \mathsf{Sol}(F)$$

Proof. First, we show the soundness of each cube: we claim that for every cube c_i in C, $\mathsf{Polytope}(c_i) \subseteq \mathsf{Sol}(F)$. Let $c_i = \bigwedge_{j=1}^{n_i} \ell_{ij}$ be an arbitrary cube of C. By construction, we have $\ell_{i,j} \models C$. Therefore, by the property of Boolean abstraction, if $\bigwedge M(\ell_{i,j})$ can be satisfied, then F is satisfied. Any point $x \in \mathsf{Polytope}(c_i)$ satisfies $\bigwedge M(\ell_{i,j})$, therefore $x \in \mathsf{Sol}(F)$, proving that $\mathsf{Polytope}(c_i) \subseteq \mathsf{Sol}(F)$.

Next, we show the completeness of the union, that $\mathsf{Sol}(F) \subseteq \bigcup_{i=1}^m \mathsf{Polytope}(c_i)$. Consider any arbitrary point $x \in \mathsf{Sol}(F)$. Now x induces a Boolean assignment satisfying C. Since $C = \bigvee_{i=1}^m c_i$, there exists at least one cube c_i such that x satisfies every literal in c_i. By the definition of M, we have that for every $l \in c_i$, $M(l)(x)$ holds, which implies that $x \in \mathsf{Polytope}(c_i)$. Thus, $\mathsf{Sol}(F) \subseteq \bigcup_{i=1}^m \mathsf{Polytope}(c_i)$. □

Lemma 2 (Theorem 3 of Kannan and Vempala (1997))**.** *Let P be a polytope in $\mathbb{R}^n$ that contains an ℓ_2-ball of radius at least $8n\sqrt{\log(2\mathsf{facet}(P)/\eta)}$, where $0 < \eta \leq 1$ is a constant. Then $\mathsf{Volume}(P)$ satisfies the following bounds,*

$$\mathsf{Volume}(P) \in \left[\frac{1-\eta}{2}|P \cap \mathbb{Z}^n|,\ (1+\eta)|P \cap \mathbb{Z}^n|\right]$$

We extend this result to the case of the union of polytopes in the following lemma.

Lemma 3. *Let $Q = \bigcup_{i=1}^m P_i$ with $P_i = \mathsf{Polytope}(c_i)$. If every P_i contains an ℓ_2-ball of radius at least $\frac{16n}{\eta}\sqrt{\log \frac{4r}{\eta}}$ where $0 < \eta \leq 1$ a constant and $r = \sum_{i=1}^m \mathsf{facet}(P_i)$, then*

$$\mathsf{Volume}(Q) \in [(1-\eta)\,|Q \cap \mathbb{Z}^n|, (1+\eta)\,|Q \cap \mathbb{Z}^n|]$$

Proof. Recall the description of P_i as $P_i = \{x \in \mathbb{R}^n \mid A_{ij}x \leq b_{ij}\ (j = 1, \ldots, \mathsf{facet}(P_i))\}$ and set

$$\kappa := \sqrt{2\log\frac{4}{\eta}} + \sqrt{2\log r}, \quad k := \max_{i,j} \frac{b_{ij} + \kappa||A_{ij}||}{b_{ij} - \kappa||A_{ij}||}$$

where $r = \sum_{i=1}^m \mathsf{facet}(P_i)$ and $||\cdot||$ denotes the ℓ_2-norm. Expand/shrink every facet by $\pm\kappa||A_{ij}||$ and write $P_i' := \{x \mid A_{ij}x \leq b_{ij} + \kappa||A_{ij}||\}$, $P_i'' := \{x \mid A_{ij}x \leq b_{ij} - \kappa||A_{ij}||\}$, and let $Q' := \bigcup_i P_i'$, $Q'' := \bigcup_i P_i''$. Let X be

the randomized rounding process as defined in Kannan and Vempala (1997), which takes a point $p \in \mathbb{R}^n$ and returns a point in the lattice $x \in \mathbb{Z}^n$ such that $|x-p| \leq 1$. To complete the proof, we will need the following lemma.

Lemma 4. *The following properties hold for the sets Q' and Q'': **(a)** Draw p uniformly from the continuous body Q'. Then for every $x \in Q \cap \mathbb{Z}^n$, $\Pr[\mathsf{X}_p = x] \geq \frac{1-\eta/2}{\mathsf{Volume}(Q')}$. **(b)** Draw p uniformly from the* continuous *body Q''. The event that p still lies in Q satisfies $\Pr[\mathsf{X}_p \in Q] \geq 1 - \frac{\eta}{2}$, and **(c)** for every $x \in Q \cap \mathbb{Z}^n$, $\Pr[\mathsf{X}_p = x] \leq \frac{1}{\mathsf{Volume}(Q'')}$.*

We now use this lemma to complete the proof. Consider the case where p is drawn uniformly from Q''. Then using lemma 4(b) and 4(c) we have:

$$1 - \frac{\eta}{2} \leq \sum_{x \in Q \cap \mathbb{Z}^n} \Pr[\mathsf{X}_p = x] \leq \sum_{x \in Q \cap \mathbb{Z}^n} \frac{1}{\mathsf{Volume}(Q'')} \leq \frac{|Q \cap \mathbb{Z}^n|}{\mathsf{Volume}(Q)}$$

The last inequality follows from $Q \subseteq Q''$. Since for all $0 < \eta \leq 1$ we have, $\left(1 - \frac{\eta}{2}\right)^{-1} \leq (1+\eta)$, therefore, $\mathsf{Volume}(Q) \leq (1+\eta)|Q \cap \mathbb{Z}^n|$. Again, consider the case where p is drawn uniformly from Q'. Then, using lemma 4(a)

$$1 \geq \sum_{x \in Q \cap \mathbb{Z}^n} \Pr[\mathsf{X}_p = x] \geq \sum_{x \in Q \cap \mathbb{Z}^n} \frac{1-\eta/2}{\mathsf{Volume}(Q')} = \left(1 - \frac{\eta}{2}\right) \cdot \frac{|Q \cap \mathbb{Z}^n|}{\mathsf{Volume}(Q')}$$

Now consider $p \in Q'$. Then $A_{ij} p \leq b'_{ij}$ for all i, j and therefore $A_{ij} \frac{p}{k} \leq b''_{ij}$ for all i, j. This implies that $\frac{p}{k} \in Q''$. This implies that $Q' \subseteq kQ''$. Therefore, $\mathsf{Volume}(Q') \leq k^n \mathsf{Volume}(Q'')$. Since each P_i contains an ℓ_2-ball of radius at least $\frac{16n}{\eta}\sqrt{\log \frac{4r}{\eta}}$, it follows that for the pair i, j at which k is attained, we have $b_{ij} \geq \frac{16n}{\eta}\sqrt{\log \frac{4r}{\eta}}||A_{ij}||$. Consequently,

$$k^n = \left(\frac{\frac{16n}{\eta}\sqrt{\log \frac{4r}{\eta}} + \kappa}{\frac{16n}{\eta}\sqrt{\log \frac{4r}{\eta}} - \kappa}\right)^n$$

Claim 1. *For $n \in \mathbb{N}$ we have,*

$$k^n \leq \frac{1+\eta/4}{1-\eta/4}$$

Using Claim 1, we can upper bound $\mathsf{Volume}(Q')$ as follows: $\mathsf{Volume}(Q') \leq \frac{1+\eta/4}{1-\eta/4} \cdot \mathsf{Volume}(Q'') \leq \frac{1+\eta/4}{1-\eta/4} \cdot \mathsf{Volume}(Q)$. Using the inequality $\frac{1-\eta/4}{1+\eta/4} \cdot \left(1 - \frac{\eta}{2}\right) \geq (1-\eta)$ for all $0 < \eta \leq 1$, we have $\mathsf{Volume}(Q) \geq (1-\eta)|Q \cap \mathbb{Z}^n|$, completing the proof of the lemma. □

Lemma 5. *Let $Q = \bigcup_{i=1}^m P_i$ with $P_i = \mathsf{Polytope}(c_i)$ such that every P_i follows the condition of lemma 3, then we have the following bound on the number of lattice points in Q,*

$$\mathsf{Volume}(Q) \in \left[\frac{1-\eta}{10^{b \cdot n}}|Q \cap \mathbb{L}^n|,\ \frac{1+\eta}{10^{b \cdot n}}|Q \cap \mathbb{L}^n|\right]$$

where b is the precision parameter selected by GetPrecision.

Proof. We define a canonical isomorphism $\phi : \mathbb{L}^n \to \mathbb{Z}^n$. For convenience, we use the same notation to denote the image of a polytope P_i (resp. Q) under ϕ, writing $\phi(P_i)$ (resp. $\phi(Q)$). This mapping scales the distance between any two consecutive points $[a, b]$ along a dimension by 10^b. Consequently, a ball of radius r in $\mathbb{L}^n$ corresponds to a ball of radius $10^b r$ in $\mathbb{Z}^n$. Furthermore, the volume of a polytope P_i (resp. Q) scales by a factor of $10^{b \cdot n}$ when transformed into $\mathbb{Z}^n$, leading to $\mathsf{Volume}(P_i) \cdot 10^{b \cdot n}$ (resp. $\mathsf{Volume}(Q) \cdot 10^{b \cdot n}$). Since the mapping ϕ preserves the lattice structure, we have $|\phi(P_i) \cap \mathbb{Z}^n| = |P_i \cap \mathbb{L}^n|$ and $|\phi(Q) \cap \mathbb{Z}^n| = |Q \cap \mathbb{L}^n|$.

Finally, the choice of b as selected by GetPrecision guarantees that each P_i contains a ball of radius at least $10^{-b}\frac{16n}{\eta}\sqrt{\log(4\sum_{i=1}^m \mathsf{facet}(P_i)/\eta)}$. Therefore, we can apply lemma 3 to complete the proof. □

Lemma 6. *Let $Q = \bigcup_{i=1}^m \mathsf{Polytope}(c_i)$. Then* ttc *outputs an estimate* Est *such that with probability at least $1 - \delta$,*

$$\mathsf{Est} \in \left[\frac{(1-\varepsilon/3)}{10^{b \cdot n}}|Q \cap \mathbb{L}^n|,\ \frac{(1+\varepsilon/3)}{10^{b \cdot n}}|Q \cap \mathbb{L}^n|\right]$$

Now we finish the proof of theorem 1 by combining the results from lemma 6 and lemma 5.

Proof of theorem 1. Using lemma 6, we have $\mathsf{Est} \geq \frac{1-\varepsilon/3}{10^{b \cdot n}} \cdot |Q \cap \mathbb{L}^n|$, and from lemma 5, $\frac{|Q \cap \mathbb{L}^n|}{10^{b \cdot n}} \geq \frac{\mathsf{Volume}(Q)}{(1+\varepsilon/8)}$. Combining these two inequalities, we get $\mathsf{Est} \geq \frac{1-\varepsilon/3}{1+\varepsilon/8} \cdot \mathsf{Volume}(Q) \geq (1-\varepsilon) \cdot \mathsf{Volume}(Q)$. Similarly, the corresponding upper bounds from lemma 6 and lemma 5 we yield $\mathsf{Est} \leq \frac{1+\varepsilon/3}{1-\varepsilon/8} \cdot \mathsf{Volume}(Q) \leq (1+\varepsilon) \cdot \mathsf{Volume}(Q)$. □

4.3 Implementation

We implemented a Python prototype[2] for testing the algorithm and its efficiency. We use the following tools for different parts of the Algorithm 1.

Decomposing into Polytopes. We use a combination of the existing SMT solver and Circuit AllSAT solver to decompose the SMT solution space into convex polytopes. Specifically, we do the following:

Boolean Abstraction. To create the Boolean abstraction of the SMT formula in line 1 of Algorithm 1, we instrument CVC5 (Barbosa et al. 2022) to create the abstraction as an AIG. By default, CVC5 creates the abstraction as a CNF, which we wanted to avoid, since converting to CNF introduces numerous auxiliary variables, making it difficult to enumerate the solutions in a DNF form. For this purpose,

[2]Source code: https://github.com/meelgroup/ttc

we carefully examined each different type of Boolean operation used in SMT formulas, which is typically translated to CNF, including And, Or, Iff, Implies, Ite, and Xor. For each of these operations, we constructed corresponding gates for the AIG. This avoids unnecessary blow-up and retains the semantic structure of the original formula.

Circuit to DNF. To convert the AIG to DNF in line 2, we use the HALL tool (Fried, Nadel, and Shalmon 2023; Fried et al. 2024), which employs a dual-rail based implementation to generate all solutions of the AIG. The solutions are represented as a non-disjoint DNF.

Constructing the Polytopes. While instrumenting CVC5 to generate the Boolean abstraction, we simultaneously capture which variables correspond to which linear inequalities. For each *cube* of the DNF, therefore, we maintain a mapping indicating which set of linear inequalities this cube corresponds to. Given this map and the cube, we construct the polytope.

Polytope Volume Computation. In ComputeVolume (line 8), we employ the Gaussian cooling-based algorithm developed by Cousins and Vempala (2016), which offers a more practical implementation of their theoretically rigorous approach (Cousins and Vempala 2018). While the original algorithm (Cousins and Vempala 2018) provides volume approximation with (ε, δ) guarantees, its prohibitive running time of $10^{16}\mathcal{O}(n^3)$ limits practical applications. The key insight in (Cousins and Vempala 2016) was that these substantial constant factors can be significantly reduced in practical scenarios without severely compromising accuracy, though this comes at the cost of theoretical guarantees. Our implementation utilizes VolEsti, the software package by Chalkis and Fisikopoulos (2021) that implements this practical algorithm.

Preprocessing The polytopes generated by the Polytope(c) procedure may contain redundancies and hidden equalities. Hidden equalities render the polytope *degenerate* - resulting in zero volume in d dimensions. Additionally, redundancies significantly complicate the volume computation for the underlying algorithm. We address these challenges by incorporating CDD as a preprocessing step, which effectively identifies hidden equalities and eliminates redundant inequalities. This preprocessing phase yields substantial performance improvements, particularly valuable when processing inequalities derived from SMT files, which often lack optimal formulation.

Polytope Sampling. For polytope sampling algorithm GenerateSamples in line 18, we employ the Monte Carlo random walk hit-and-run algorithm (Smith 1984). Each random walk begins from a point within the convex body and executes a specified number of steps, termed the *walk length*. Greater walk lengths produce final points less correlated with the starting position. The number of steps required to generate an uncorrelated point, one approximately sampled from a distribution, is known as the *mixing time*. Lovász and Vempala(2004) showed that $10^{10}\mathcal{O}(n^2)$ is a sufficient mixing time for hit and run. However, such requirements are computationally intractable in practice. Following the empirical observations of Lovász and Deák (2012), we therefore constrain our implementation to n steps of hit-and-run, which offers a reasonable balance between sampling quality and computational efficiency.

Compromises. While ttc is theoretically sound, the implementation involves a few compromises that prioritize efficiency over strict adherence to theoretical guarantees:

1. For volume approximation, algorithms with theoretical guarantees require a running length of $10^{16}\mathcal{O}(n^3)$ steps, which is impractical. Therefore, in our implementation (line 8), we use a practical volume algorithm that employs a convergence-based criterion to determine running length.
2. The sampling algorithm in line 18 relies on the hit-and-run method. Known results indicate that achieving independent samples requires $10^{10}n^2$ steps, which is also impractical. However, Lovász and Deák (2012) demonstrated that taking n steps of hit-and-run provides practically independent samples. As a result, we use n steps in our implementation. Notably, the theoretical guarantees of this sampling algorithm are in ℓ_1 norm, that is, the samples are guaranteed to be within ℓ_1 distance of the uniform distribution over the polytope. However, ttc requires samples to be within ℓ_∞ distance of the uniform distribution - a stronger requirement than the ℓ_1 guarantees provided by the hit-and-run sampling algorithm.

It is worth noting that the state-of-the-art tool, SharpSMT (in the non-exact mode i.e., when relying on polyvest), also makes similar compromises, and therefore, ttc and SharpSMT (in non-exact mode) have similar behavior. The exact mode of SharpSMT, on the other hand, fails to scale to larger instances, as demonstrated in the following section.

5 Experimental Evaluation

We evaluated our implementation concerning both efficiency and accuracy.

Baseline. For performance evaluation, we used the current state of the art volume computation framework SharpSMT(Ge 2024b), which offers two distinct modes: the polyvest algorithm, which estimates the volume, and vinci, which performs exact volume computation. To establish meaningful comparisons, we utilized polyvest for performance analysis and vinci for accuracy check. Another relevant baseline is Abboud, Ceylan, and Dimitrov (2020). Their approach assumes a DNF representation, i.e., a set of polytopes as input; for this purpose we would pass the polytopes generated at line 2 of Algorithm 1. The publicly available implementation appears to be an early prototype, and in our environment we observed volume discrepancies on some benchmark-derived instances. Resolving these likely requires additional engineering, so we defer a thorough integration and evaluation of this baseline to future work.

Benchmarks. As a first step, we sought to rely on benchmarks from SMT-Lib, but these benchmarks could not be handled by polyvest or vinci owing to them containing extremely thin geometric regions where dimensions may be constrained to narrow ranges (e.g., $(0,10^{-7})$). In these cases, polyvest and vinci fail to handle the required precision and incorrectly classify these polytopes as degenerate with zero volume. To avoid results confounded by numerical precision rather than

Solver	Solved	PAR-2
vinci	142	6097.63
polyvest	145	6199.73
ttc	**1112**	**255.48**

Table 1: Performance comparison on 1131 instances.

algorithmic differences, we do not include SMT-LIB in our evaluation. Accordingly, we focus on the construction of synthetic instances that are disjunctions of intersecting polytopes, with each polytope defined in H-representation ($Ax \leq b$). In total, our benchmark suite consists of 1131 benchmarks.

1. We vary two parameters: (1) dimension parameter n: ranges from 6 to 34, (2) number of polytopes m: ranges from 6 to 42.
2. For each instance, we generate polytopes using three different geometric shapes studied by Cousins and Vempala (2016): (a) *Cubes:* n-dimensional cubes with random bounds, then translated and rotated. (b) *Zonotypes:* Minkowski sum of n different d-dimensional vectors generated randomly. (c) *Simplex:* Shapes where all coordinates are nonnegative and sum to at most 1, defined as $\{x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i \leq 1, x_i \geq 0\}$.

Environment. We conducted all our experiments on a high-performance computer cluster, with each node consisting of Intel Xeon Gold 6148 CPUs. We allocated one CPU core and a 5GB memory limit to each solver instance pair. To adhere to the standard timeout used in model counting competitions, we set the timeout for all experiments to 3600 seconds. We use values of $\varepsilon = 0.8$ and $\delta = 0.2$, in line with prior work in the model counting community.

With the above setup, we conduct extensive experiments to understand the following:

RQ1. How does the runtime performance of ttc compare to that of polyvest?

RQ2. How does the performance of ttc scale with different benchmark parameters?

RQ3. How accurate is the count computed by ttc in comparison to the exact count?

Summary of Results. ttc achieves a significant performance improvement over polyvest by finishing on 1112 instances in a benchmark set consisting of 1131, while polyvest could only finish on 145 instances. polyvest barely finishes on instances with more than 25 polytopes or 20 dimensions, while ttc seamlessly handles 40 polytopes of 35 dimensions. The accuracy of the approximate count is also noteworthy, with an average error of a count by ttc of only 0.059[3].

5.1 Performance of ttc

Instances Solved. In Table 1, we compare the number of benchmarks that can be solved by polyvest and ttc. First, it

[3]Benchmarks and logfiles: https://doi.org/10.5281/zenodo.16782810

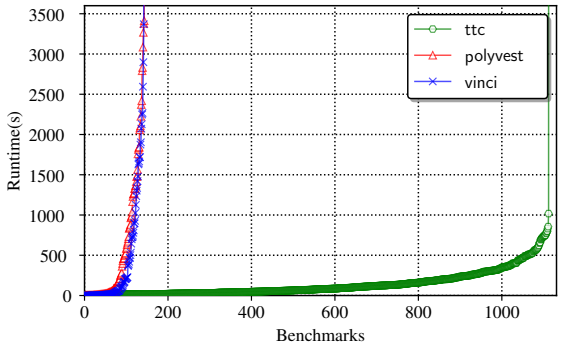


Figure 5: *Cactus plot comparing runtime of different tools.*

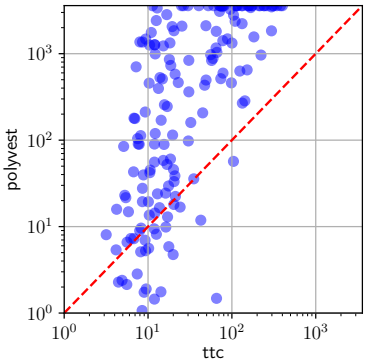


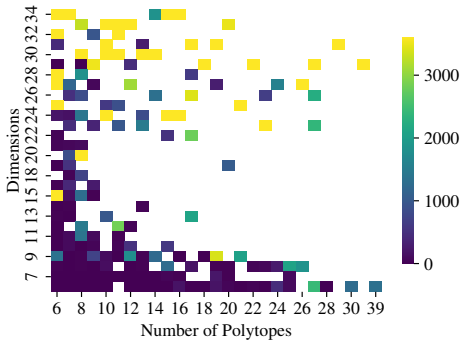
Figure 6: *Runtime comparison of* ttc *w.r.t.* polyvest.

is evident that the polyvest only solved 145 out of the 1131 benchmarks in the test suite, indicating its lack of scalability. Conversely, ttc solved 1112 instances, demonstrating a substantial improvement compared to polyvest.

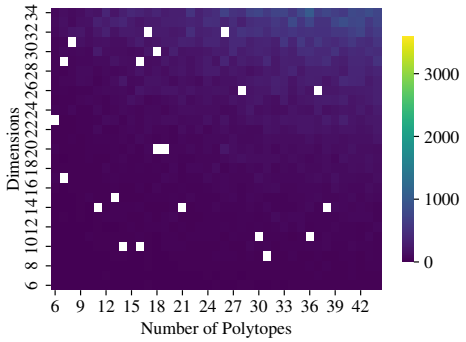
Solving Time Comparison. A performance evaluation of polyvest and ttc is depicted in Figure 5, which is a cactus plot comparing the solving time. The x-axis represents the number of instances, while the y-axis shows the time taken. A point (i, j) in the plot represents that a solver solved j benchmarks out of the 1131 benchmarks in the test suite in less than or equal to j seconds. The curves for polyvest and ttc indicate that for a few instances, polyvest was able to give a quick answer, while in the long run, ttc could solve many more instances given any fixed timeout.

In Table 1 we also show the PAR-2 score of the solvers, which is the mean runtime over all instances, assigning a cost of $2T$ to each instance timed out at T. ttc shows significantly small PAR-2 score. In Figure 6, we present a comparative analysis of solving times between ttc and polyvest. Each data point (x, y) represents an instance that was solved in x seconds by ttc and y seconds by polyvest. Points appearing below the dotted red diagonal line indicate instances where polyvest outperformed ttc in solving time. ttc demonstrates superior performance on the vast majority of instances.

Time utilization in different components. Theoretically, AIG to DNF conversion can take exponential amount of time.



(a) polyvest



(b) ttc

Figure 7: Time taken by w.r.t. dimensions and #polytopes.

Theoretical		0.1	0.4	0.8
Observed	Median	0.03	0.04	0.04
	Max	0.22	0.20	0.39

Table 2: Theoretical vs. observed error at different ε.

However, in our experiments, the DNF conversion time was negligible. The maximum time spent on this conversion was 1.2 seconds, with most instances completing the conversion in less than one second. The majority of the computation time was spent on calculating the individual volumes of the polytopes. For all instances that took more than 100 seconds to solve, over 60% of the time was dedicated to polytope volume computation.

5.2 Scaling

In Figure 7a and 7b, we evaluate the scalability of ttc and polyvest with respect to both the number of polytopes and dimensions. The plots are organized with the number of polytopes increasing from left to right along the x-axis, while the number of dimensions increases from top to bottom along the y-axis. Each pixel corresponds to a specific instance in our benchmark dataset, with the color intensity representing the solver's runtime performance. As demonstrated in Figure 7a, polyvest exhibits significant performance degradation when handling instances exceeding 12 polytopes or 12 dimensions. By contrast, Figure 7b reveals that ttc efficiently processes

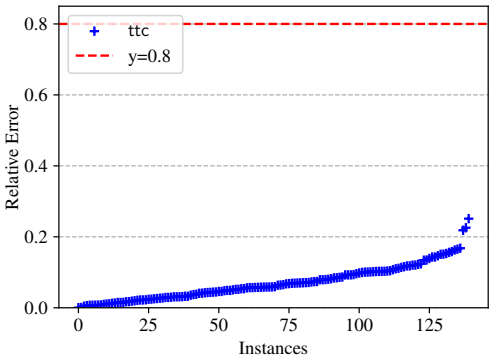


Figure 8: Quality of approximation: observed error.

configurations with up to 40 polytopes and 35 dimensions without notable performance deterioration.

5.3 Quality of Approximation

In our experimental evaluation, we found the exact volume of 142 benchmarks from vinci, enabling us to calculate the error made by ttc on these instances. We quantify the error made by ttc by the parameter $e = \frac{|b-s|}{b}$, where b represents the count from vinci and s from ttc. This measure is the *observed error*, analogous to the theoretical error guarantees provided by ttc. Analysis of all 142 cases found the median e to be 0.059, geometric mean 0.038, and maximum 0.39, contrasting sharply with a theoretical guarantee of 0.8. This signifies ttc substantially outperforms its theoretical bounds. In Figure 8 we plot the observed error, where x-axis, we have the benchmarks, and on the y-axis we have the observed errors. The observed error is below 0.2 for most of the instances.

In Table 2 we showed different observed errors when we run ttc with different ε values. The median and maximum observed errors decrease with the theoretical ε. The maximum *observed error* with $\varepsilon = 0.1$, is greater than theoretical, which is not unnatural, given the (ε, δ) guarantee nature.

6 Conclusion

This paper introduces ttc, a scalable approximate SMT volume computation tool that demonstrates exceptional performance on practical benchmarks. Our approach harnesses probabilistic techniques to deliver theoretical guarantees on computation results, and empirical results significantly surpassing theoretical guarantees. Our work suggests several promising research directions. First, many formulas contain equality constraints that result in zero volume when computing in d dimensions. A natural extension would be to develop methods for correctly computing $(d-k)$-dimensional volume in such cases. Second, while we prioritized performance over strict theoretical guarantees in our implementation, experimental results consistently demonstrate error rates well below theoretical bounds. This raises the intriguing question, whether rigorous guarantees can be established for our current implementation without sacrificing its performance advantages.

Acknowledgements

We are thankful to Sourav Chakraborty, Radoslav Dimitrov, Arijit Ghosh and Mate Soos for the many useful discussions. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) [RGPIN-2024-05956]. The work was done when Arijit Shaw was a visiting graduate student at the University of Toronto. Uddalok Sarkar was supported by Google PhD fellowship. Computations were performed on the Niagara supercomputer at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario Research Fund: Research Excellence; and the University of Toronto.

References

Abboud, R.; Ceylan, İ. İ.; and Dimitrov, R. 2020. On the approximability of weighted model integration on DNF structures. In *Proc. of KR*.

Abboud, R.; Ceylan, İ. İ.; and Dimitrov, R. 2022. Approximate weighted model integration on DNF structures. *Artif. Intell.*

Applegate, D., and Kannan, R. 1991. Sampling and integration of near log-concave functions. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 156–163.

Aydin, A.; Bang, L.; and Bultan, T. 2015. Automata-based model counting for string constraints. In *Proc. of CAV*.

Backes, J.; Berrueco, U.; Bray, T.; Brim, D.; Cook, B.; Gacek, A.; Jhala, R.; Luckow, K.; McLaughlin, S.; Menon, M.; et al. 2020. Stratified abstraction of access control policies. In *Proc. of CAV*.

Baluta, T.; Shen, S.; Shinde, S.; Meel, K. S.; and Saxena, P. 2019. Quantitative verification of neural networks and its security applications. In *Proc. of CCS*.

Barbosa, H.; Barrett, C.; Brain, M.; Kremer, G.; Lachnitt, H.; Mann, M.; Mohamed, A.; Mohamed, M.; Niemetz, A.; Nötzli, A.; et al. 2022. cvc5: A versatile and industrial-strength smt solver. In *Proc. of TACAS*.

Barrett, C.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2021. Satisfiability modulo theories. In *Handbook of satisfiability*.

Belle, V.; Passerini, A.; and Van den Broeck, G. 2015. Probabilistic inference in hybrid domains by weighted model integration. In *Proc. of IJCAI*.

Brummayer, R., and Biere, A. 2009. Boolector: An efficient SMT solver for bit-vectors and arrays. In *Proc. of TACAS*.

Cashmore, M.; Magazzeni, D.; and Zehtabi, P. 2020. Planning for hybrid systems via satisfiability modulo theories. *Journal of Artificial Intelligence Research*.

Chakraborty, S.; Meel, K.; Mistry, R.; and Vardi, M. 2016. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2021. Approximate model counting. In *Handbook of Satisfiability*.

Chalkis, A., and Fisikopoulos, V. 2021. volesti: Volume Approximation and Sampling for Convex Polytopes in R. *R Journal* (2).

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence*.

Chistikov, D.; Dimitrova, R.; and Majumdar, R. 2017. Approximate counting in smt and value estimation for probabilistic programs. *Acta Informatica* 54(8):729–764.

Cimatti, A.; Griggio, A.; Schaafsma, B. J.; and Sebastiani, R. 2013. The mathsat5 SMT solver. In *Proc. of TACAS*.

Cimatti, A.; Mover, S.; and Tonetta, S. 2012. Smt-based verification of hybrid systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2100–2105.

Cousins, B., and Vempala, S. 2016. A practical volume algorithm. *Mathematical Programming Computation* (2).

Cousins, B., and Vempala, S. 2018. Gaussian cooling and oˆ*(nˆ3) algorithms for volume and gaussian volume. *SIAM Journal on Computing* 47(3):1237–1273.

Duenas-Osorio, L.; Meel, K.; Paredes, R.; and Vardi, M. 2017. Counting-based reliability estimation for power-transmission grids. In *Proc. of AAAI*.

Dyer, M. E., and Frieze, A. M. 1988. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing* 17(5):967–974.

Dyer, M.; Frieze, A.; and Kannan, R. 1991. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM (JACM)* 38(1):1–17.

Flajolet, P., and Martin, G. N. 1985. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences* 31(2):182–209.

Fried, D.; Nadel, A.; Sebastiani, R.; and Shalmon, Y. 2024. Entailing generalization boosts enumeration. In *Proc. of SAT*.

Fried, D.; Nadel, A.; and Shalmon, Y. 2023. Allsat for combinational circuits. In *Proc. of SAT*.

Ge, C., and Biere, A. 2021. Decomposition strategies to count integer solutions over linear constraints. In *Proc. of IJCAI*.

Ge, C.; Ma, F.; Zhang, P.; and Zhang, J. 2018. Computing and estimating the volume of the solution space of SMT (LA) constraints. *Theoretical Computer Science*.

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 *Proc. of IJCAI*.

Ge, C. 2024a. Approximate integer solution counts over linear arithmetic constraints. In *Proc. of AAAI*.

Ge, C. 2024b. sharpsmt: A scalable toolkit for measuring solution spaces of smt(la) formulas. *Frontiers of Computer Science (FCS)*.

Gibbons, P. B., and Tirthapura, S. 2001. Estimating simple functions on the union of data streams. In *Proc. of SPAA*, 281–291.

Girol, G.; Farinier, B.; and Bardin, S. 2021. Not all bugs are created equal, but robust reachability can tell the difference. In *Proc. of CAV*. Springer.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2021. Model counting. In *Handbook of satisfiability*.

Hajdu, Á., and Jovanović, D. 2020. solc-verify: A modular verifier for solidity smart contracts. In *Proc. of VSTTE*.

Kane, D. M.; Nelson, J.; and Woodruff, D. P. 2010. An optimal algorithm for the distinct elements problem. In *Proc. of PODS*, 41–52.

Kannan, R., and Vempala, S. 1997. Sampling lattice points. In *Proc. of STOC*.

Kannan, R.; Lovász, L.; and Simonovits, M. 1997. Random walks and an o*(n5) volume algorithm for convex bodies. *Random Structures & Algorithms* 11(1):1–50.

Karp, R. M., and Luby, M. 1983. Monte-carlo algorithms for enumeration and reliability problems. In *Proc. of FOCS*, 56–64.

Karp, R. M.; Luby, M.; and Madras, N. 1989. Monte-carlo approximation algorithms for enumeration problems. *Journal of algorithms* 10(3):429–448.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient smt solver for verifying deep neural networks. In *Proc. of CAV*.

Kim, S., and McCamant, S. 2018. Bit-vector model counting using statistical estimation. In *Proc. of TACAS*.

Kolb, S.; Mladenov, M.; Sanner, S.; Belle, V.; and Kersting, K. 2018. Efficient symbolic integration for probabilistic inference. In *Proc. of IJCAI*.

Koley, I.; Dey, S.; Mukhopadhyay, D.; Singh, S.; Lokesh, L.; and Ghotgalkar, S. V. 2023. CAD Support for Security and Robustness Analysis of Safety-critical Automotive Software. *ACM Transactions on Cyber-Physical Systems*.

Kroening, D., and Strichman, O. 2016. *Decision procedures*.

Lovász, L., and Deák, I. 2012. Computational results of an O(n4) volume algorithm. *European journal of operational research* 216(1):152–161.

Lovász, L., and Simonovits, M. 1990. The mixing rate of markov chains, an isoperimetric inequality, and computing the volume. In *Proceedings [1990] 31st annual symposium on foundations of computer science*, 346–354. IEEE.

Lovász, L., and Simonovits, M. 1992. On the randomized complexity of volume and diameter. In *Proc. of FOCS*, 482–492. IEEE Computer Society.

Lovász, L., and Simonovits, M. 1993. Random walks in a convex body and an improved volume algorithm. *Random structures & algorithms* 4(4):359–412.

Lovász, L., and Vempala, S. 2004. Hit-and-run from a corner. In *Proc. of STOC*, 310–314.

Lovász, L., and Vempala, S. 2006. Simulated annealing in convex bodies and an o*(n4) volume algorithm. *Journal of Computer and System Sciences* 72(2):392–417.

Lovász, L. 1991. *How to compute the volume?* DIMACS, Center for Discrete Mathematics and Theoretical Computer Science.

Ma, F.; Liu, S.; and Zhang, J. 2009. Volume computation for boolean combination of linear arithmetic constraints. In *Proc. of CADE*.

Mattarei, C.; Mann, M.; Barrett, C.; Daly, R. G.; Huff, D.; and Hanrahan, P. 2018. Cosa: Integrated verification for agile hardware design. In *Proc. of FMCAD*.

Meel, K. S.; Vinodchandran, N.; and Chakraborty, S. 2021. Estimating the size of union of sets in streaming models. In *Proc. of PODS*.

Morettin, P.; Passerini, A.; and Sebastiani, R. 2017. Efficient weighted model integration via smt-based predicate abstraction. In *Proc. of AAAI*.

Morettin, P.; Passerini, A.; and Sebastiani, R. 2019. Advanced smt techniques for weighted model integration. *Artificial Intelligence*.

Niemetz, A., and Preiner, M. 2023. Bitwuzla. In *Proc. of CAV*.

Schkufza, E.; Sharma, R.; and Aiken, A. 2016. Stochastic program optimization. *Communications of the ACM* (2).

Shaw, A., and Meel, K. S. 2024. Model counting in the wild. In *Proc. of Knowledge Representation and Reasoning (KR)*.

Shaw, A., and Meel, K. S. 2025. Approximate smt counting beyond discrete domains. In *Proc. of DAC*.

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

Soos, M.; Sarkar, U.; Aggarwal, D.; Chakraborty, S.; Meel, K. S.; and Obremski, M. 2024. Engineering an efficient approximate dnf-counter. *arXiv preprint arXiv:2407.19946*.

Spallitta, G.; Masina, G.; Morettin, P.; Passerini, A.; and Sebastiani, R. 2022. Smt-based weighted model integration with structure awareness. In *Uncertainty in Artificial Intelligence*, 1876–1885. PMLR.

Spallitta, G.; Masina, G.; Morettin, P.; Passerini, A.; and Sebastiani, R. 2024. Enhancing smt-based weighted model integration by structure awareness. *Artificial Intelligence* 328:104067.

Teuber, S., and Weigl, A. 2021. Quantifying software reliability via model-counting. In *Proc. of QEST*.