Efficient Volume Computation for SMT Formulas

Arijit Shaw
IAI, TCG CREST, Kolkata, India
Chennai Mathematical Institute, India

Uddalok Sarkar
Indian Statistical Institute, Kolkata, India

Kuldeep S. Meel
Georgia Institute of Technology, USA
University of Toronto, Canada

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.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Volume Computation, SMT, Model Counting

1 Introduction

Satisfiability Modulo Theories (SMT) has revolutionized automated reasoning, serving as the foundational technology for diverse problems [32]. 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 [6, 7, 9, 16, 42]. This versatility has established SMT as the de facto decision procedure not only in formal verification of software and hardware workflows [29, 40], but across numerous domains requiring sophisticated logical reasoning, including security [4], test-case generation, synthesis, planning [10], and optimization [43].

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 [12, 28] techniques calculate the number of satisfying assignments — establishing a robust framework for addressing quantitative challenges like probabilistic inference [14], software verification [47], network reliability [19], neural network verification [5], and numerous other problems [44].

[0] This paper was published in the proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR) 2025. Full version available at https://arxiv.org/abs/2508.09934

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. Chistikov, Dimitrova and Majumdar [15] presented a notation of *measured theories*, unifying quantitative problems across different discrete and continuous thories and theory combinations. Recent years have witnessed remarkable progress across these domains, yielding both theoretical insights and practical algorithms for bitvectors [11, 15, 31], linear integers [15, 25, 23, 26], reals [15, 27, 39], strings [3], and projected counting over large classes of SMT formulas [15, 45]. Apart from [39, 15, 27], 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 volume problem is #P-hard, the seminal work by Dyer, Frieze and Kannan [20] showed a polynomial-time randomized approximation algorithm (FPRAS) for this problem. Furthermore, advancements have not only improved the asymptotic running times of these algorithms [35, 2, 33, 36, 37, 30, 34, 18] but also yielded practically efficient methods that forgo certain theoretical guarantees to eliminate prohibitive hidden constants in their runtime [17].

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 [41]. 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.

2 Algorithm and Analysis

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.

Algorithm 1 ttc(F, ε, δ)

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

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

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 $\frac{|\mathcal{X}|}{p}$.

The description of GetPrecision subroutine is deferred to the final version of the paper.

2.1 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.*

▶ **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. Deferred to the extended version. ◀

2.2 Implementation

We implemented a Python prototype 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 [6] 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 each of these Boolean operations of SMT formula, 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 [22, 21], 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 [17], which offers a more practical implementation of their theoretically rigorous approach [18]. While the original algorithm [18] 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 [17] 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 [13] 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 [46]. 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 [38] 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 [34], we therefore constrain our implementation to n steps of hit-and-run, which offers a reasonable balance between sampling quality and computational efficiency.

3 Experimental Evaluation

We evaluated our implementation concerning both efficiency and accuracy[1].

Baseline. For performance evaluation, we used the current state of the art volume computation framework SharpSMT[24], 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. We also compare our performance against hashing-based volume computation tool by [15]. We use the name cdm for this tool, by initials of the authors. Another possible baseline is to replace the union part of ttc algorithm (lines 3 - 19) with the algorithm of [8, 1]. We implemented their algorithm, and used it as a baseline. We name this tool as bf/acd.

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 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 [17]: (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:

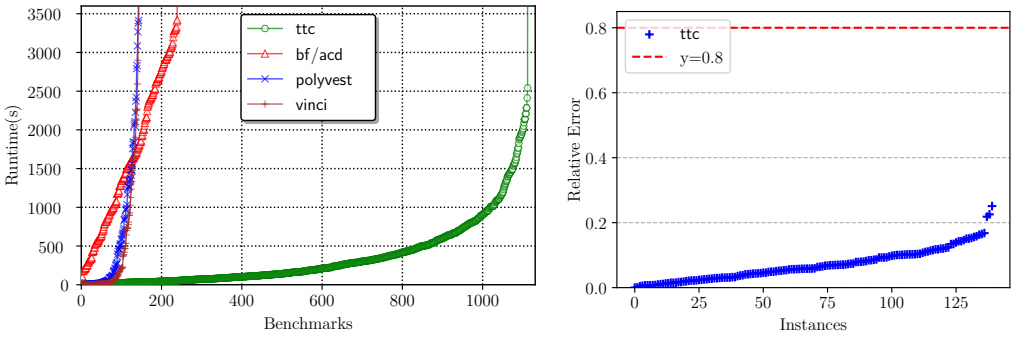
RQ1. How does the runtime performance of ttc compare to the baselines?
RQ2. How does the performance of ttc scale with different benchmark parameters?
RQ3. How accurate is the count computed by ttc in comparison to the exact count?

Summary of Results. ttc achieves a significant performance improvement over baselines, by finishing on 1112 instances in a benchmark set consisting of 1131, while polyvest could only finish on 145 instances, and bf/acd finished on 250 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.

[1] All benchmarks and logfiles are available at https://doi.org/10.5281/zenodo.16782810

Solver	Solved	PAR-2
ttc	**1112**	**255.48**
bf/acd	250	5982.03
vinci	142	6097.63
polyvest	145	6199.73

Table 1 Performance comparison on 1131 instances.



(a) Cactus plot comparing runtimes.

(b) Quality of approximation: observed error.

Figure 1 Performance and Accuracy

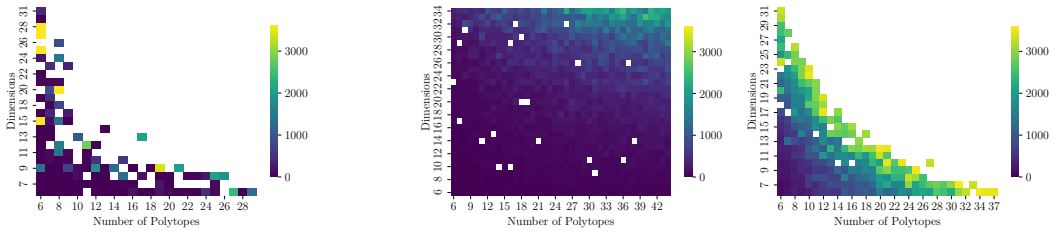
Performance of ttc

Instances Solved. In Table 1, we compare the number of benchmarks that can be solved by baselines and ttc. First, it is evident that the polyvest only solved 145 out of the 1131 benchmarks in the test suite, bf/acd solved 250 instances. Conversely, ttc solved 1112 instances, a substantial improvement compared to polyvest. cdm solved none of the instances.

Solving Time Comparison. A performance evaluation of polyvest, bf/acd and ttc is depicted in Figure 1a, 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, bf/acd and ttc indicate that for a few instances, polyvest and bf/acd 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.

Performance of existing tools. We examined why other tools do not scale well:

1. polyvest and vinci use disjoint decomposition. In some instances this yields an excessive number of convex polytope components. Because the algorithm must run the $\mathcal{O}(n^3)$ polytope volume computation on each component, the component count becomes a bottleneck.
2. bf/acd requires running the polytope volume computation with extremely high precision. For $m = 20$ and $\varepsilon = 0.8$, the maximum allowable ε_V is 0.00068. Computing volumes at this precision is slow, limiting scalability for higher-dimensional instances.
3. cdm needs many XOR-based hash constraints to reach the required precision during volume computation. For example, the smallest benchmark in our suite uses a 198-bit representation.



(a) polyvest **(b)** ttc **(c)** bf/acd

Figure 2 Time taken w.r.t. dimensions and # polytopes.

Scaling

In Figure 2a and 2c, 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 2a, polyvest exhibits significant performance degradation when handling instances exceeding 12 polytopes or 12 dimensions. By contrast, Figure 2c reveals that ttc efficiently processes configurations with up to 40 polytopes and 35 dimensions.

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 1b we plot the observed error, where x-axis, we have the benchmarks, and on the y-axis we have the observed errors.

4 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.

References

1 Ralph Abboud, İsmail İlkan Ceylan, and Radoslav Dimitrov. Approximate weighted model integration on DNF structures. *Artif. Intell.*, 2022.

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

3 Abdulbaki Aydin, Lucas Bang, and Tevfik Bultan. Automata-based model counting for string constraints. In *Proc. of CAV*, 2015.

4 John Backes, Ulises Berrueco, Tyler Bray, Daniel Brim, Byron Cook, Andrew Gacek, Ranjit Jhala, Kasper Luckow, Sean McLaughlin, Madhav Menon, et al. Stratified abstraction of access control policies. In *Proc. of CAV*, 2020.

5 Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. In *Proc. of CCS*, 2019.

6 Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. cvc5: A versatile and industrial-strength smt solver. In *Proc. of TACAS*, 2022.

7 Clark Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of satisfiability*. 2021.

8 Karl Bringmann and Tobias Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry*, 43, 2010.

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

10 Michael Cashmore, Daniele Magazzeni, and Parisa Zehtabi. Planning for hybrid systems via satisfiability modulo theories. *Journal of Artificial Intelligence Research*, 2020.

11 Supratik Chakraborty, Kuldeep Meel, Rakesh Mistry, and Moshe Vardi. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*, 2016.

12 Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. Approximate model counting. In *Handbook of Satisfiability*. 2021.

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

14 Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 2008.

15 Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. Approximate counting in smt and value estimation for probabilistic programs. *Acta Informatica*, 54(8):729–764, 2017.

16 Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 SMT solver. In *Proc. of TACAS*, 2013.

17 Ben Cousins and Santosh Vempala. A practical volume algorithm. *Mathematical Programming Computation*, (2), 2016.

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

19 Leonardo Duenas-Osorio, Kuldeep Meel, Roger Paredes, and Moshe Vardi. Counting-based reliability estimation for power-transmission grids. In *Proc. of AAAI*, 2017.

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

21 Dror Fried, Alexander Nadel, Roberto Sebastiani, and Yogev Shalmon. Entailing generalization boosts enumeration. In *Proc. of SAT*, 2024.

22 Dror Fried, Alexander Nadel, and Yogev Shalmon. Allsat for combinational circuits. In *Proc. of SAT*, 2023.

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

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

347 **25** Cunjing Ge and Armin Biere. Decomposition strategies to count integer solutions over linear
348 constraints. In *Proc. of IJCAI*, 2021.
349 **26** Cunjing Ge, Feifei Ma, Xutong Ma, Fan Zhang, Pei Huang, and Jian Zhang. Approximating
350 integer solution counting via space quantification for linear constraints. In *Proc. of IJCAI*,
351 2019.
352 **27** Cunjing Ge, Feifei Ma, Peng Zhang, and Jian Zhang. Computing and estimating the volume
353 of the solution space of SMT (LA) constraints. *Theoretical Computer Science*, 2018.
354 **28** Carla P Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of*
355 *satisfiability*. 2021.
356 **29** Ákos Hajdu and Dejan Jovanović. solc-verify: A modular verifier for solidity smart contracts.
357 In *Proc. of VSTTE*, 2020.
358 **30** Ravi Kannan, László Lovász, and Miklós Simonovits. Random walks and an o*(n5) volume
359 algorithm for convex bodies. *Random Structures & Algorithms*, 11(1):1–50, 1997.
360 **31** Seonmo Kim and Stephen McCamant. Bit-vector model counting using statistical estimation.
361 In *Proc. of TACAS*, 2018.
362 **32** Daniel Kroening and Ofer Strichman. *Decision procedures*. 2016.
363 **33** László Lovász. *How to compute the volume?* DIMACS, Center for Discrete Mathematics and
364 Theoretical Computer Science, 1991.
365 **34** László Lovász and István Deák. Computational results of an O(n4) volume algorithm. *European*
366 *journal of operational research*, 216(1):152–161, 2012.
367 **35** László Lovász and Miklós Simonovits. The mixing rate of markov chains, an isoperimetric
368 inequality, and computing the volume. In *Proceedings [1990] 31st annual symposium on*
369 *foundations of computer science*, pages 346–354. IEEE, 1990.
370 **36** László Lovász and Miklós Simonovits. On the randomized complexity of volume and diameter.
371 In *Proc. of FOCS*, pages 482–492. IEEE Computer Society, 1992.
372 **37** László Lovász and Miklós Simonovits. Random walks in a convex body and an improved
373 volume algorithm. *Random structures & algorithms*, 4(4):359–412, 1993.
374 **38** László Lovász and Santosh Vempala. Hit-and-run from a corner. In *Proc. of STOC*, pages
375 310–314, 2004.
376 **39** Feifei Ma, Sheng Liu, and Jian Zhang. Volume computation for boolean combination of linear
377 arithmetic constraints. In *Proc. of CADE*, 2009.
378 **40** Cristian Mattarei, Makai Mann, Clark Barrett, Ross G Daly, Dillon Huff, and Pat Hanrahan.
379 Cosa: Integrated verification for agile hardware design. In *Proc. of FMCAD*, 2018.
380 **41** Kuldeep S. Meel, N.V. Vinodchandran, and Sourav Chakraborty. Estimating the size of union
381 of sets in streaming models. In *Proc. of PODS*, 2021.
382 **42** Aina Niemetz and Mathias Preiner. Bitwuzla. In *Proc. of CAV*, 2023.
383 **43** Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic program optimization. *Communic-*
384 *ations of the ACM*, (2), 2016.
385 **44** Arijit Shaw and Kuldeep S Meel. Model counting in the wild. In *Proc. of Knowledge*
386 *Representation and Reasoning (KR)*, 2024.
387 **45** Arijit Shaw and Kuldeep S Meel. Approximate smt counting beyond discrete domains. In
388 *Proc. of DAC*, 2025.
389 **46** Robert L Smith. Efficient monte carlo procedures for generating points uniformly distributed
390 over bounded regions. *Operations Research*, 32(6):1296–1308, 1984.
391 **47** Samuel Teuber and Alexander Weigl. Quantifying software reliability via model-counting. In
392 *Proc. of QEST*, 2021.