Efficient Volume Computation for SMT Formulas*

Arijit Shaw

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

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.

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 state-of-the-art approaches.

Keywords

Satisfiability Modulo Theories, Quantitative Reasoning, Volume Computation

1. Introduction

Satisfiability Modulo Theories (SMT) has revolutionized automated reasoning, serving as the foundational technology for diverse problems. 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. This versatility has established SMT as the de facto decision procedure not only in formal verification of software and hardware workflows, but across numerous domains requiring sophisticated logical reasoning, including security, synthesis, planning, and optimization.

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 techniques calculate the number of satisfying assignments - establishing a robust framework for addressing quantitative challenges like probabilistic inference, software verification, network reliability, neural network verification, and numerous other problems [2].

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, linear integers, and strings. But these approaches are mostly limited to discrete domains, and hardly been 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.

Doctoral Consortium of the 22nd International Conference on Principles of Knowledge Representation and Reasoning (KR 2025 DC), November 11-17, 2025, Melbourne, Australia

* This extended abstract summarizes results from joint work with Uddalok Sarkar and Kuldeep S. Meel that appeared in the Proceedings of KR 2025 [1]. The full version is available at arXiv:2508.09934.

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 et al. [3] showed a polynomial-time randomized approximation algorithm (FPRAS) for this problem. Furthermore, advancements have not only improved the asymptotic running times of these algorithms, but also yielded practically efficient methods that forgo certain theoretical guarantees to eliminate prohibitive hidden constants in their runtime [4].

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 et al. (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.

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 $V(\mathcal{B}) = \int_{\mathcal{B}} d\mathbf{x}$, where $d\mathbf{x}$ denotes the differential volume element. In Cartesian coordinates, this element is expressed as $d\mathbf{x} = dx_1\, dx_2 \cdots dx_n$.

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 [6], and control systems [7] and timed systems [8]. Advanced verification tools like Reluplex [9] 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 [10, 11] and binarized neural networks [12].

2. Algorithm and Analysis

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

2.1. 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.

Algorithm 1 ttc(F, ε, δ)

1: F_B^A,$M \leftarrow$ BooleanAbstraction(F)
2: $C \leftarrow$ toDNF(F_B^A)
3: $b \leftarrow$ GetPrecision$(C, M, \frac{\varepsilon}{4})$
4: $\varepsilon' \leftarrow \frac{\varepsilon}{12}$
5: 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: $\quad t \leftarrow$ Volume(Polytope$(c_i), \varepsilon', \delta')$
9: $\quad$ **for** $s \in \mathcal{X}$ **do**
10: $\quad\quad$ **if** $s \in$ Polytope(c_i) **then** remove s from $\mathcal{X}$
11: $\quad$ **while** $p \geq \frac{\text{Thresh}}{t}$ **do**
12: $\quad\quad$ Remove every element of $\mathcal{X}$ with prob. $1/2$
13: $\quad\quad p \leftarrow p/2$
14: $\quad N_i \leftarrow$ Poisson$(t \cdot p)$
15: $\quad$ **while** N_i + $|\mathcal{X}| >$ Thresh **do**
16: $\quad\quad$ Remove every element of $\mathcal{X}$ with prob. $1/2$
17: $\quad\quad N_i \leftarrow$ Poisson$(t \cdot p/2)$ and $p \leftarrow p/2$
18: $\quad S \leftarrow$ GenerateSamples(Polytope$(c_i), N_i, b)$
19: $\quad \mathcal{X}$.Append(S)
20: 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 Polytope(c_i). We can show that $\bigcup_{i=1}^{m}$ Polytope$(c_i) =$ 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, 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 Polytope(c_i) sequentially. For each cube, we first compute the (approximate) volume of its corresponding polytope using a polytope volume computation algorithm (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 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 Polytope(c_i) $\cap\, \mathbb{L}^n$. Finally, the algorithm outputs the final volume estimate $\frac{|\mathcal{X}|}{p}$.

3. 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, 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.

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. Such a study would simply showcase the abilty of ttc to handle constraints requiring high precision arithmetic but would not allow a more meaningful comparison with SharpSMT. 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 [4]: (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 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.

3.1. Performance of ttc

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

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

Table 1
Performance comparison on 1131 instances.

Solving Time Comparison. A performance evaluation of polyvest and ttc is depicted in Figure 1, 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.

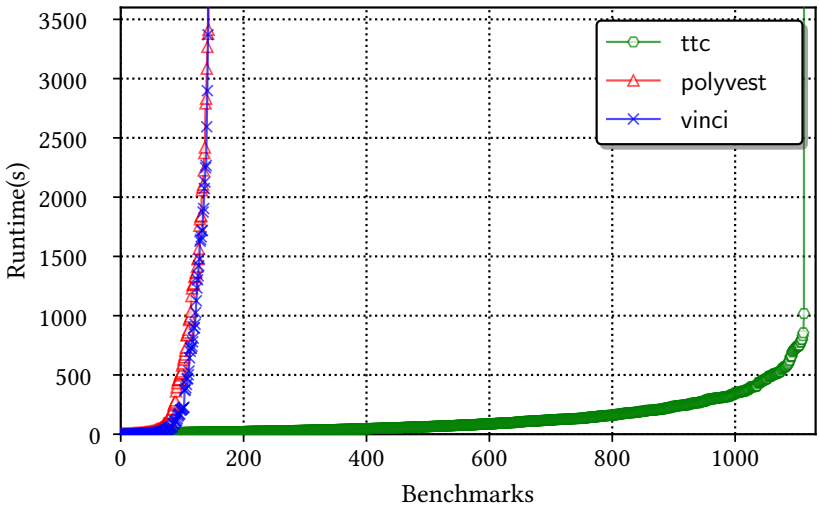

Figure 1: *Cactus plot comparing runtime of different tools.*

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 2, 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.

3.2. 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

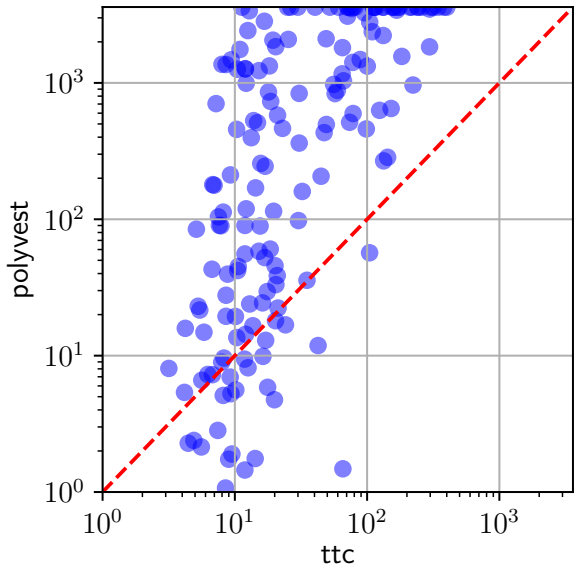

Figure 2: *Runtime comparison of* ttc *w.r.t.* polyvest.

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.

4. Conclusion and Future Work

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.

Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] A. Shaw, U. Sarkar, K. S. Meel, Efficient volume computation for smt formulas, in: Proceedings of Knowledge Representation and Reasoning (KR), 2025.
[2] A. Shaw, K. S. Meel, Model counting in the wild, in: Proc. of KR, 2024.
[3] M. Dyer, A. Frieze, R. Kannan, A random polynomial-time algorithm for approximating the volume of convex bodies, Journal of the ACM (JACM) 38 (1991) 1–17.
[4] B. Cousins, S. Vempala, A practical volume algorithm, Mathematical Programming Computation (2016).

[5] K. S. Meel, N. Vinodchandran, S. Chakraborty, Estimating the size of union of sets in streaming models, in: Proc. of PODS, 2021.
[6] I. Koley, S. Dey, D. Mukhopadhyay, S. Singh, L. Lokesh, S. V. Ghotgalkar, CAD Support for Security and Robustness Analysis of Safety-critical Automotive Software, ACM Transactions on Cyber-Physical Systems (2023).
[7] A. Cimatti, S. Mover, S. Tonetta, Smt-based verification of hybrid systems, in: Proc. of AAAI, 2012.
[8] A. Cimatti, A. Griggio, B. J. Schaafsma, R. Sebastiani, The mathsat5 SMT solver, in: Proc. of TACAS, 2013.
[9] G. Katz, C. Barrett, D. L. Dill, K. Julian, M. J. Kochenderfer, Reluplex: An efficient smt solver for verifying deep neural networks, in: Proc. of CAV, 2017.
[10] G. Girol, B. Farinier, S. Bardin, Not all bugs are created equal, but robust reachability can tell the difference, in: Proc. of CAV, 2021.
[11] S. Teuber, A. Weigl, Quantifying software reliability via model-counting, in: Proc. of QEST, 2021.
[12] T. Baluta, S. Shen, S. Shinde, K. S. Meel, P. Saxena, Quantitative verification of neural networks and its security applications, in: Proc. of CCS, 2019.