Theory Comput Syst (2015) 56:347–371
DOI 10.1007/s00224-014-9553-9

Estimating the Volume of Solution Space for Satisfiability Modulo Linear Real Arithmetic

**Min Zhou · Fei He · Xiaoyu Song · Shi He ·
Gangyi Chen · Ming Gu**

Received: 11 June 2013 / Accepted: 6 May 2014 / Published online: 13 June 2014

Abstract Satisfiability Modulo Theories techniques can check if a formula is satisfiable. In many cases, not only the qualitative judgment (satisfiable or not) but also the quantitative judgment (the dimension and size of the solution space) are of practical interest. For instance, the volume of path condition formula reflects the probability of the corresponding program path being taken. However, existing algorithms are not practical because they only work for small instances. Given a formula with Boolean structures, its volume is typically obtained by first decomposing it to a series of conjunctions (of linear constraints) with disjoint solution spaces and then accumulating the volume of each one. For the former step, we propose a BDD-based search algorithm which sharply reduces the number of conjunctions. For the latter one, we propose a Monte-Carlo integration with a ray-based sampling strategy, which

This work was supported by the Chinese National 973 Plan under grant No. 2010CB328003, the NSF of China under grants No. 11326070, 61272001, 60903030, 91218302, the Chinese National Key Technology R&D Program under grant No. SQ2012BAJY4052, the Tsinghua University Initiative Scientific Research Program, and the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions under grant No. YETP0167

M. Zhou (✉) · F. He · S. He · M. Gu
School of Software, Tsinghua University, Beijing 100084, China
e-mail: zhoumin03@gmail.com

M. Zhou · F. He · S. He · M. Gu
Key Laboratory for Information System Security, MOE, Shanghai, China

M. Zhou · F. He · S. He · M. Gu
Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China

X. Song
Electrical and Computer Engineering, Portland State University, Portland, OR 97207, USA

G. Chen
School of Mathematics and Information Science, Guangzhou University, Guangzhou, 510006, China

approximates the volume efficiently and accurately. Furthermore, degenerate solution spaces, which are not considered by other algorithms, could be handled properly by ours. Experimental results show that our method can handle formulas with up to 20 variables, which will cover many practical cases in software engineering

Keywords Satisfiability modulo theories · Linear arithmetic · Volume estimation · Monte-Carlo integration

1 Introduction

Satisfiability Modulo Theories (SMT) techniques [6] are getting increasingly popular in the field of software engineering. An SMT solver takes a formula given in a specific fragment of first order logic and returns a judgment telling whether the formula is satisfiable. A formula is satisfiable if and only if there exists at least one valuation under which the formula evaluates to true. In software engineering, SMT techniques are often used for software analysis and testing [19, 29] in which the path conditions are formulated as SMT formulas and their feasibility is checked by SMT solvers.

While the qualitative judgment of satisfiability is useful, the quantitative judgment of the solution space volume could be used to reveal more program properties [20]. In symbolic execution, the solution space volume of the path condition reflects the probability that the path is followed in execution. Such information could be used to detect hot/cold path in the program [9] and thus could be utilized to generate evenly distributed test data [30]. Furthermore, the volume of the branching conditions could be used for branch prediction, which provides important information for compiler optimization [5].

In this paper, we focus on the theory of linear real arithmetic, which is also the most commonly used background theory. To compute the volume of SMT formulas, the first challenge is to deal with the Boolean structure. Since an SMT formula may be an arbitrary Boolean combination of linear constraints, it is difficult to compute its volume directly. Typically, the formula is first decomposed to a series of conjunctions of linear constraints[1] with disjoint solution spaces. Then the volume of each conjunction is computed and summated to obtain the volume of the original formula. The decomposition scheme is not necessarily unique. Because each conjunction will be passed to the volume computation algorithm, which is time-consuming, the number of conjunctions will significantly affect the performance.

The bounded solution space of a conjunction of linear constraints is a polytope. When dealing with problems from software engineering, we can assume that solution spaces are bounded since primitive float values stored in computers are bounded. Computing the volume of a polytope is difficult in general. The complexity is #P-hard and no known algorithm can compute the exact volume of a polytope in polynomial time [15]. However, the results of applying randomized algorithms is somehow

[1] Since linear constraints are by default *conjuncted*, we use “linear constraints” short for “a conjunction of linear constraints” if no ambiguity is caused.

surprising. Polynomial time randomized algorithms are presented in [14, 18, 22, 23]. Notice that the solution space may be degenerate, i.e, the actual dimension may be strictly less than the number of variables. For instance, the solution space of $0 \leq x \leq y \wedge y \leq x \leq 1$ is a line segment in the plane. Existing volume computation algorithms do not consider the degenerate cases at all while such degenerate cases are indeed quite common in practice. Our algorithm can handle them properly.

In this paper, we address the volume computation problem for SMT formulas. We aim at designing an estimation algorithm that solves problem instances of practical interest. One possible application of our algorithm is to analyze the path conditions extracted from the control flow of computer programs. Statistics of Java libraries such as Apache Math, Colt, etc. has shown that over 99 % of conditional expressions involve less than 10 variables. With the help of variable substitution, many practical cases can be solved if we are able to solve formulas with up to 20 variables.

The contributions of this paper are:

- We propose a method for approximating the volume of polytopes based on Monte-Carlo integration. Our method uses a ray-based sampling strategy and the average relative error converges to 5 % quickly.
- Degenerate cases could be handled by our method. We transform the degenerate input problem to a compact form where the number of variables equals to the actual dimension. The volume of the original problem is then computed from that of the compact form.
- We propose a BDD-based search algorithm which reduces the number of calls to volume computation as well as the overall execution time.

This paper is organized as follows: Section 2 shows a motivating example. Section 3 explains the notations and terminologies. The proposed method is introduced in Section 4. Then experimental results are presented in Section 5. Related works are discussed in Section 6. Finally, conclusion and future works are given in Section 7.

2 Motivating Example

Figure 1 is a function that takes 3 parameters x, y and z. When it is invoked, x, y, z need to be sorted in ascending order before doing the real business. It is meaningful

```
function foo(double x, double y, double z) {
        if (y < x) {
                swap(x, y);
        }
        if (z < y) {
                swap(y, z);
        }
        if (y < x) {
                swap(x, y);
        }
        // the following operation requires: x <= y <= z
        doSomething(x, y, z);
}
```

Fig. 1 A motivating example

Table 1 The probability that each `swap` is executed

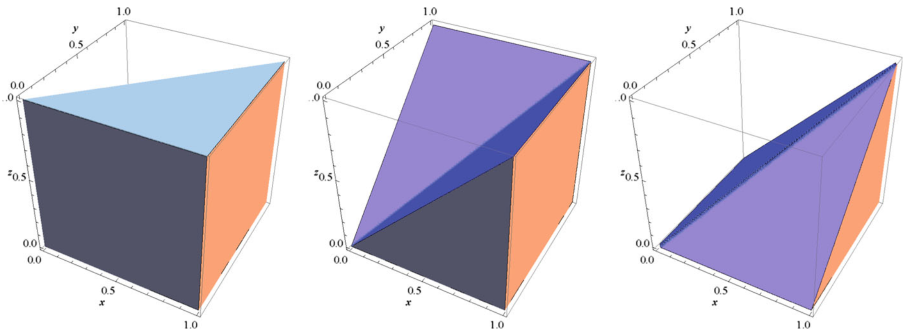
Line	Statistics	Path condition	Vol.	Fig.
3	0.5001445	$D_1 := y < x$	1/2	2a
6	0.6667387	$D_2 := z < x \vee z < y$	2/3	2b
9	0.3333531	$D_3 := (y < x \wedge z < y)$		
		$\vee(x < y \wedge z < x)$	1/3	2c

to know how frequently those `swap` functions will be executed. In software engineering, such information can be used for compiler optimization [9] or to select a set of evenly distributed test data in test data generation [30].

Since there is not any assumption on the distribution of x, y, z, we assume they are uniformly distributed over [0, 1]. Table 1 shows the frequency of each `swap` being executed after feeding 10^7 random inputs. On the other hand, the path conditions for each `swap` could be obtained by symbolic execution. For instance, the path condition of the first `swap` (line 3) is $D_1 := y < x$. The solution set for D_1 is shown in Fig. 2a and its volume is exactly $1/2$. For the second and third `swap` (line 6, 9), their path conditions are more complex. The solution space for D_3 is a polytope (with volume $1/3$) while that for D_2 is the union of two polytopes (with volume $2/3$). They are depicted in Fig. 2b and c respectively. As we could see, the statistics results are very close to the theoretical values $1/2$, $2/3$, $1/3$ respectively. It will be quite helpful if such volume can be computed mechanically. That is the motivation of this paper.

3 Preliminaries

In this work we focus on volume computation for SMT formulas in the theory of linear real arithmetic (denoted by T). It is one of the most commonly used theory



(a) Solution space for D_1 (b) Solution space for D_2 (c) Solution space for D_3

Fig. 2 Solution spaces for the path conditions

that can formulate linear operations and constraints over real variables. We assume that the solution space is bounded since real values are often encoded in the floating point format proposed by IEEE[2] and the range of real values are actually bounded.

Throughout this paper, Boolean values are denoted by $\{\mathsf{ff}, \mathsf{tt}\}$, Boolean variables are denoted by p, numeric variables are denoted by x, y and numeric constants are denoted by a, b. T-atoms are linear constraints in the form of $\sum_{i=1}^{n} a_i \cdot x_i \lhd b$ where $a_i \in \mathbb{R}$ are coefficients, x_i are free variables and $\lhd$ is one of the predicates $\{\leq, \neq\}$. The vector form is $a^\top x \lhd b$ where superscript $\cdot^\top$ means transpose of a vector or a matrix. An example of linear constraints is $x_1 + 2x_2 - x_3/2 \leq 1/2$. Notice that other predicates $\{=, \geq, >, <\}$ can be expressed using $\{\leq, \neq\}$. For instance $l = r$ is equivalent to $(l \leq r) \wedge (r \leq l)$ and $l < r$ is equivalent to $(l \leq r) \wedge (l \neq r)$. A T-formula f is a Boolean combination of linear constraints. Its propositional skeleton is a propositional formula f^p which has the same Boolean structure with f and each linear constraint is replaced with a propositional variable.

A valuation υ is a map from numeric variables to real values $\mathbb{R}$. Given a valuation υ, the truth value of f is denoted by $\upsilon(f)$. A propositional (partial) assignment α is a map from (a subset of) propositional variables to truth values and the truth value of f^p under α is denoted by $\alpha(f^p)$. In propositional logic, the truth value $\alpha(f^p)$ can be evaluated even if α is a partial function. For instance, $b_1 \wedge (b_2 \vee b_3)$ evaluates to ff whenever b_1 is ff. A partial assignment α can be refined to a set of T-atoms $\mathsf{ref}(\alpha) = \bigcup\{l_p \mid \alpha(p) = \mathsf{tt}\} \cup \bigcup\{\neg l_p \mid \alpha(p) = \mathsf{ff}\}$ where l_p is the linear constraint corresponding to p. $\mathsf{ref}\alpha$ is also called the refinement of α.

A T-formula f is T-satisfiable if and only if there is at least one valuation υ such that $\upsilon(f) = \mathsf{tt}$, denoted by $\upsilon \models_T f$. A propositional formula f^p is satisfiable if and only if there is at least one assignment α such that $\alpha(f^p) = \mathsf{tt}$, denoted by $\alpha \models f^p$. A partial assignment α is T-consistent if $ref(\alpha)$ is T-satisfiable. The solution space of f is denoted by $[\![f]\!]$. The volume of f is denoted by $\mathsf{vol}f$.

We use B_n to denote the n-dimensional unit ball in $\mathbb{R}^n$. Specially, $r\mathsf{B}_n$ is the unit ball scaled by the factor r (the radius).

4 The Proposed Method

Since a T-formula may have arbitrary Boolean structures, it is impossible to compute its volume directly. But the volume of a set of linear constraints can be computed. The first two subsections focus on volume computation for a conjunction of linear constraints. The revised simplex method is used to check the feasibility of the constraints and a randomized algorithm is proposed to compute the volume. The third subsection discusses the dimension issue and our solution. We extend the algorithm to formulas with Boolean structures in the last subsection.

[2] http://en.wikipedia.org/wiki/IEEE_floating_point

Table 2 Initial

	x	y	b
s_1^+	1	-1	0
s_2^+	-1	-1	1
s_3^+	0	-1	1/2
s_4^+	0	1	$-1/4$

4.1 Checking T-satisfiability of Linear Constraints

The decision procedure that can check the T-satisfiability of a conjunction of T-atoms is called the T-solver. For the theory of linear real arithmetic, the T-solver is typically based on the well-known simplex algorithm. To integrate it into an SMT framework, we use a revised simplex method (RSM) which solves a stack of T literals incrementally. Thus the T-consistency checking is time efficient. Since reference [12] already contains a basic description of RSM, we only explain the incremental part which is specific in our work.

As a matter of fact, RSM also provides extra interfaces for implicit equation detection (checking whether a constraint is an implicit equation) and dimension computation (computing the dimension of the current solution space). Those details are introduced later in the corresponding subsections.

4.1.1 Asserting a Linear Constraint

The input of RSM consists of two parts: (1) a set P^+ of variables that are restricted to be nonnegative and (2) a set of linear constraints in the form of $a_i^\top x = b_i$ where $a_i^\top x$ is a linear combination and b_i is a constant. Constraints in other forms are preprocessed by the following steps:

E1 For constraints in form of $a_i{}^\top x \leq b_i$, it is rewritten to $s^+ = b_i - a_i{}^\top x$ and then the slack variable s^+ is added to P^+.

E2 The constraint $a_i{}^\top x \neq b_i$ is rewritten to $s = a_i{}^\top x - b_i$ and we check whether $s = 0$ is entailed.[3]

The preprocessed linear constraints are expressed as a tableau. Asserting a linear constraint will add a row to the tableau. Take $\{x \geq y, x + y \leq 1, y \leq 1/2, y \geq 1/4\}$ for example. It is first rewritten to $\{s_1^+ = x - y, s_2^+ = 1 - x - y, s_3^+ = 1/2 - y, s_4^+ = y - 1/4\}$ (the restricted variables are $P^+ = \{s_1^+, s_2^+, s_3^+, s_4^+\}$). Then these 4 linear constraints are incrementally asserted, i.e., pushed to the stack and inserted to the tableau. Then the tableau becomes like Table 2. Its T-consistency is checked through a series of pivot operations (as shown in Tables 3 and 4). One feasible solution is $x = y = 1/4$ and the entire solution space is the triangle shown in Fig. 5a.

[3] This is done by checking whether the maximal and minimal value of s are both 0. The technique is similar to that used for implicit equation detection, which will be introduced later in Section 4.3.

Table 3 Pivot(y, s_4)

	x	s_4^+	b
s_1^+	1	-1	$-1/4$
s_2^+	-1	-1	3/4
s_3^+	0	-1	1/4
y	0	1	1/4

4.1.2 Canceling a Linear Constraint

When canceling a linear constraint, all its effects should also be canceled so that the solution space could be restored. In an incremental RSM, linear constraints are maintained as a stack and only the last constraint could be canceled. Thanks to the introduction of slack variables, each constraint could be canceled easily in the following way:

1. Pivot the corresponding slack variable to row.
2. Remove the entire row.

The intuition is that while a variable is in row, its coefficients in others rows are all 0. When this variable is a slack variable, which is specific to a linear constraint, it implies that the constraint is not added to any other row. Although this procedure is quite simple, it is guaranteed that the solution space is restored exactly to that before adding the constraint.

Take the above example, to cancel the last linear constraint $y \geq 1/4$ (slack variable s_4), the table is first pivoted to Table 5 and then removed the corresponding row as in Table 6. Canceling the $y \leq 1/2$ (slack variable s_3) is easier because s_3 is already in the row and we just have to remove the entire row, as shown in Table 7. The remaining tableau corresponds to the first two constraints $\{x \geq y, x + y \leq 1\}$ and the one feasible solution is $x = y = 0$.

4.2 RVC: Ray-based Volume Computation

Once the linear constraints are satisfiable, its volume can be computed. We focus on conjunctions of linear constraints at first. Boolean combination are considered later in this section.

Table 4 Pivot(x, s_1)

	s_1^+	s_4^+	b
x	1	1	1/4
s_2^+	-1	-2	1/2
s_3^+	0	-1	1/4
y	0	1	1/4

Table 5 Pivot(s_4, y)

	s_1^+	y	b
x	1	1	0
s_2^+	−1	−2	1
s_3^+	0	−1	1/2
s_4^+	0	1	−1/4

This subsection presents a randomized volume computation algorithm based on Monte-Carlo integration. We mention the fact that there are already randomized algorithms for volume computation [14, 18, 22, 23]. The theoretical time complexity is polynomial. The absence of implementation is somehow a surprise. So we implemented the algorithm proposed in [18] but found that it takes too long to solve even simple cases with only 3 variables. Thus we tend to believe that there is still distance between theoretical result and practical implementations.

The firm obstacle for using randomized algorithms is called "the curse of dimensionality[4]". When dimension increases, the probability of sampling inside the solution space decreases sharply. Assume we are estimating the volume of B_n by sampling uniformly over $x \in [-1, 1]^n$. The probability that $\mathbf{Pr}\{x \in \mathsf{n}\}$ decreases dramatically from 0.0025 to 8.9×10^{-8} as n increases from 10 to 20. Under such circumstances, it is almost not possible to sample enough points inside B_n in reasonable time, let alone computing its volume. The ray-based sampling strategy used in this paper may solve the problem to some extent.

Let C be the set of linear constraints, each in the form of ${a_i}^\top x \le b_i$ or ${a_i}^\top x \ne b_i$. The following lemma holds:

Lemma 1 *If C' is obtained by removing constraints in the form of ${a_i}^\top x \ne b_i$ from C to C is T-consistent, then* $\mathsf{dim}(C) = \mathsf{dim}(C')$ *and* $\mathsf{vol}(C) = \mathsf{vol}(C')$.

Rigorous prrof for this lemma is omitted but the intuition behind the proof is explained. Because $\llbracket C \rrbracket = \llbracket C' \rrbracket \backslash \llbracket a_i^\top x = b_i \rrbracket$, if $\llbracket C \rrbracket$ is nonempty, the removed part must have lower dimension than $\llbracket C' \rrbracket$. Excluding a lower dimensional subset does not change its dimension and volume.

Lemma 1 assures that we do not have to take into account those constraints in the form of ${a_i}^\top x \ne b_i$. Therefore, we assume the input is purely inequalities in the form of ${a_i}^\top x \le b_i$. The matrix form is $Ax \le b$, denoted by K for short. For simplicity, we assume that $\llbracket K \rrbracket$ is full-dimensional in this subsection, i.e, $\mathsf{dim}(K) = n$ where n is the number of variables. Techniques for handling degenerate solution spaces will be introduced in the next subsection.

Our algorithm is based on Monte-Carlo integration. Intuitively, the volume of K can be approximated directly by randomly sampling points in a n-dimensional cube that contains K. But the approximation could be very inaccurate because the number of sample points inside K will become very small when n increases. Experimental

[4] http://en.wikipedia.org/wiki/Curse_of_dimensionality

Table 6 Cancel s_4^+

	s_1^+	y	b
x	1	1	0
s_2^+	−1	−2	1
s_3^+	0	−1	1/2

results that support this statement are reported in Section 5. Therefore, we propose a ray-based approach for volume approximation. Different from direct Monte-Carlo methods, we make sure that every sample point is located inside K. Each sample defines a direction and we calculate the length from the origin point to K along the direction. Then the volume is approximated by computing the average value of each sample. Despite the exponential time complexity, our approach works well for many practical cases. The proposed method consists of two phases: rounding and sampling.

4.2.1 Rounding

Before sampling, the solution space K is rounded, i.e., it is affinely transformed to another space K' such that $\mathsf{B}_n \subseteq K' \subseteq (n+1)\mathsf{B}_n$. Intuitively, sampling in K' is easier and more efficient.

In this subsection, A is an $n \times n$ positive definite matrix and a is an $n \times 1$ vector. $E(A, a)$ is the ellipsoid $\{x \in \mathbb{R}^n \mid (x-a)^\top A^{-1}(x-a) \leq 1\}$ and $\delta E(A, a) := \{x \in \mathbb{R}^n \mid (x-a)^\top A^{-1}(x-a) \leq \delta^2\}$ is $E(A, a)$ scaled by a factor of δ. Technically, assume K is a full-dimensional, nonempty and bounded polytope. Theorem (3.1.9) in [16] assures that there exists an ellipsoid $E(A, a)$ such that $\frac{1}{n}E(A, a) \subseteq K \subseteq E(A, a)$. Such ellipsoid is called the Löwner-John Ellipsoid (LJE [16]) for K. In general, it is hard to compute exactly the LJE for a polytope K. Thus we relax the constant from $\frac{1}{n}$ to $\frac{1}{n+1}$ and compute it iteratively using the shallow-cut ellipsoid method [16]. Once an approximated LJE is found, an affine transformation which transforms K to K' is also obtained.

The rounding algorithm is shown in Algorithm 1. It terminates when the space K is properly rounded or $\mathsf{vol}(K)$ is found to be less than a small constant ϵ. Throughout the algorithm, it holds that $K \subseteq E(A, a)$. Thus $\mathsf{vol}(K) \leq \mathsf{vol}(E(A, a))$. In each iteration, we check whether the current ellipsoid has volume smaller than ϵ or $E(A, a)$ is already a good candidate. In the former case, the algorithm terminates and the volume of K is reported to be 0. In the latter case, there is one linear constraint $c^\top x \leq \gamma$

Table 7 Cancel s_3^+

	s_1^+	y	b
x	1	1	0
s_2^+	−1	−2	1

Algorithm 1 Rounding

```
Input: The solution space $K$, a positive constant number $\epsilon$
Output: A rounded polytope $K'$ such that $\mathsf{B}_n \subseteq K' \subseteq (n+1)\mathsf{B}_n$
$E(A,a) \leftarrow$ an ellipsoid s.t. $a \in K \subseteq E(A,a)$;
while $\mathsf{vol}(E(A,a)) \geq \epsilon \wedge (\frac{1}{n+1}E(A,a) \nsubseteq K)$ do
    $(c,\gamma) \leftarrow$ a constraint in $K$ such that $(\frac{1}{n+1}E(A,a) \nsubseteq [\![c^\top x \leq \gamma]\!])$;
    $E(A,a) \leftarrow \mathsf{sce}(E(A,a) \cap [\![c^\top x \leq c^\top a + \frac{1}{n+1}\sqrt{c^\top Ac}]\!])$;
end
if $\mathsf{vol}(E(A,a)) \geq \epsilon$ then
    // assume $A^{-1} = Q^\top Q$
    return $K' = \{x \in K \mid (n+1) \cdot Q \cdot (x-a)\}$;
else
    Exit with $\mathsf{vol}(K) = 0$;
end
```

which does not contain the whole ellipsoid $\frac{1}{n+1}E(A,a)$. In the shallow-cut ellipsoid method, a minimum volume ellipsoid that contains

$$E(A,a) \cap \left[\!\!\left[c^\top x \leq c^\top a + \frac{1}{n+1}\sqrt{c^\top Ac} \right]\!\!\right]$$

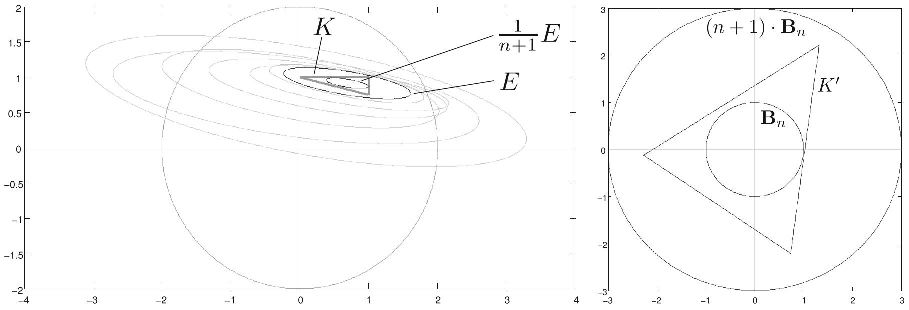
is computed by the formula

$$\begin{cases} a' = a - \rho \frac{Ac}{\sqrt{c^\top Ac}} \\ A' = \zeta \cdot \sigma \left(A - \tau \frac{Acc^\top A}{c^\top Ac} \right) \end{cases}$$

$$\rho = \frac{1}{(n+1)^2}, \quad \sigma = \frac{n^3(n+2)}{(n^2-1)(n+1)^2}, \quad \tau = \frac{2}{n(n+1)}, \quad \zeta = \frac{1}{2n^2(n+1)^2}$$

which is done in the function sce. Then the next iteration could start. The whole algorithm is guaranteed to terminate in polynomial time (Theorem (3.3.9) in [16]). If a candidate ellipsoid $E(A,a)$ is found, the matrix A must be a positive definite matrix (so is A^{-1}). Thus there is some Q such that $A^{-1} = Q^\top Q$. An affine transformation (T_0, t_0) with $T_0 = (n+1)Q$ and $t_0 = -(n+1)Qa$ could transform K to a position such that $\mathsf{B}_n \subseteq K' \subseteq (n+1)\mathsf{B}_n$. More importantly, $\mathsf{vol}(K) = \frac{1}{|\det(T_0)|}\mathsf{vol}(K') = \frac{1}{(n+1)^n\sqrt{\det A^{-1}}}\mathsf{vol}(K')$. Then the original problem is reduced to computing the volume of K'.

An example is shown in Fig. 3a where K is the triangle $\triangle ABC$. The candidate ellipsoid is plotted every 20 iterations and the last candidate $E(A,a)$ along with $\frac{1}{n+1}E(A,a)$ is plotted in black. Since $\frac{1}{n+1}E \subseteq K \subseteq E$, an affine transformation is obtained and K is transformed to K' as shown in Fig. 3b. Compared with K, K' is at a place better for sampling.



(a) Rounding steps (b) Rounding result

Fig. 3 Rounding example

4.2.2 Sampling

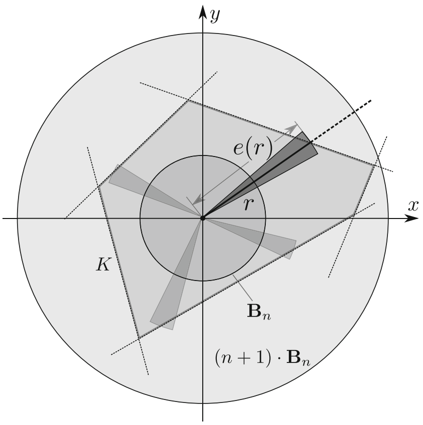
The rounded polytope is still denoted by K. Since $\mathsf{B}_n \subseteq K$, the origin and the n-dimensional unit ball are both contained in K. Thus the volume of K can be obtained by the following integration:

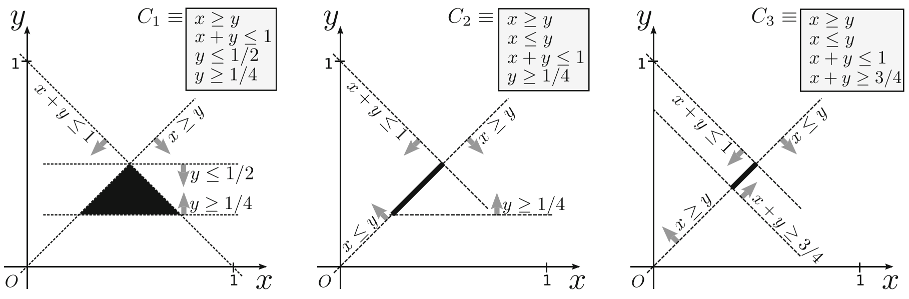
$$\mathsf{vol}(K) = \int_{\mathsf{S}^{(n-1)}} V_n(e(r))\mathbf{d}r \tag{1}$$

where $\mathsf{S}^{(n-1)}$ is the $(n-1)$-sphere (the surface of the n-dimensional unit ball), $V_n(u) = \mathsf{vol}(u\mathsf{B}_n) = u^n \cdot \mathsf{vol}(\mathsf{B}_n)$ is the volume of a n-dimensional ball with radius u and $e(r) = \sup\{\delta > 0 \mid \delta \cdot r \in K\}$ is the maximum length of line segment along the direction r. Since $\mathsf{B}_n \subseteq K \subseteq (n+1)\mathsf{B}_n$, it always holds that $1 \le e(r) \le (n+1)$.

The idea (for 2 dimension) is depicted in Fig. 4. The integration is actually summating the volume of small sectors. The n-dimensional variable r is sampled from the S^{n-1} and $e(r)$ is the length from the origin to K along the direction of r. Then $V_n(e(r))$ is the volume of n-dimensional ball with radius $e(r)$ and $\mathsf{vol}(K)$ is obtained by averaging such volume.

Fig. 4 The idea of sampling




$C_1 \equiv$ $x \geq y$, $x + y \leq 1$, $y \leq 1/2$, $y \geq 1/4$
$C_2 \equiv$ $x \geq y$, $x \leq y$, $x + y \leq 1$, $y \geq 1/4$
$C_3 \equiv$ $x \geq y$, $x \leq y$, $x + y \leq 1$, $x + y \geq 3/4$
$x + y \leq 1$
$x \geq y$
$x \leq y$
$y \leq 1/2$
$y \geq 1/4$
$x + y \geq 3/4$
y
x
O
1

(a) Solution for C_1 (b) Solution for C_2 (c) Solution for C_3

Fig. 5 Examples of full-dimensional and degenerate solution spaces

Monte-Carlo integration is used to estimate the integration in Eq. (1). Algorithm 2 shows the detailed steps. Let $\overline{\mathsf{vol}}(K)$ be the approximation for $\mathsf{vol}(K)$. A n-dimensional vector r is first sampled uniformly on S^{n-1}, and then $e_r = e(r)$ is computed. The approximation for the integration is the average value of those $V_n(e_r)$. By the law of large numbers, $\overline{\mathsf{vol}}(K)$ converges to $\mathsf{vol}(K)$.

4.3 The Dimension Issue

Constraints over the same set of variable may have solution spaces in various dimensions. For example, Fig. 5a, b and c show three cases where each has 4 linear constraints over 2 variables. The solution space for C_1 is a triangle (2-dimension)

Algorithm 2 Volume approximation

```
Input: $\mathsf{B}_n \subseteq K \subseteq \mathsf{B}_n(n+1)$, the maximum number of iterations $M$
Output: Approximation of $\mathsf{vol}(K)$
$\sigma \leftarrow 0$ ;                                   // the approximation
$v \leftarrow \mathsf{vol}(\mathsf{B}_n)$ ;              // the volume of $n$-dimensional unit ball
for $i \leftarrow 1$ to $M$ do
    $r \leftarrow \mathsf{sample}_{\mathsf{S}^{(n-1)}}()$;
    $e_r \leftarrow +\infty$;
    foreach constraint $a_i{}^\top x \leq b_i$ of $K$ do
        // it must hold $b_i > 0$ because the origin is in $K$
        if $a_i{}^\top r > 0$ then
            $e_r \leftarrow \min\{b_i/(a_i{}^\top r), e_r\}$;
        end
    end
    // $e_r$ must be finite because $K$ is bounded
    $\sigma \leftarrow \sigma + e_r^n \cdot v/M$;
end
return $\sigma$;
```

while those for C_2 and C_3 degenerate to line segments (1-dimension). In principle, a solution space with higher dimension is strictly larger than that with lower dimension. If they are extracted from two path conditions, then the path with higher dimensional solution space is likely to be reached more often than that with the lower one (while assuming the inputs are uniformly distributed). Quantitative comparison of volume only makes sense for two solution spaces of the same dimension. For instance, $[\![C_1]\!]$ is larger than both $[\![C_2]\!]$ and $[\![C_3]\!]$. Moreover, $[\![C_2]\!]$ is larger $[\![C_3]\!]$. Degenerate solution spaces are quite common in practice. For instance, the solution space of an equality `x==y` is degenerate. This issue is discussed in this section.

Randomized methods rely on sampling in a full-dimensional polytope because the probability of sampling inside degenerate solution spaces is 0. In our method, the input linear constraints are first transformed to its compact form which has the same solution space with the original problem. The support part of the compact form is full-dimensional and the volume of the original problem can be calculated from that of the support part.

Definition 1 (**Dimension**) The dimension, $\mathsf{dim}(C)$, for a polytope $C \subseteq \mathbb{R}^n$ is the number of affinely independent points in C minus 1.

While the definition is mathematical, the concept of dimension is intuitive. By definition, the dimension can be -1 (if C is empty), 0 (when C consists of a single point), 1 (when C is a line segment), and up to n when C is in R^n. In the last case, we say that C is *full-dimensional* otherwise it is *degenerate*. For instance, a cube in R^3 is full-dimensional and a line segment in R^3 is degenerate.

Definition 2 (**Implicit equation**) Let ${a_i}^\top x \le b_i$ be one constraint from $Ax \le b$. We say ${a_i}^\top x \le b_i$ is an implicit equation (with respect to $Ax \le b$) if and only if $Ax \le b$ entails $a_i^\top x = b_i$.

For instance, $y \le x$ and $x \le y$ in C_2 are both implicit equations. Detection of implicit equations can be done by maximizing the slack variable corresponding to the equation. Assume s_i^+ is the slack variable corresponding to ${a_i}^\top x \le b_i$. Once a feasible solution is found, a new row that encodes the objective function $f = s_i^+$ is introduced to the tableau. Then we check if it is possible to increase the value of f while preserving feasibility (similar to the procedure for picking a column variable in [12]). ${a_i}^\top x \le b_i$ is an implicit equation if and only if the maximal value of s_i^+ is 0.

Definition 3 (**Compact form**) Given an input problem $Ax \le b$, its compact form (which has the same solution set as $Ax \le b$) is

$$\begin{cases} \quad x_p = A_p x_s + b_p & \text{projection part} \\ A_s x_s \le b_s & \text{support part} \end{cases}$$

where the variable set x is partitioned to $x_p \cup x_s$ and the support part $A_s x_s \le b_s$ is full-dimensional.

Given $Ax \leq b$, its compact form can be obtained by the following steps: (1) Maximize each slack variable in P^+ to find all implicit equations. (2) Rewrite all implicit equations to equations. (3) Apply Gaussian elimination on the set of equations and remove those trivial equations in the form of $0 = 0$. Then the equations are in reduced row echolon form [5]. (4) Choose each variable corresponding to the leading coefficients of each row as x_p. The equations are in the form of $x_p = A_p x_s + b_p$. (5) Eliminate variables in x_p from the inequalities by substitution and remove trivial inequalities in the form of $c_l \leq c_r$ where c_l, c_r are constants and the value of c_l is less than c_r to obtain the support part $A_s x_s \leq b_s$. Specially, if $Ax \leq b$ is already full-dimensional, its compact form is just $Ax \leq b$ itself and the projection part is empty.

Theorem 1 (**Actual dimension**) *Given C a group of T-consistent linear constraints and $(x_p = A_p x_s + b_p) \wedge (A_s x_s \leq b_s)$ its compact form, then* $\mathsf{dim}(C)$ *equals to the dimension of the vector x_s.*

For instance, C_2 in Fig. 5b has 4 constraints and two of them are implicit equations $(\{x \geq y, x \leq y\})$. Its compact form is:

$$\begin{cases} \qquad y = x & \text{projection part} \\ \begin{bmatrix} -1 \\ 1 \end{bmatrix} x \leq \begin{bmatrix} 1/4 \\ 1/2 \end{bmatrix} & \text{support part} \end{cases} \tag{2}$$

which contains 1 variable in the support part. Thus its dimension is 1.

The volume of the support part can be approximated using the method introduced above. The correspondence between the volume of $Ax \leq b$ and its support part is formulated by the following theorem.

Theorem 2 (**Volume**) *Given a group of T-consistent linear constraints $Ax \leq b$, let $(x_p = A_p x_s + b_p) \wedge (A_s x_s \leq b_s)$ be its compact form. The following equation holds:*

$$\mathsf{vol}(Ax \leq b) = \sqrt{\det(I + {A_p}^\top A_p)} \cdot \mathsf{vol}(A_s x_s \leq b_s)$$

Proof Without loss of generality, we assume $b_p = 0$ (i.e., $x_p = A_p x_s$) because removing the offset does not change the volume.

Assume the dimension of x_s and x_p are n_s and n_p respectively. Thus A_p is a $n_p \times n_s$ matrix. The support part is actually a polytope $K_0 := A_s x_s \leq b_s$ in the n_s dimensional subspace and the actual polytope $K := Ax \leq b$ is in $n_s + n_p$ dimension due to the projection part. The idea of proof is that there exists an orthogonal matrix M such that M can rotate K to K' which is in the same subspace of the K_0 and $\mathsf{vol}(K') = \mathsf{vol}(K)$. Furthermore, the relation between K' and K_0 is established by a linear transformation thus their volumes are related by the determinant of the transformation matrix.

[5] http://en.wikipedia.org/wiki/Gaussian_elimination

First of all, there is some orthogonal matrix M which rotates K to KK'. Assume that

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}$$

it should satisfy $(0, {x'_s}^\top)^\top = M \cdot ({x_p}^\top, {x_s}^\top)^\top$ where $(0, {x'_s}^\top)^\top$ is the coordinate of a point in K'. Actually, K' is obtained from K_0 by a linear transformation and the relation between x'_s and x_s is:

$$x'_s = M_{21}x_p + M_{22}x_s = (M_{21}A_p + M_{22})x_s$$

Let $Q = (M_{21}A_p + M_{22})$ be the transformation matrix.

Since M is orthogonal, it should hold that $||({x_p}^\top, {x_s}^\top)|| = ||({x'_p}^\top, 0)||$, i.e:

$$\begin{aligned} {x_p}^\top x_p + {x_s}^\top x_s &= {x'_p}^\top x'_p \\ \iff (A_p x_s)^\top A_p x_s + {x_s}^\top x_s &= (Qx_s)^\top Q x_s \\ \iff {x_s}^\top (I + {A_p}^\top A_p) x_s &= {x_s}^\top Q^\top Q x_s \end{aligned}$$

The above equation holds for all x_s, therefore $I + {A_p}^\top A_p = Q^\top Q$. Then the import relation holds that $(\det(Q))^2 = \det(I + A_p^\top A_p)$. The conclusion of this theorem can be proved:

$$\mathsf{vol}(K') = \mathsf{vol}(Q(K_0)) = |\det(Q)| \cdot \mathsf{vol}(K_0) = \sqrt{\det(I + {A_p}^\top A_p)} \cdot \mathsf{vol}(K_0)$$

□

In case that the projection part is empty, the determinant is considered to be 1. To avoid diving into the technical details, we take C_2 in Fig. 5b as an example. Its compact form is shown in Eq. (2). The support part is essentially $1/4 \le x \le 1/2$ which has the volume (length) of 1/4. A_p is a 1×1 matrix $[1]_{1\times 1}$ and $A_p^\top A_p = [1]_{1\times 1}$. Therefore, $\mathsf{vol} C_2 = \sqrt{2} \cdot (1/4) = \sqrt{2}/4$, which is consistent with our intuition.

Given a set of constraints, its compact form is not necessarily unique. However, the volume is unique no matter which one is used in computation.

4.4 Enumerating Partial Models

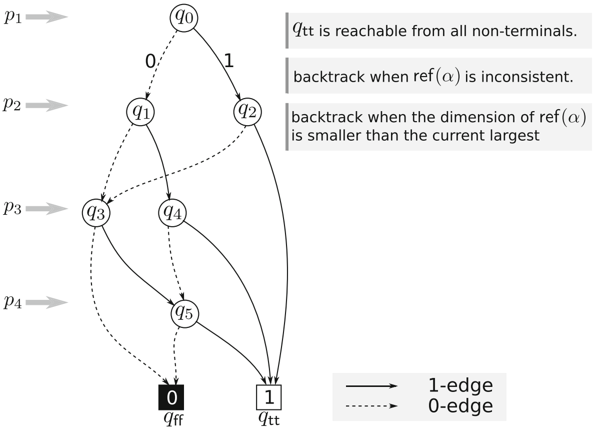
Given a T-formula f, its propositional skeleton f^p can be obtained by replacing each atom in T with a propositional variable. Then the computation for $\mathsf{vol} f$ is decomposed to computing the volume of a series of conjunctions of linear constraints. If a partial assignment α satisfies that $\alpha \models f^p$ and α is T-consistent ($\mathsf{ref}\alpha$ is T-satisfiable), we call α a (propositional) partial model. Given A a set of partial models, we call A a partition for f if and only if $[\![f]\!] = \cup_{\alpha \in A} [\![\mathsf{ref}(\alpha)]\!]$ and $[\![\mathsf{ref}(\alpha_i)]\!] \cap [\![\mathsf{ref}(\alpha_j)]\!] = \emptyset$ for all $\alpha_i, \alpha_j \in A$ and $\alpha_i \neq \alpha_j$, i.e, the entire solution space of f is partitioned to disjoint regions in A and its volume is computed by:

$$\mathsf{vol}(f) = \sum_{\alpha \in A} \mathsf{vol}(\mathsf{ref}(\alpha))$$

It is known that volume computation is relatively time consuming, thus the size of A dominates the algorithm performance. Usually, the partition A is obtained by an extended DPLL(T) algorithm [28] which is originally designed for finding one partial model. The intuition is that: when a partial model α is found, it is excluded from the solution space by replacing f with $f \wedge \neg\alpha$ and the search continues until no more partial model could be found. Reference [24] is based on this idea and it further reduces the number of calls to volume computation by minimizing each partial model and incorporating with theory learning. Their algorithm is much better than the straightforward approach for the formulas given in Conjuntion Normal Form (CNF).

In this work, we use Binary Decision Diagram [1] (BDD) to facilitate the propositional model finding. BDD is a directed acyclic graph that encodes the propositional formula f^p. Each BDD has a root q_0 and two terminal nodes $\{q_{\mathsf{tt}}, q_{\mathsf{ff}}\}$. Each non-terminal node q is associated with a decision variable $\mathsf{var}(q)$ and two edges labeled with $\{0, 1\}$ respectively. The 0-edge corresponds to assigning ff to $\mathsf{var}(q)$ and the 1-edge corresponds to assigning tt to $\mathsf{var}(q)$. Each route from the root node q_0 to q_{tt} corresponds to a partial assignment that satisfies f^p. Although constructing the BDD for f^p requires exponential time in the worst case, it is still cheaper than volume computation. Then propositional partial models can be enumerated by searching on the BDD. We incorporate an incremental T-solver which further filters out those T-inconsistent partial assignments. Using BDD, it is not necessary to introduce extra proxy variables when transforming the formula into CNF, thus the number of partial assignments is reduced. Moreover, the dimension information is also used to prune the search space. Experimental results show that our method largely reduces the number of calls to RVC as well as the execution time.

The idea of our search algorithm is depicted in Fig. 6 and it is described as a Depth-First-Search (DFS) procedure in Algorithm 3. It is essentially a DFS procedure incorporated with theory-learning and backtracking. It starts with the root node q_0. $stack$ is used to maintain the context. Each element in $stack$ is a triple (q, a, e) where q is a node, a is the assignment that is associated with the incoming edge to q and e records whether q has been explored. The partial assignment α is maintained during the search and once the search reaches q_{tt}, the current assignment $\mathsf{ref}\alpha$ is checked for



q_{tt} is reachable from all non-terminals.

backtrack when $\mathsf{ref}(\alpha)$ is inconsistent.

backtrack when the dimension of $\mathsf{ref}(\alpha)$ is smaller than the current largest

Fig. 6 The idea of partial model enumeration

Algorithm 3 Partial model enumeration

```
Input: A $T$-formula $f$
Output: $\mathsf{vol}(f)$
$S \leftarrow 0$ ;                                   // the accumulated volume
$bdd \leftarrow \mathsf{construct}\ (f^p)$ ;             // construct the BDD
$stack \leftarrow [(\mathsf{root}(bdd), nil, \mathsf{ff})]$ ;    // the stack used in DFS
$\alpha \leftarrow []$ ;                              // the partial assignment
while $stack \neq []$ do
    $(q, a, e) \leftarrow \mathsf{pop}(stack)$;
    if $\neg e$ then                               /* $q$ is not explored */
        $\mathsf{push}(stack,\ (q, a, \mathsf{tt}))$ ;            // set $q$ as explored
        $\mathsf{push}(\alpha,\ a)$;
        if $q = q_{\mathsf{tt}} \vee q = q_{\mathsf{ff}}$ then            /* $q$ is terminal node */
            if $q = q_{\mathsf{tt}}$ then
                if $\mathsf{consistent}_T(\mathsf{ref}(\alpha))$ then
                    $S \leftarrow S + \mathsf{vol}(\mathsf{ref}(\alpha))$;
                else      /* backtrack and restore the context */
                    $level \leftarrow \mathsf{analyze}(\alpha)$;
                    $(stack, \alpha) \leftarrow \mathsf{backtrack}(level)$;
                end
            end
        else                  /* non-terminal, push the children */
            $\mathsf{push}(stack,\ (\mathsf{child}_1(q), \mathsf{var}(a), \mathsf{ff}))$;
            $\mathsf{push}(stack,\ (\mathsf{child}_0(q), \neg\mathsf{var}(a), \mathsf{ff}))$;
        end
    else                            /* pop $\alpha$ when $q$ is explored */
        $\mathsf{pop}(\alpha)$;
    end
end
return $S$;
```

T-consistency. Its volume is computed and accumulated if it is T-consistent. Otherwise, we analyze the conflict and backtrack to a proper level (the search context *stack* and α are modified accordingly). The backtracking is chronological and the backtrack level is the latest level that is T-consistent.

The search space is pruned in three ways: (1) q_{tt} is reachable from all non-terminal nodes, which is ensured by the BDD construction rule. This property actually assures that the search procedure will not waste time in propositional unsatisfiable assignments. (2) Once a propositional model α is found, it is checked for T-consistency. If $\mathsf{ref}\alpha$ is inconsistent, the search will be backtracked immediately to the conflict. We use an incremental T-solver so that search and backtrack could be done incrementally. (3) While α is satisfiable, its dimension is calculated. It is guaranteed that the deeper it goes during the search, the smaller dimension it has. If the dimension is smaller than the current discovered largest, we can backtrack immediately. With all the optimization strategies, our algorithm yields much smaller number of invocations to the T-solver and volume computation, which significantly reduces the execution time.

5 Implementation and Experimental Results

Our algorithm is implemented in Matlab (version 7.10) and tested on a PC (Intel E5300 Dual-Core 2.60GHz, 4G RAM, Ubuntu 10.10). Our tool is named RVC. It accepts T-formulas with arbitrary Boolean structures.

There are several implementation issues. The first is that the volume of B_n (denoted by V_n) is needed for computation. It is calculated efficiently by the recursion formula: $V_1 = 2$, $V_2 = \pi$ and $V_n = (2\pi/n) \cdot V_{n-2}$ for all $n > 2$ (see [2]). Another issue is to sample uniformly on $\mathsf{S}^{(n-1)}$. We follow the method proposed in [25]. Let $x = (x_1, x_2, \ldots, x_n)$ and each x_i be generated in $N(0, 1)$ (the standard normal distribution). The vector $x/|x|$ is then uniformly distributed on $\mathsf{S}^{(n-1)}$. The BDD implementation used in the experiments is CUDD[6].

The T-solver is based on an incremental implementation of the revised simplex method. I.e, it maintains a stack of linear constraints and stays online. The upper-level search algorithm could communicate with the T-solver during the search by pushing a new constraint, popping out one or checking the satisfiability of the current stack. The T-solver is efficient when re-checking the T-consistency after modifying the constraints set.

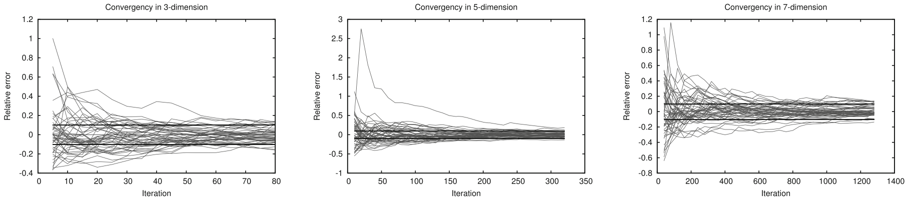
Evaluation of our method is done in the following manner: (1) First of all, RVC is experimented on a group of randomly generated conjunctions of linear constraints. The convergence and accuracy of RVC are shown and the time consumption is compared with the tool LRS[7] which is an implementation of the Lexicographic Reverse Search algorithm [4]. Although there are various exact algorithms [8], we choose LRS because it has the best performance when solving our randomly generated instances. Convergence and accuracy are not compared since the exact algorithm is designed to be accurate. For a fair comparison, all conjunctions are full-dimensional although RVC could handle cases with degenerate solution space. (2) Then we implement a direct Monte-Carlo method and compare its convergence and accuracy with RVC. Actually, we also implemented the polynomial time randomized algorithms in [18] but the implementation is slow[8] and thus could not be used for comparison. (3) Finally, the experimental results for solving formulas with arbitrary Boolean structures are presented.

Test cases used in the experiments are randomly generated. To generate a linear constraint, we randomly select a vector of coefficients and variable names and

[6] http://vlsi.colorado.edu/~fabio/CUDD/

[7] http://cgm.cs.mcgill.ca/~avis/C/lrs.html

[8] The reason is that their algorithm contains constants such as 801, 1600 in nested loops. It takes hundreds of seconds to solve a 3-dimensional case.



(a) 3 dimension)

(b) 5 dimension

(c) 7 dimension

Fig. 7 Convergence in lower dimensions

product them into a linear constraint. Various settings of the number of variables and constraints are given so that test cases of different scale are generated. For a Boolean combination of atomic formulas, the generation is done recursively. In each level, we have several options: (1) generate an atomic formula and return, (2) recursively generate a conjunction of sub-formulas, (3) recursively generate a disjunction of sub-formulas, (4) recursively generate a negation of a sub-formula. The probability of taking each option is given by a group of parameters. Various settings of these parameters are used in the experiments, so that our algorithm is tested under different scenarios. We intentionally select those cases with nonempty solution space because T-inconsistent cases are filtered out early at the consistency checking phase thus will not trigger the volume computation procedure. For each case, we make sure that the origin is contained in its solution space and we also put an n-cube to bound it. For a fair comparison, no other selection is applied.

The implementation (including the source code) and test cases are publicly available as a Google Code project[9].

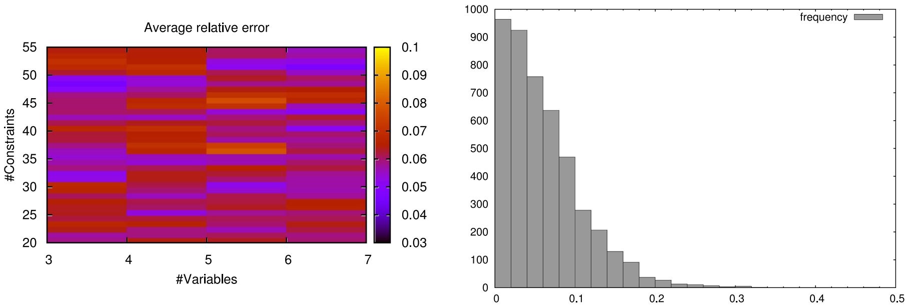
5.1 Performance of RVC

Figure 7a, b and c show convergence of RVC in the dimension of 3, 5 and 7. Each curve in the figure shows the change of relative error while the number of iteration grows. As we could see, the maximum relative error for each case reaches about 15 % quickly (the horizontal bars indicates ±10 %).

The temperature for relative errors are shown in Fig. 8a. The data for each scenario (fixing the number of variables and constraints) is averaged from 50 cases. As we can see, the relative error is below 7 % in most scenarios and no one exceeds 10 %. The histogram is shown in Fig. 8b and the average relative error over all cases is as small as about 5 %.

The accuracy data for higher dimensional cases are currently not available because the actual volume for higher dimensional solution spaces can not be computed by the exact algorithms. For instance, LRS takes hundreds of seconds to solve a single instance in 8 dimension or higher. On the other hand, RVC could handle those cases and the convergence situation is similar.

[9] http://code.google.com/p/rvc/



(a) Temperature of relative error (b) Histogram of relative error

Fig. 8 Experiments on the accuracy of RVC

5.2 Compare RVC with the Exact Algorithm

In the previous subsection, we found that the relative error gets quite small after $n \cdot 2^n$ iterations where n is the dimension (number of variables). In the rest of this section, we fix $M = n \cdot 2^n$ as the maximum number of iterations. Though it is exponential, this number is acceptable when n is as large as 20. Then the time consumption of RVC and LRS are compared in Fig. 9. In the maximal allowed runtime (10 minutes), LRS handles 7 variables while RVC could handle up to 18, which will cover many practical cases. Notice that the the vertical axis is in logarithmic scale.

5.3 Compare RVC with Direct Monte-Carlo Method

It is worthwhile to compare RVC with direct Monte-Carlo method. Given a test case after rounding, it is bounded in the n-dimensional cube $[-(n+1), (n+1)]^n$. The direct Monte-Carlo method is implemented in such a way that (1) it samples in the cube and counts the ratio of points locates inside K, say the ratio is z. (2) return

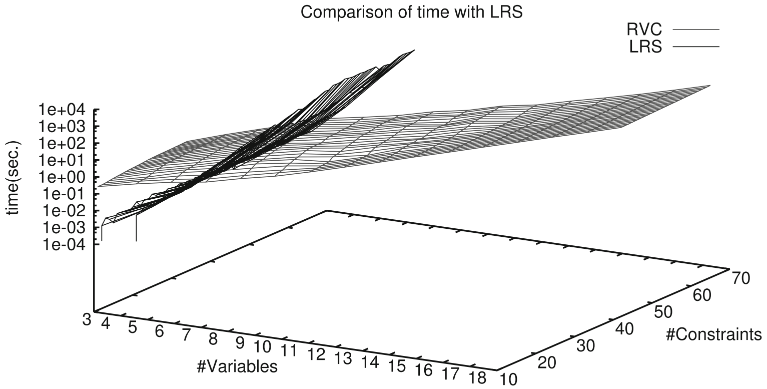
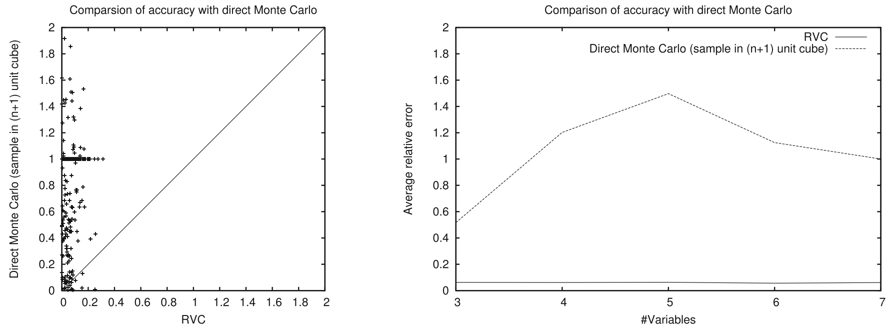


Fig. 9 Time consumption of RVC and LRS



(a) Accuracy for individual cases

(b) Average accuracy

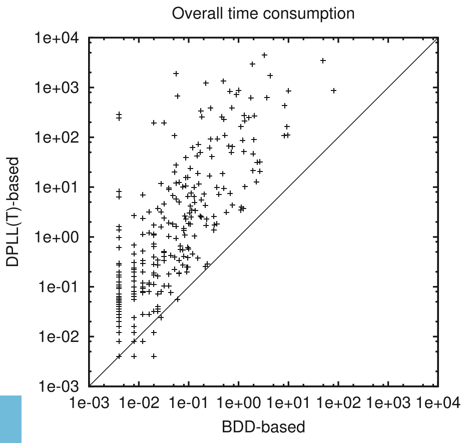
Fig. 10 Compare RVC with a direct Monte-Carlo method

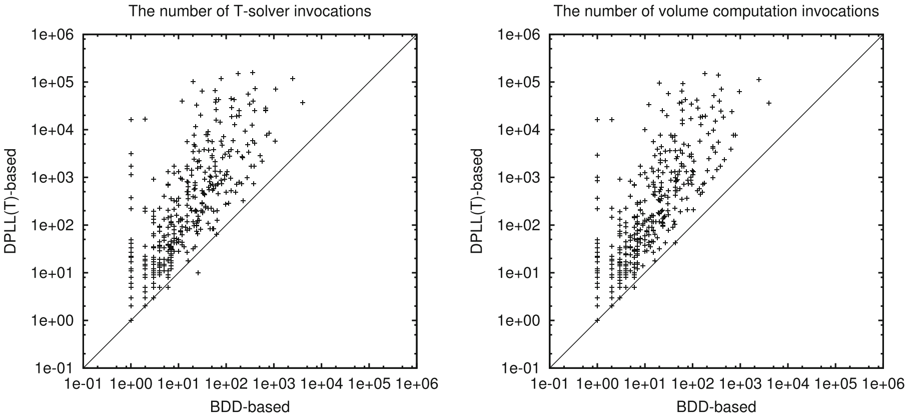
$\bar{\mathsf{vol}}K = z \cdot (2(n+1))^n$. The accuracy for individual cases is shown in Fig. 10a. It is obvious that RVC outperforms the direct Monte-Carlo method. Figure 10b shows the average accuracy for different dimensions. RVC is much better than the direct Monte-Carlo method. Moreover, when the dimension increases, the approximation returned by the direct Monte-Carlo method converges to 0 (relative error converges to 100 %) because there is no sample point that lies in the solution space. We also tried to sample in $(n+1)\mathsf{B}_n$ and the result is similar.

5.4 Handling T-formulas with Boolean Structures

RVC is also tested on formulas with logical connectors $\{\wedge, \vee, \neg\}$. To the best of our knowledge, the closest work is [24], which is based on DPLL(T). They aimed at minimizing the number of calls to volume computation and two methods were proposed (although there is not significant performance difference between them). In this part, we implement their framework and compare with them on the number of calls to the T-solver and the number of calls to the volume computation. Since [24]

Fig. 11 Runtime comparison with the DPLL(T)-based approach





(a) Invocations to the T-solver

(b) Invocations to RVC

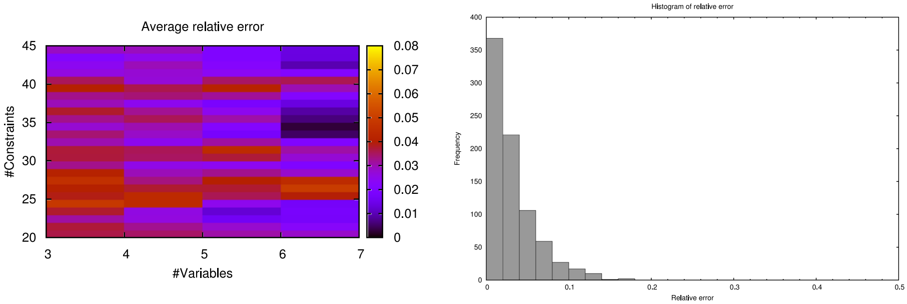
Fig. 12 Comparison with the DPLL(T)-based approach

is just a framework in which the volume computation algorithm is parameterized, we use our algorithm RVC for a fair comparison of time consumption.

Experimental results are shown in Figs. 11, 12a and b. Notice that all the figures are in logarithmic coordinates. The comparison of runtime is in Fig. 11 and the comparison of invocations to the T-solver and RVC are in Fig 12a and b respectively. Our approach (BDD-based) outperforms the DPLL(T)-based one on both the number of T-solver invocation and volume computation invocation. The overall time consumption of our approach is hundreds of times (about 832) smaller than the DPLL(T)-based one.

The accuracy for handling formulas with Boolean structures is shown in Fig. 13a and b. The result is even better than solving conjunctions of constraints. The average relative error is less than 5 %.

Degenerate solutions spaces are handled properly by RVC. The dimension information is also helpful to bound the search when handling formulas with Boolean structures.



(a) Temperature of relative error

(b) Histogram of relative error

Fig. 13 Experiments on the formulas with Boolean structure

Recall the motivating example at the beginning of this paper. RVC is used to compute the volume of the 3 path conditions. Each path condition is estimated for 10 times and the average volumes are 0.50, 0.67, 0.34 respectively, which is very close to the theoretical value $1/2$, $2/3$, $1/3$. Each invocation of RVC finished in about 0.5 second.

6 Related Works

The satisfiability of SMT formulas can be decided by the state-of-the-art solvers (Z3 [11], Yices [13], CVC3 [7], etc.). A model could also be obtained while the formula is satisfiable. The more interesting question "how well the formula is satisfied" is not well studied.

While there are various underlying theories in SMT, the most commonly used one is the linear real arithmetic. Linear operations and constraints can be encoded in the theory and there are efficient algorithms that can solve its satisfiability. The most well-known algorithm is the simplex method which is initially proposed by Dantzig [10]. Given a set of linear constraints, the worst-case time complexity of simplex method (with any known pivoting strategy) is exponential. However, it is also proven that the time complexity is polynomial [31] on average. To utilize the simplex method for SMT formulas, revised versions are proposed in [12, 26, 27], which are easier to be integrated into SMT solving frameworks that incorporates multiple theory solvers. In this paper, we further develop the revised simplex method for implicit equation detection and dimension computation.

We mention the fact that the ellipsoid algorithm can check the satisfiability of linear constraints in polynomial time. However, it is not faster than the simplex method in practice. Furthermore, it is difficult to be integrated into the SMT solving framework. In this paper, we employ ellipsoid algorithm to help rounding a polytope.

Generally speaking, an SMT formula may be an arbitrary Boolean combination of linear constraints. As in this paper, the volume computation typically consists of two steps: (1) enumerating all propositional models and (2) summating the volume of the linear constraints corresponding to each propositional model.

For the first step, it is essentially partitioning the solution space to disjoint regions. This is usually done by analyzing the Boolean structure of the input formula. The algorithm iteratively finds propositional models for the Boolean skeleton of the input formula, each corresponding to a region. This is similar to propositional model counting [32]. In practice, partitioning is often done by a modified DPLL(T) [24, 28] algorithm or by searching on certain data structure that is compiled from the input formula [17]. In the former manner, we search iteratively for propositional models. Once a model is found, it should be excluded in the following search in order to avoid counting the same region for multiple times. In the latter manner, we should develop a search strategy that better utilizes the theory solver. For instance, in this paper, we incorporate the theory solver and dimension detection to prune the search space. In both ways, the number of partial models should be reduced as much as possible since it dominates the number of calls to volume computation for a conjunction of linear constraints. The closest work to this paper is [24].

The second step is to compute the volume of each propositional model. Each proposition variable corresponds to a linear constraints and thus the propositional model corresponds to a set of linear constraints. Its volume is computed either by exact algorithms [3, 8] or approximation algorithms [14, 18, 22, 23]. The (bounded) solution space for linear constraints is a polytope. In exact algorithms, the polytope is often triangulated to simplices whose volume is easy to compute. In approximation algorithms, the volume is computed by a multi-phase Monte-Carlo procedure. A known fact is that the complexity of exact algorithms is proven to be #P-hard [15] while the randomized algorithms could reach a given accuracy with a given probability in polynomial time. Actually, the series of works after the path-breaking paper [14] has reduced the theoretical complexity from $O^*(n^{23})$ to $O^*(n^4)$. What's more interesting is the absence of any implementation for such algorithms. Despite the beautiful theoretical result, the intrinsic complexity of such randomized algorithms is rather high which makes them impractical. While the volume computation is of practical interest, there is also research that aims at practically estimating the volume without pursuing beautiful theoretical complexity. In [21] the author proposed a direct Monte-Carlo method. However, only simple examples in 3 dimension are given. The major obstacle to use direct Monte-Carlo method is called the curse of dimensionality, i.e, when the dimension increases, the probability of a uniformly sampled point lies inside a polytope decreases fast. The approximation could be very inaccurate when there is not enough sample points inside the polytope. For degenerate cases, theoretical volume (in full-dimension) is 0. Thus we are not even likely to have any sample point inside. To the best of our knowledge, there is not much related work on dealing with degenerate cases. But we believe the ability to handle degenerate cases is required for a practical volume computation algorithm.

7 Conclusion

In this paper, we proposed a randomized algorithm for volume approximation. Our algorithm can solve problem instances of practical interest. Experimental results show that our method is accurate and time-saving. It consumes much less time than the exact algorithms and is more accurate than the randomized algorithms. It is helpful for estimating the volume of path conditions and thus can be used in software engineering.

Acknowledgments I would like to thank Feifei Ma, a coauthor of the related work [24], for the insightful suggestions. I benefited from discussing with her when carrying out this work.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **100**(6), 509–516 (1978)
2. Athreya, K.: Unit ball in high dimensions. Resonance **13**(4), 334–342 (2008)
3. Avis, D.: Computational experience with the reverse search vertex enumeration algorithm. Optim. Methods Softw. **10**(2), 107–124 (1998)

4. Avis, D.: A revised implementation of the reverse search vertex enumeration algorithm. In: Polytopescombinatorics and Computation, pp. 177–198. Springer (2000)
5. Ball, T., Larus, J.R.: Branch prediction for free. In: Proceedings of PLDI'93, pp. 300–313. ACM, New York (1993)
6. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. Handb. Satisfiability **185**, 825–885 (2009)
7. Barrett, C., Tinelli, C.: CVC3. In: Computer Aided Verification, pp. 298–302. Springer (2007)
8. Büeler, B., Enge, A., Fukuda, K.: Exact volume computation for polytopes: a practical study. In: Polytopes-Combinatorics and Computation, pp. 131–154. Springer (2000)
9. Buse, R.P., Weimer, W.: The road not taken: estimating path execution frequency statically. In: Proceedings of ICSE'09, pp. 144–154. IEEE Computer Society (2009)
10. Dantzig, G.B.: Linear Programming and Extensions. Princeton university press (1998)
11. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer (2008)
12. Dutertre, B., De Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proceedings of CAV'06, pp. 81–94. Springer (2006)
13. Dutertre, B., De Moura, L.: The yices SMT solver. Tool paper at http://yices.csl.sri.com/tool-paper.pdf, **2**, 2, (2006)
14. Dyer, M., Frieze, A., Kannan, R.: A random polynomial-time algorithm for approximating the volume of convex bodies. JACM **38**(1), 1–17 (1991)
15. Dyer, M.E., Frieze, A.M.: On the complexity of computing the volume of a polyhedron. SIAM J. Comput. **17**(5), 967–974 (1988)
16. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization. Springer (1988). http://eudml.org/doc/204187
17. Huang, J., Darwiche, A.: Using DPLL for efficient OBDD construction. In: Theory and Applications of Satisfiability Testing, pp. 157–172. Springer (2005)
18. Kannan, R., Lovász, L., Simonovits, M.: Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. Random Struct. Algoritm. **11**(1), 1–50 (1997)
19. King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976)
20. Liu, S., Zhang, J.: Program analysis: from qualitative analysis to quantitative analysis (nier track). In: Proceedings of ICSE'11, pp. 956–959. IEEE (2011)
21. Liu, S., Zhang, J., Zhu, B.: Volume computation using a direct monte carlo method. In: Computing and Combinatorics, pp. 198–209. Springer (2007)
22. Lovász, L., Simonovits, M.: Random walks in a convex body and an improved volume algorithm. Random Struct. Algoritm. **4**(4), 359–412 (1993)
23. Lovász, L., Vempala, S.: Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. J. Comput. Syst. Sci. **72**(2), 392–417 (2006)
24. Ma, F., Liu, S., Zhang, J.: Volume computation for boolean combination of linear arithmetic constraints. In: Proceedings of CADE'09, pp. 453–468. Springer (2009)
25. Marsaglia, G.: Choosing a point from the surface of a sphere. Ann. Math. Stat. **43**(2), 645–646 (1972)
26. Necula, G.C.: Proof-Carrying Code. Design and Implementation. Springer (2002)
27. Nelson, C.G.: Techniques for program verification. XEROX Research Center (1981)
28. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). JACM **53**(6), 937–977 (2006)
29. Păsăreanu, C.S., Visser, W.: A survey of new trends in symbolic execution for software testing and analysis. Softw. Tools Technol. Transfer **11**(4), 339–353 (2009)
30. Poulding, S., Clark, J.A.: Efficient software verification: Statistical testing using automated search. IEEE Trans. Softw. Eng. **36**(6), 763–777 (2010)
31. Smale, S.: On the average number of steps of the simplex method of linear programming. Math. Program. **27**(3), 241–262 (1983)
32. Wei, W., Selman, B.: A new approach to model counting. In: Theory and Applications of Satisfiability Testing, pp. 324–339. Springer (2005)