VolCE: An Efficient Tool for Solving #SMT(LA) Problems

Cunjing Ge[13], Feifei Ma[123], and Jian Zhang[13]

[1] State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
{gecj,maff,zj}@ios.ac.cn,
[2] Laboratory of Parallel Software and Computational Science,
Institute of Software, Chinese Academy of Sciences
[3] University of Chinese Academy of Sciences

Abstract. We present VolCE, a tool for computing and estimating the size of the solution space of SMT(LA) constraints. VolCE supports the SMT-LIB format. It integrates SMT solving with volume computation/estimation and integer solution counting for convex polytopes. Several effective techniques are adopted, which enable the tool to deal with high-dimensional problem instances efficiently.

1 Introduction

In recent years, there have been a lot of works on solving the Satisfiability Modulo Theories (SMT) problem. Quite efficient SMT solvers have been developed, such as CVC3, Z3 and Yices [1, 5, 7]. In [17, 10], we studied the counting version of SMT solving, and presented some techniques for computing the size of the solution space efficiently. This problem can be regarded as an extension to SMT solving, and also an extension to model counting in the propositional logic. Its interesting applications in several fields include program analysis [15, 11, 12], probabilistic inference [18, 4], planning [6] and privacy/confidentiality verification [8].

This paper presents the tool VolCE[4] for the counting version of SMT(LA)[5]. The input of the tool is an SMT(LA) formula. The output of the tool is the "volume" of the solution space, or the number of solutions in case that the domains (of variables) are all discrete. VolCE supports the SMT-LIB (v2.0) format which is a common language to describe SMT formulas. It is an integration of SMT solving and polytope subroutines, such as, volume computation, volume estimation and integer point counting for convex polytopes. Moreover, VolCE adopts bunch and cache strategy to reduce the number of calls of polytope subroutines, and employs several more techniques to reduce the difficulty of polytope subroutines. As a result, the tool is enabled to deal with high-dimensional problem instances efficiently.

[4] Our tool VolCE is available at http://www.github.com/bearben/volce

[5] Here LA stands for linear arithmetic. So LIA and LRA stands for linear integer arithmetic and linear real arithmetic respectively.

2 Background

This section provides brief overviews of SMT(LA) formulas and convex polytopes as far as is needed to make this paper self-contained.

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

Let us consider two specific types of numeric variables, the integers and reals. There are **SMT(LIA)** formulas and **SMT(LRA)** formulas. For an SMT(LIA) formula, we count the number of solutions. For an SMT(LRA) formula, we compute the volume of the solution space instead. The assignment of the propositional skeleton of the SMT(LA) formula corresponds to a conjunction of linear inequalities which can be regarded as a convex polytope.

Tools for Polytope Subroutines. To describe `VolCE`, we introduce the following three tools, which are also called **polytope subroutines**:

- **PolyVest.** In [9, 10], we introduced a probabilistic algorithm for volume estimation on convex polytopes. A tool called `PolyVest` was implemented which can give results with (ϵ, δ)-bound, i.e., for a given polytope P, `PolyVest` returns an estimation that lies in the interval $[(1+\epsilon)^{-1}\#P, (1+\epsilon)\#P]$ with probability at least $1-\delta$, where $\#F$ is the exact volume of P. It can efficiently handle polytopes with dozens of dimensions.
- **Vinci.** Before `PolyVest`, there was already a tool available called `Vinci` [2] which can compute the volume of a convex polytope. It gives the exact volume instead of an estimation. The input of `Vinci` should be a set of LRA constraints.
- **LattE.** `LattE` [16] is a tool dedicated to the counting of lattice points inside convex polytopes. It requires the input polytopes in the integer matrix A and vector b.

3 Architecture

`VolCE` has the following three functions:

- Estimate volume for SMT(LRA) formulas;
- Compute volume for SMT(LRA) formulas;
- Count the number of lattice points for SMT(LIA) formulas.

The basic idea of `VolCE` is to enumerate feasible assignments by solving the SMT(LA) formula and accumulate the volumes of these assignments. Polytope volume computation/estimation (or lattice counting) serves as a subroutine which produces the volume (or solution count) of each feasible assignment. A schematic overview is shown in Fig. 1. A regular run of `VolCE` has three stages: parsing and rewriting, feasible assignments enumeration and polytope subroutines.

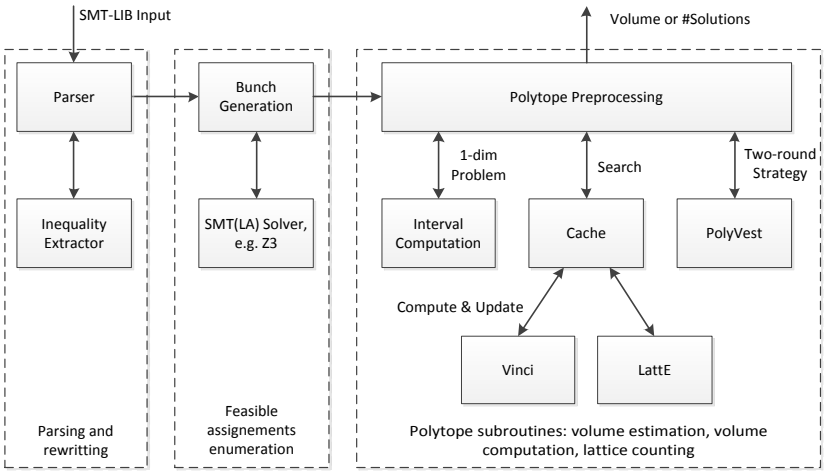

Fig. 1. Schematic overview of `VolCE`

3.1 Parsing and Rewritting

The parser reads the input and recursively builds an abstract syntax DAG (directed acyclic graph). Like most SMT solvers, basic rewriting rules are applied to simplify the DAG during the parsing process.

Inequality Extraction. `VolCE` is essentially a divide-and-conquer algorithm. It partitions the solution space of an SMT(LA) formula into polytopes and calls polytope subroutines to handle each polytope. Recall that polytope subroutines only take linear inequalities as inputs, so `VolCE` has to extract inequalities from SMT(LA) formulas. The inequality extractor is a component to rewrite the SMT(LA) expression into a combination of Boolean skeletons and normalized linear inequalities. Parser calls extractor whenever it has parsed a comparison operator ($=, <, \leq, >, \geq$, distinct) which is the root of a DAG of inequalities. Note that such a DAG may consist of more than one inequality if it contains `ite` operators. For a DAG with `ite`s, the extractor recursively traverses the DAG, records the conditions of each `ite` and then reconstructs a syntax tree with Boolean variables which represents those extracted inequalities. Each time an inequality is extracted, `VolCE` creates a Boolean variable to substitute this inequality in the DAG and adds a constraint to link the variable with the inequality.

Inequality Normalization. The extraction procedure also normalizes inequalities. With Boolean operations, one can represent $<, \leq, >$ and $\geq$ simply by $\leq$, e.g., $x + y < 0 \Leftrightarrow not\ (x + y \geq 0) \Leftrightarrow not\ (-x - y \leq 0)$. Thus a normalized inequality is of the form $a_0 + a_1x_1 + \ldots + a_nx_n = (\leq)\ 0$, where a_is are constants, x_is are variables, and $a_0 \geq 0$.

3.2 Feasible Assignments Enumeration

In the second stage, `VolCE` tries to enumerate all feasible assignments. It calls the SMT solver to obtain a feasible assignment, then adds a constraint to rule out the assignment just found, and so forth. `VolCE` currently uses `Z3`, as the SMT(LA) solver, through APIs. Note that it is easy to employ other modern SMT solvers since `VolCE` sees SMT solving as a black box.

Bunch Strategy [17]. To reduce the number of calls of polytope subroutines, we proposed a strategy in [17] that combines the feasible assignments into "bunches". Each time a feasible assignment is obtained, we search the neighbourhood of this assignment by negating its literals. We can combine the original assignment with one of its feasible neighbour assignments. Then we obtain a partial assignment that still propositionally satisfies the formula. The resulting assignment may cover a bunch of feasible assignments, hence is called a "bunch". The polytope subroutine is called for the polytope corresponding to each bunch rather than each feasible assignment, so that the number of calls is reduced.

3.3 Polytope Subroutines

In the third stage, `VolCE` calls polytope subroutines for each bunch to obtain the volume or the solution count, and then sums up. Recall that `VolCE` has three functions, so there are three polytope subroutines: (i) polytope volume estimation with tool `PolyVest`, (ii) polytope volume computation with tool `Vinci`, and (iii) polytope lattice couting with tool `LattE`. Recall that `PolyVest` produces estimation with (ϵ, δ)-bound. Since `VolCE` simply sums up the estimations from `PolyVest`, the volume estimation presented by `VolCE` also satisfies (ϵ, δ)-bound. As a result, `VolCE` has two command-line parameters `-epsilon` and `-delta` to control the accuracy of the volume estimation.

Polytope subroutines are time-consuming and usually the bottleneck of the whole program. `VolCE` employs a new cache strategy to reuse the results of polytope subroutines. `VolCE` also integrates techniques proposed in [10] to detect and solve cases which frequently occur in #SMT problems but inefficiently handled by polytope subroutines.

Reusing with Cache. `VolCE` stores the results of calls of polytope subroutines in a cache, so that it can retrieve the result of a problem which has already been solved from the cache. Each call of a polytope subroutine corresponds to a polytope. Because `VolCE` has extracted and stored all inequalities, the polytope passed to a polytope subroutine is essentially an assignment of inequalities. So we set the assignment of inequalities to be the key and bind it to the results returned by the polytope subroutine. For SMT(LA) formulas without free Boolean variables, the conjunctions of inequalities under different feasible assignments should be disjoint with each other. However, it is common that an SMT(LA) formula contains free Boolean variables. In this case, different feasible assignments may correspond to the same polytope. Besides, polytope partition techniques could partition different polytopes into the same sub-dimensional polytope.

Polytope Preprocessing [10]. The number of variables is a key factor about the difficulty of a polytope subroutine. `VolCE` employs polytope preprocessing techniques for dimension reduction. `VolCE` first partitions the set of linear inequalities, which corresponds to a given bunch, into mutually independent groups of inequalities, then applies the polytope subroutine to obtain the size of solution space of each group of inequalities, and finally returns the multiplications. `VolCE` also tries to check and reduce irrelevant variables whenever it calls polytope subroutine.

Two-round Strategy [10]. As the number of random points increases, the accuracy of estimation improves, and the estimation process also takes more time. It is important to balance the accuracy and the running time since the estimation subroutine is usually called many times. Therefore, `VolCE` employs a **two-round strategy** [10] that can dynamically determine a proper weight for each feasible (partial) assignment.

4 Experimental Results

In this section, we report experimental results about the performance of `VolCE`. `VolCE` provides a command-line parameter `-w` to quickly set bounds to numeric variables. Specifically, with such word-length parameter w, `VolCE` bounds each variable in the domain $[-2^{w-1}, 2^{w-1} - 1]$. We set $w = 8$, $\epsilon = 0.5$ and $\delta = 0.1$ for experiments on SMT(LRA) formulas. The settings of w on SMT(LIA) formulas are listed with experimental results. The experiments are conducted on a workstation with 3.40GHz Intel Core i7-2600 CPU and 8GB memory. The test cases [6] (more than 300) include:

- Random instances `lra`$\_b\_n\_l$: which have b Boolean variables, n real variables and l inequalities. They are generated by randomly choosing coefficients of inequalities and literals of clauses.
- Random instances `lia`$\_b\_n\_l$: which are similar to `lra`$\_b\_n\_l$ but are SMT(LIA) formulas.
- Instances generated from static program analysis. We analyzed the following programs: (i) `abs`: a function which calculates absolute values; (ii) `findmiddle`: a function which finds the middle number among 3 numbers; (iii) `Space_manage`: a program related to space technology; (iv) `tritype`: a program which determines the type of a triangle; (v) `calDate`: a function which converts the special date into a Julian date; (vi) `tcas`: a program about the traffic collision avoidance system; (vii) `FINDpath`: a selection program FIND [13]; (viii) `getopPath`: a program function called *getop()* [14].

In the following tables, "——" means that the instance takes more than one hour to solve (or the tool runs out of memory).

Table 1 presents the performance results of volume estimation and computation functions of `VolCE` on SMT(LRA) formulas. For lack of space, only a part

[6] The set of benchmarks is available at http://www.github.com/bearben/volce

			Estimation					Computation					
Instance	#FA	#bunch	#part	#calls	dims	Result	Time(s)	#part	#calls	#reuses	dims	Result	Time(s)
lra_23_23_23	225	194	0	240	22.95	2.23E+50	2730.71	——	——	——	——	——	——
lra_18_18_18	180	137	226	456	9.06	1.21E+41	1785.82	——	——	——	——	——	——
lra_22_22_22	49	23	34	68	10.99	3.82E+48	1331.35	——	——	——	——	——	——
lra_24_24_24	139	84	106	212	10.97	2.73E+53	850.54	——	——	——	——	——	——
lra_16_16_16	509	269	2	416	14.43	2.43E+36	409.29	——	——	——	——	——	——
lra_25_25_12	189	98	67	261	14.49	1.94E+60	316.67	——	——	——	——	——	——
lra_22_22_11	16	6	4	16	13.25	4.33E+50	127.09	——	——	——	——	——	——
lra_13_13_13	31	26	0	51	12.96	3.77E+29	70.14	——	——	——	——	——	——
lra_14_14_7	51	38	0	70	10.97	1.96E+34	23.78	0	5	33	10.80	2.05E+34	4.82
lra_17_17_8	70	24	0	47	10.83	1.15E+41	20.89	0	13	11	10.69	1.22E+41	6.74
lra_21_21_10	3	1	2	4	7.50	1.07E+48	18.37	1	2	0	7.50	1.04E+48	4.13
lra_24_12_12	63	30	0	49	10.88	4.42E+26	12.70	0	25	5	10.88	4.22E+26	30.18
lra_11_11_11	57	30	0	41	10.00	6.72E+22	8.04	0	19	11	10.00	6.52E+22	9.27
lra_12_12_12	14	5	0	9	11.00	1.13E+27	6.78	0	5	0	11.00	1.11E+27	11.53
lra_20_10_10	19	10	0	19	9.79	3.02E+22	6.78	0	10	0	9.80	3.04E+22	5.37
lra_20_20_10	68	67	134	526	3.92	5.91E+47	4.59	67	15	248	3.60	5.68E+47	0.08
lra_10_10_10	17	13	0	16	10.00	4.43E+21	3.80	0	7	6	10.00	4.24E+21	2.18
lra_22_11_11	32	18	0	22	10.86	3.75E+22	3.54	0	9	9	10.78	3.74E+22	10.19
lra_15_15_7	74	38	71	142	4.97	1.86E+36	3.05	38	15	61	5.67	1.86E+36	0.06
lra_7_7_7	50	27	0	46	6.83	7.56E+16	1.22	0	21	6	6.86	7.52E+16	0.20
lra_10_10_5	3	2	2	6	5.00	1.43E+23	0.28	1	3	0	5.00	1.37E+23	0.03

Table 1. Performance results of `VolCE` on SMT(LRA) formulas

of results are listed here. In Table 1, column 1 gives the instance name, column 2 the number of feasible assignments and column 3 the number of bunches enumerated by `VolCE`. Column 4 to 8 give the results of volume estimation function which correspond to the number of polytopes which are partitioned, the number of calls of estimation subroutine, the average number of dimensions of problems handled by polytope estimation subroutine, the outputs and the running times, respectively. Column 9 to 14 give the results of volume computation function. Note that column 11 presents the number of solutions reused.

The results in Table 1 show that `VolCE` can solve random instances with over 20-dimensions in reasonable time. Compared with the volume estimation function, the volume computation is more efficient on small instances. We observe that for all the benchmarks, `VolCE` gives estimations within the tolerance ϵ.

In addition, we find that the bunch strategy works well. The feasible assignments are reduced by half on most of instances. Table 1 also shows that the solution reuse technique works on some instances and sometimes very effective, e.g., "lra_20_20_10" and "lra_15_15_7". Note that the reuse technique is currently not available to the volume estimation function. Besides, the results show that polytope partition technique is useful on both functions. Due to the two-round strategy, the number of polytopes partitioned during volume estimation is sometimes more than the number of bunches.

To evaluate the performance of `VolCE` for solution counting, we compare it with `SMTApproxMC` [3] which is a state-of-the-art hashing-based approximate counter for SMT(BV) formulas. For comparison, we transformed SMT(LIA) formulas into SMT(BV) formulas manually by replacing integer variables with fixed-length variables, bit-vector constants and bit operations. We experimented `SMTApproxMC` with parameters $\epsilon = 0.8$ and $\delta = 0.2$. It guarantees the output

						VolCE				SMTApproxMC		
Instance	w	#bool	#var	#ineq	#FA	#bunch	#part	Result	Time(s)	TB.	Result	Time(s)
FINDpath_1	8	0	8	10	1	1	0	4.08E+06	0.07	32	3.98E+06	1021
FINDpath_2	8	0	8	21	1	1	0	8.75E+04	0.05	32	9.00E+04	116.8
getopPath_1	8	0	1	10	5	5	0	2.42E+02	0.02	8	2.45E+02	2.57
getopPath_2	8	0	3	20	18	18	18	8.09E+03	0.06	24	8.38E+03	14.96
findmiddle_4	8	0	3	6	2	2	0	5.53E+06	0.04	24	——	——
findmiddle_6	8	0	3	6	4	4	0	1.31E+05	0.05	24	1.33E+05	151.5
findmiddle_8	8	0	3	6	2	2	0	6.53E+04	0.04	24	6.21E+04	207.9
Space_manage_38	8	0	7	15	6	6	6	5.41E+14	0.04	56	——	——
Space_manage_49	8	0	13	20	70	30	30	2.51E+27	0.06	104	——	——
tcas_1200	16	0	5	14	11	4	4	2.81E+14	0.04	80	——	——
tcas_1201	16	0	7	18	88	16	16	1.21E+24	0.03	112	——	——
tcas_1214	16	0	7	18	22	8	8	1.84E+19	0.04	112	——	——
lia_12_6_6	8	12	6	6	4	2	0	4.16E+13	889.08	96	——	——
lia_6_6_3	8	6	6	3	3	2	1	9.79E+13	344.01	96	——	——
lia_7_7_3	8	7	7	3	2	2	0	2.48E+16	70.44	102	——	——
lia_8_4_4	8	8	4	4	9	7	0	1.14E+09	33.52	64	——	——
lia_10_5_5	8	10	5	5	6	3	0	2.15E+11	8.43	80	——	——
lia_4_4_2	8	4	4	2	3	2	0	3.66E+09	0.40	64	——	——
lia_6_3_3	8	6	3	3	1	1	0	1.28E+06	0.14	48	1.12E+06	878.01
lia_2_2_2	8	2	2	2	3	3	0	9.43E+04	0.05	32	8.74E+04	21.86
lia_3_3_3	8	3	3	3	2	2	0	2.97E+06	0.04	48	2.91E+06	909.15
lia_2_2_1	8	2	2	1	2	2	0	6.55E+04	0.03	32	6.50E+04	22.41

Table 2. Comparative results of `VolCE` and `SMTApproxMC` on SMT(LIA) formulas

lying in interval $[1.8^{-1}R_F, 1.8R_F]$ with probability at least 80%, where R_F is the real count of a given formula F.

Table 2 presents the result of comparing the performance of `VolCE` with `SMTApproxMC` on a subset of our benchmarks. Column 2 gives the word-length setting, and column 11 the total number of bits of the transformed SMT(BV) formula. In these experiments, `VolCE` calls `LattE` for integer solution counting inside a polytope, so our tool returns the exact counts instead of estimations. Table 2 shows that our approach significantly outperforms `SMTApproxMC` for a large class of benchmarks. We observe that the running time of `SMTApproxMC` is closely related to the number of the solutions rather than the number of variables, i.e., the larger number of solutions, the more difficult for `SMTApproxMC` to handle. Besides, the results on industry instances demonstrate that our tool is potentially useful in program analysis.

5 Concluding Remarks

`VolCE` is a tool for computing and estimating the volume of the solution space (or counting the number of solutions), given a formula/constraint which is a Boolean combination of linear arithmetic inequalities. For medium sized SMT(LA) formulas, it can provide exact volume computation results or exact number of solutions. For larger SMT(LRA) formulas, it can perform volume estimation with high accuracy, due to the use of effective heuristics. We believe that the tool will be useful in a number of domains, such as program analysis and probabilistic verification.

References

1. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proc. of CAV*, pages 171–177, 2011.
2. B. Büeler, A. Enge, and K. Fukuda. *Exact Volume Computation for Polytopes: A Practical Study*, pages 131–154. Birkhäuser Basel, 2000.
3. S. Chakraborty, K. S. Meel, R. Mistry, and M. Y. Vardi. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*, pages 3218–3224, 2016.
4. M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.
5. L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *Proc. of TACAS*, pages 337–340, 2008.
6. C. Domshlak and J. Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *J. Artif. Intell. Res. (JAIR)*, 30:565–620, 2007.
7. B. Dutertre. Yices 2.2. In *Proc. of CAV*, pages 737–744, 2014.
8. M. Fredrikson and S. Jha. Satisfiability modulo counting: a new approach for analyzing privacy properties. In *Proc. of CSL-LICS*, pages 42:1–42:10, 2014.
9. C. Ge and F. Ma. A fast and practical method to estimate volumes of convex polytopes. In *Proc. of FAW*, pages 52–65, 2015.
10. C. Ge, F. Ma, P. Zhang, and J. Zhang. Computing and estimating the volume of the solution space of SMT(LA) constraints. *Accepted to appear in TCS, http://dx.doi.org/10.1016/j.tcs.2016.10.019.*
11. J. Geldenhuys, M. B. Dwyer, and W. Visser. Probabilistic symbolic execution. In *Proc. of ISSTA*, pages 166–176, 2012.
12. K. Gleissenthall, B. Köpf, and A. Rybalchenko. Symbolic polytopes for quantitative interpolation and verification. In *Proc. of CAV*, pages 178–194, 2015.
13. C. A. R. Hoare. Proof of a program: FIND. *Commun. ACM*, 14(1):39–45, 1971.
14. B. W. Kernighan. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.
15. S. Liu and J. Zhang. Program analysis: from qualitative analysis to quantitative analysis. In *Proc. of ICSE*, pages 956–959, 2011.
16. J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *J. Symb. Comput.*, 38(4):1273–1302, 2004.
17. F. Ma, S. Liu, and J. Zhang. Volume computation for boolean combination of linear arithmetic constraints. In *Proc. of CADE*, pages 453–468, 2009.
18. D. Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.