Solution Counting for Propositional Logic and Satisfiability Modulo Theories

Feifei Ma

Institute of Software
Chinese Academy of Sciences

2017-04-11

Outline

1 #SAT
- Exact Model Counting
- Approximate Model Counting

2 #SMT
- Exact Approach
- Approximate Approach

Model Counting (#SAT)

- Count #models of a propositional formula
  - E.g. $(p \wedge q \vee r)$ has 5 models.
- has found applications in
  - probabilistic inference
  - planning
  - combinatorial designs
- #P-complete problem, even for some polynomial-time solvable problems like 2-SAT

Outline

1 #SAT
- Exact Model Counting
- Approximate Model Counting

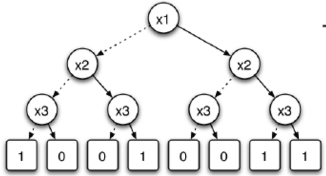
2 #SMT
- Exact Approach
- Approximate Approach

DPLL-based Model Counting

- The earliest practical approach is based on an extension of DPLL SAT solvers.
- Key techniques:
  - Component Analysis: If the constraint graph G of a CNF formula F can be partitioned into disjoint components $G_1, G_2, \ldots, G_k$, then $\#F = \#F_1 \times \#F_2 \ldots \times \#F_k$.
  - Component Caching: Store the sub-formulas and their model counts for reutilization. Works better if more reasoning is employed at each node of the DPLL search tree.

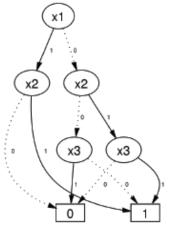
Model Counting based on Knowledge Compilation

- Convert or compile the CNF formula into other logic forms from which the count can be deduced easily.
- Binary Decision Diagram (BDD):



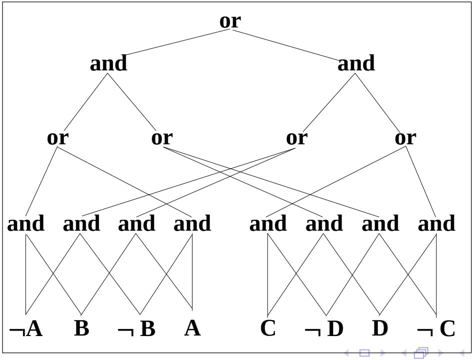
x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Binary decision tree and truth table for the function f(x1, x2, x3) = -x1 * -x2 * -x3 + x1 * x2 + x2 * x3



BDD for the function f

- deterministic,Decomposable Negation Normal Form (d-DNNF): An NNF satisfying the following properties
  - Decomposability: $Var(n_i) \cap Var(n_i) = \phi$ for any two children n_i and n_j of an and-node n.
  - Determinism:$F(n_i) \wedge F(n_i) =$ is inconsistent for any two children n_i and n_j of an or-node n.



Outline

1 #SAT
- Exact Model Counting
- Approximate Model Counting

2 #SMT
- Exact Approach
- Approximate Approach

Approximate Model Counters

1. Guarantee-less counters: can be very efficient and provide good approximation without guarantees.
2. Bounding counters: provide a lower/upper bound for $\#F$ with probability at least $1 - \delta$.
3. (ϵ, δ)-counters: on every input formula F, $\epsilon > 0$ and $\delta > 0$, output a number $\tilde{Y}$ such that $\Pr[(1+\epsilon)^{-1}\#F \leq \tilde{Y} \leq (1+\epsilon)\#F] \geq 1 - \delta$.

Hash Functions

- Let $\mathcal{H}_F$ be a family of XOR-based bit-level hash functions on the variables of a formula F. Each hash function $H \in \mathcal{H}_F$ is of the form $H(x) = a_0 \bigoplus_{i=1}^{n} a_i x_i$, where $a_0, \ldots, a_n$ are Boolean constants. In the hashing procedure `Hashing(F)`, a function $H \in \mathcal{H}_F$ is generated by independently and randomly choosing a_is from a uniform distribution. Thus for an assignment α, it holds that $\Pr_{H \in \mathcal{H}_F}(H(\alpha) = \mathit{true}) = \frac{1}{2}$. Gevin a formula F, let F_i denote a hashed formula $F \wedge H_1 \wedge \cdots \wedge H_i$, where $H_1, \ldots, H_i$ are independently generated by the hashing procedure. [1]

[1] S. Chakraborty, K. S. Meel and M. Y. Vardi, A Scalable Approximate Model Counter, Proc. of CP 2013

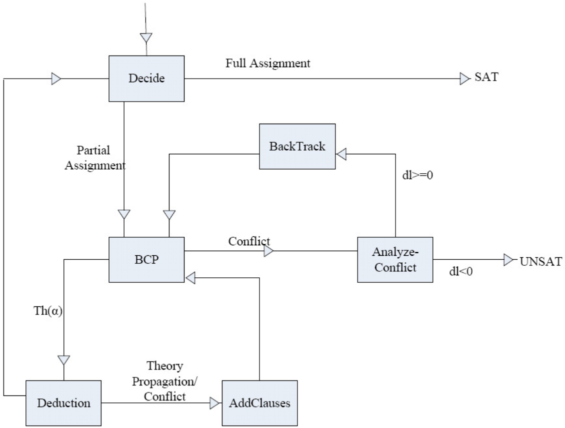
Algorithm 1 A hash-based (ϵ, δ)-Counter

```
function ApproxMC(F, T, pivot)
    for 1 to T do
        c ← ApproxMCCore(F, pivot)
        if (c ≠ 0) then AddToList(C, c)
    end for
    return FindMedian(C)
end function
function ApproxMCCore(F, pivot)
    F_0 ← F
    for i ← 0 to ∞ do
        s ← Counting(F_i, pivot + 1)
        if (0 ≤ s ≤ pivot) then return 2^i s
        H_{i+1} ← Hashing(F)
        F_{i+1} ← F_i ∧ H_{i+1}
    end for
end function
```

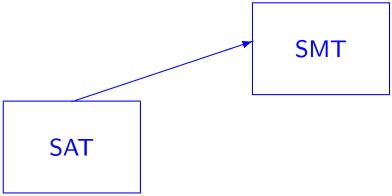
SMT

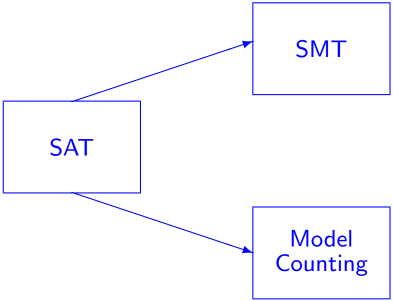
- **Satisfiability Modulo Theories (SMT)**
  - Satisfiability of logic formulas w.r.t background theories
  - Theories include: *linear arithmetic, arrays, bit vectors, uninterpreted functions*
  - E.g. $x + y > 0 \wedge y \leq 3 \vee x - y = 8$

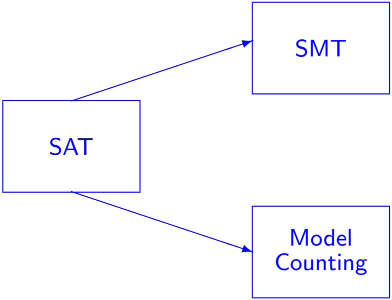
The DPLL(T) Procedure



SAT

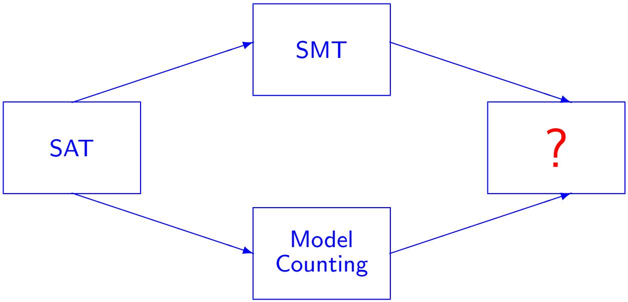

SMT
SAT


SMT
SAT
Model Counting



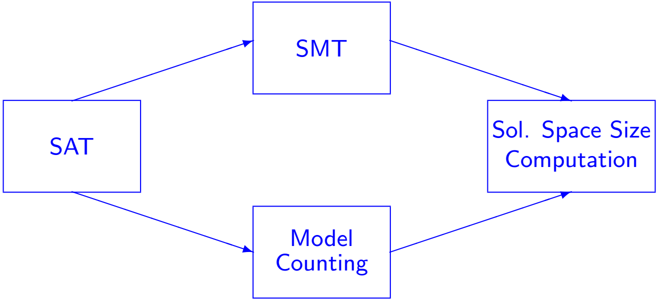
Question

What is the counting version of SMT?



Question

What is the counting version of SMT?



Question

What is the counting version of SMT?

To compute the number of solutions of an SMT formula

The Problem

#SMT

Computing the number of solutions of an SMT formula, i.e., the size/volume/density of the solution space.

- has potential applications in various areas.
  - Approximate reasoning.
    E.g. *Given a knowledge base* Φ *and a formula* φ*, where neither* $\Phi \models \varphi$ *nor* $\Phi \models \neg\varphi$*, compute the likelyhood of* φ *being* $\mathtt{True}$*.*
  - Program analysis and verification.
    path execution frequency, hot path

A New Measurement for Path Execution Frequency

- [Zhang, COMPSAC 2004]
- $\delta(P)$– the number of solutions of the path condition (or the percentage of solutions in the possible solution space).
  - We can get a feeling of how complete the testing is.
  - We may try to optimize the program by focusing on the “hot” paths.
- Related work: [Buse-Weimer, ICSE 2009]
  - syntactic v.s. semantic
  - estimation v.s. accurate calculation

How to compute $\delta(P)$?

Example

- ```
  int i, j;
  if (i+j > 10)
      j = 2;
  else j = 1;
  ```
  ```

How to compute $\delta(P)$?

Example

- ```
  int i, j;
  if (i+j > 10)
      j = 2;
  else j = 1;
  ```
- P1 (if-then): $i + j > 10$
  P2 (else): $i + j \leq 10$
  ```

How to compute $\delta(P)$?

Example

- ```
  int i, j;
  if (i+j > 10)
      j = 2;
  else j = 1;
  ```
- P1 (if-then): $i + j > 10$
  P2 (else): $i + j \leq 10$
- If i, j :[1..10], $\delta(P1) = 55$, $\delta(P2) = 45$.
  ```

How to compute $\delta(P)$?

Example

- ```
  int i, j;
  if (i+j > 10)
      j = 2;
  else j = 1;
  ```
- P1 (if-then): $i + j > 10$
  P2 (else): $i + j \leq 10$
- If i, j :[1..10], $\delta(P1) = 55$, $\delta(P2) = 45$.
- If i, j :[1..100], $\delta(P1) = 9955$, $\delta(P2) = 45$.
  ```

How to compute $\delta(P)$?

Example

- ```
  int i, j;
  if (i+j > 10)
      j = 2;
  else j = 1;
  ```
- P1 (if-then): $i + j > 10$
  P2 (else): $i + j \leq 10$
- If i, j :[1..10], $\delta(P1) = 55$, $\delta(P2) = 45$.
- If i, j :[1..100], $\delta(P1) = 9955$, $\delta(P2) = 45$.

$$\delta(P) = Volume(PathCond(P))$$
  ```

Path Execution Frequency / Probability

- Suppose there are m variables in the path condition of P, and the range length or domain size of the ith variable is l_i. We have

The Execution Frequency of P

$$\mathcal{XP}(P) = \frac{\delta(P)}{\prod_{1 \le i \le m} l_i} = \frac{Volume(PathCond(P))}{\prod_{1 \le i \le m} l_i}$$

Outline

1 #SAT
- Exact Model Counting
- Approximate Model Counting

2 #SMT
- Exact Approach
- Approximate Approach

SMT(LA)

- An SMT formula on linear arithmetic theory, denoted by SMT(LA), is composed of
  - Boolean variables b_i, numeric variables x_j.
  - The constraint ϕ: a Boolean formula $PS_\phi(b_1, \ldots, b_n)$ with definitions $b_i \equiv expr_{i1} \otimes \; expr_{i2}$.

  PS_ϕ is called the propositional skeleton of ϕ.
- Computing the volume of the solution space for SMT(LA) formulas generalizes
  - Model counting in propositional logic.
  - Classical volume computation problem for convex polytopes.

Classical Volume Computation

- A polytope is the bounded intersection of finitely many halfspaces/inequalities. Formally $\{\vec{x} | A\vec{x} \leq \vec{b}\}$.
- Tools are available to
  - compute the real solid volume of a polytope.
    Ex. **vinci, Qhull**.
  - compute the number of integer points within a polytope.
    Ex. **azove, LattE**.

A Straightforward Method

- An SMT(LA) instance ϕ is satisfiable if there is an assignment α to the Boolean variables such that

A Straightforward Method

- An SMT(LA) instance ϕ is satisfiable if there is an assignment α to the Boolean variables such that
  1. α propositionally satisfies ϕ;
  2. The corresponding linear inequalities are satisfiable/feasible.

  α is called a **feasible assignment**.

Feasible Assignment: an example

Example

$\phi = (((y + 3x < 1) \rightarrow (30 < y)) \vee (x \leq 60)) \wedge ((30 < y) \rightarrow \neg(x > 3) \wedge (x \leq 60))$, or equivalently

$$PS_\phi = ((b_1 \rightarrow b_2) \vee b_4) \wedge (b_2 \rightarrow \neg b_3 \wedge b_4)$$

where:

$$\begin{cases} b_1 \equiv (y + 3x < 1); \\ b_2 \equiv (30 < y); \\ b_3 \equiv (x > 3); \\ b_4 \equiv (x \leq 60); \end{cases}$$

$\alpha_1 = \{\neg b_1, \neg b_2, b_3, \neg b_4\}$ is a feasible assignment.

A Straightforward Method

- $Mod(\phi)$: the set of all feasible assignments of ϕ.
- $volume(\alpha)$: the volume of α, i.e., the volume of the polytope corresponding to α.
- The volume of a formula ϕ is denoted by $Volume(\phi)$.

$$Volume(\phi) = \sum_{\alpha \in Mod(\phi)} volume(\alpha)$$

A Straightforward Method

- *Mod*(ϕ): the set of all feasible assignments of ϕ.
- *volume*(α): the volume of α, i.e., the volume of the polytope corresponding to α.
- The volume of a formula ϕ is denoted by *Volume*(ϕ).

$$Volume(\phi) = \sum_{\alpha \in Mod(\phi)} volume(\alpha)$$

Find all feasible assignments, compute the volume of each assignment and add them up.

Improvement

- Computing the volume of a polytope is #P-hard. Need to reduce the number of calls to classical polytope volume computation routines.

Improvement

- Computing the volume of a polytope is #P-hard. Need to reduce the number of calls to classical polytope volume computation routines.
- Given an assignment α to the Boolean variabales in ϕ, we distinguish four cases:
  1. α satisfies ϕ propositionally, and the corresponding linear inequalities are satisfiable. (Here α is a feasible assignment.)
  2. α satisfies ϕ propositionally, while the corresponding linear inequalities are unsatisfiable.
  3. α falsifies ϕ propositionally, while the corresponding linear inequalities are satisfiable.
  4. α falsifies ϕ propositionally, and the corresponding linear inequalities are unsatisfiable.

Improvement

- Computing the volume of a polytope is #P-hard. Need to reduce the number of calls to classical polytope volume computation routines.
- Given an assignment α to the Boolean variabales in ϕ, we distinguish four cases:
  1. α satisfies ϕ propositionally, and the corresponding linear inequalities are satisfiable. (Here α is a feasible assignment.)
  2. α satisfies ϕ propositionally, while the corresponding linear inequalities are unsatisfiable.
  3. α falsifies ϕ propositionally, while the corresponding linear inequalities are satisfiable.
  4. α falsifies ϕ propositionally, and the corresponding linear inequalities are unsatisfiable.
- In cases (2) and (4), α is inconsistent, *volume*$(\alpha) = 0$, safe to be counted in.

An Example

$$PS_\phi = ((b_1 \rightarrow b_2) \vee b_4) \wedge (b_2 \rightarrow \neg b_3 \wedge b_4)$$

where:

$$\begin{cases} b_1 \equiv (y + 3x < 1); \\ b_2 \equiv (30 < y); \\ b_3 \equiv (x > 3); \\ b_4 \equiv (x \leq 60); \end{cases}$$

- There are 7 feasible assignments, 3 of which are

$$\alpha_1 = \{\neg b_1, \neg b_2, b_3, \neg b_4\}$$
$$\alpha_2 = \{\neg b_1, \neg b_2, \neg b_3, b_4\}$$
$$\alpha_3 = \{\neg b_1, \neg b_2, b_3, b_4\}$$

An Example

$$PS_\phi = ((b_1 \rightarrow b_2) \vee b_4) \wedge (b_2 \rightarrow \neg b_3 \wedge b_4)$$

where:

$$\begin{cases} b_1 \equiv (y + 3x < 1); \\ b_2 \equiv (30 < y); \\ b_3 \equiv (x > 3); \\ b_4 \equiv (x \leq 60); \end{cases}$$

- There are 7 feasible assignments, 3 of which are

$$\alpha_1 = \{\neg b_1, \neg b_2, b_3, \neg b_4\}$$
$$\alpha_2 = \{\neg b_1, \neg b_2, \neg b_3, b_4\}$$
$$\alpha_3 = \{\neg b_1, \neg b_2, b_3, b_4\}$$

- Volume computation for 3 Polytopes.

An Example: continued

Consider $\alpha_4 = \{\neg b_1, \neg b_2, \neg b_3, \neg b_4\}$. $\alpha_4 \models PS_\phi$, but the linear inequalities corresponding to α_4 are unsatisfiable, $volume(\alpha_4) = 0$.

$$\alpha_1 = \{\neg b_1, \neg b_2, b_3, \neg b_4\}$$
$$\alpha_2 = \{\neg b_1, \neg b_2, \neg b_3, b_4\}$$
$$\alpha_3 = \{\neg b_1, \neg b_2, b_3, b_4\}$$
$$\alpha_4 = \{\neg b_1, \neg b_2, \neg b_3, \neg b_4\}$$

Form a bunch with the cube $\{\neg b_1, \neg b_2\}$

$$\begin{aligned} & volume(\alpha_1) + volume(\alpha_2) + volume(\alpha_3) \\ = \; & volume(\alpha_1) + volume(\alpha_2) + volume(\alpha_3) + volume(\alpha_4) \\ = \; & volume(\{\neg b_1, \neg b_2\}) \end{aligned}$$

An Example: continued

Consider $\alpha_4 = \{\neg b_1, \neg b_2, \neg b_3, \neg b_4\}$. $\alpha_4 \models PS_\phi$, but the linear inequalities corresponding to α_4 are unsatisfiable, $volume(\alpha_4) = 0$.

$$\alpha_1 = \{\neg b_1, \neg b_2, b_3, \neg b_4\}$$
$$\alpha_2 = \{\neg b_1, \neg b_2, \neg b_3, b_4\}$$
$$\alpha_3 = \{\neg b_1, \neg b_2, b_3, b_4\}$$
$$\alpha_4 = \{\neg b_1, \neg b_2, \neg b_3, \neg b_4\}$$

Form a bunch with the cube $\{\neg b_1, \neg b_2\}$

$$\begin{aligned} & volume(\alpha_1) + volume(\alpha_2) + volume(\alpha_3) \\ = \; & volume(\alpha_1) + volume(\alpha_2) + volume(\alpha_3) + volume(\alpha_4) \\ = \; & volume(\{\neg b_1, \neg b_2\}) \end{aligned}$$

Volume computation for 1 Polytope.

Volume Computation in Bunches

Implement the idea within the deduction procedure of the SMT(LA) solver[2].

- When a feasible assignment α is found, try to obtain a smaller one α_c, such that $\alpha_c \models PS_\phi$.

[2]Feifei Ma, Sheng Liu, Jian Zhang: Volume Computation for Boolean Combination of Linear Arithmetic Constraints. CADE 2009.

Volume Computation in Bunches

Implement the idea within the deduction procedure of the SMT(LA) solver[2].

- When a feasible assignment α is found, try to obtain a smaller one α_c, such that $\alpha_c \models PS_\phi$.
- α_c contains α, and possibly other feasible assignments and inconsistent assignments, thus is called a **bunch**.

[2] Feifei Ma, Sheng Liu, Jian Zhang: Volume Computation for Boolean Combination of Linear Arithmetic Constraints. CADE 2009.

Volume Computation in Bunches

Implement the idea within the deduction procedure of the SMT(LA) solver[2].

- When a feasible assignment α is found, try to obtain a smaller one α_c, such that $\alpha_c \models PS_\phi$.
- α_c contains α, and possibly other feasible assignments and inconsistent assignments, thus is called a **bunch**.
- Add $volume(\alpha_c)$ to the total volume.

[2]Feifei Ma, Sheng Liu, Jian Zhang: Volume Computation for Boolean Combination of Linear Arithmetic Constraints. CADE 2009.

Table: Comparision of Algorithms

Instance	P	cls	V	Volume Computation in Bunches		Straightforward Method	
				Time (s)	#calls	Time (s)	#calls
Ran1	8	50	4	0.02	19	0.03	41
Ran2	10	40	5	0.06	50	0.14	182
Ran3	15	40	5	2.36	47	11.12	188
Ran4	20	40	5	116.72	259	431.15	17158
Ran5	10	20	6	1.04	41	7.81	212
Ran6	10	50	6	2.08	74	5.32	247
Ran7	15	50	6	2.15	57	10.97	257
Ran8	7	40	7	1.01	16	2.77	39
Ran9	12	40	7	50.29	250	502.75	1224
Ran10	15	50	7	303.14	856	3872.70	5224
Ran11	20	50	7	143.11	140	1889.36	807
Ran12	10	20	8	12.04	37	150.92	235
Ran13	10	40	8	51.10	91	398.02	379
Ran14	16	80	8	1074.48	669	4 hours	4273

P: number of linear constraints. `cls`: number of clauses.
V: number of numerical variables. `#calls`: number of calls to `VINCI`.

Outline

1 #SAT
- Exact Model Counting
- Approximate Model Counting

2 #SMT
- Exact Approach
- Approximate Approach

Volume Estimation for Convex Polytopes

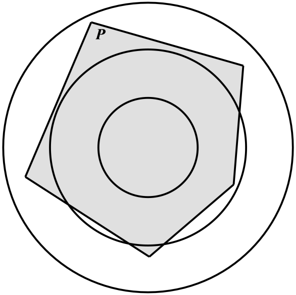
- Computing the volume of a polytope is #P-hard and is the bottleneck of volume computation for SMT(LA).
- To estimate the volume of a convex polytope:
  - The Monte-Carlo method suffers from the curse of dimensionality. The sample size has to grow exponentially to achieve a reasonable estimation.
  - The **Multiphase Monte-Carlo Algorithm**[3] is a polynomial time randomized approximation algorithm. But there lacks practical implementation.

[3] M. E. Dyer, A. M. Frieze and R. Kannan, A Random Polynomial Time Algorithm for Approximating the Volume of Convex Bodies, Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 1989

The Multiphase Monte-Carlo Algorithm (1)

Suppose a convex polytope P is defined by $P = \{Ax \leq b\}$. A sphere with radius R and center $x \in \mathbb{R}^n$ is denoted by $B(x, R)$.

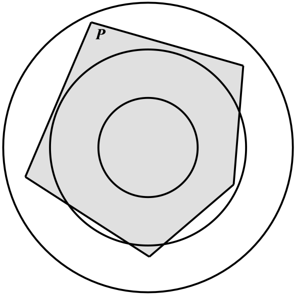
- Find an affine transformation T, $B(0,1) \subseteq T(P) \subseteq B(0,n)$
- Place $l = \lceil n \log_2 n \rceil$ concentric balls $\{B_i = B(0, 2^{i/n}),\ i = 0, \ldots, l\}$.



The Multiphase Monte-Carlo Algorithm (2)

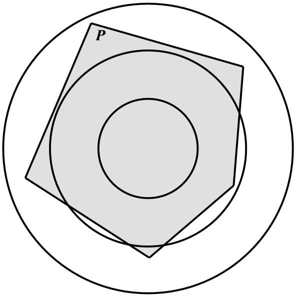
- Set $K_i = B_i \cap P$, then $K_0 = B(0,1)$, $K_l = P$ and

$$vol(P) = vol(B(0,1)) \prod_{i=0}^{l-1} \frac{vol(K_{i+1})}{vol(K_i)} \quad (1)$$

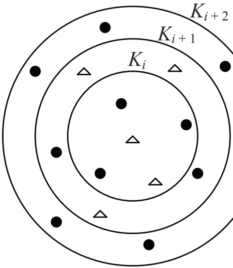


The Multiphase Monte-Carlo Algorithm (3)

- So we only have to estimates the ratio $\alpha_i = vol(K_{i+1})/vol(K_i)$, $i = 0, \ldots, l-1$. Note that $1 \leq \alpha_i \leq 2$. It is sufficient to estimate α_i with Monte-Carlo algorithm with polynomial number of random points.



Reutilization of Random Points



At the i-th phase, $\alpha_i = \frac{k_{i+1}}{k_i}$ is estimated.

- The original method: estimate α_i in natural order ($\alpha_0 \to \alpha_{l-1}$)
- Our method: estimate α_i in reverse order ($\alpha_{l-1} \to \alpha_0$). Random points are generated from K_l to K_0.

Reutilization of Random Points

Approximation of α_is in reverse order:

- fully exploits the random points generated in previous phases.
- saves 70% random points.
- has no side-effect on the error[4].

[4] Cunjing Ge, Feifei Ma, Peng Zhang, Jian Zhang. Computing and estimating the volume of the solution space of SMT(LA) constraints, Theoretical Computer Science, Available online 15 November 2016

Experiments

Comparison between Polyvest and Vinci

			Polyvest		Vinci	
Instance	n	m	Result	Time(s)	Result	Time(s)
cube_10	10	20	1038.18	0.952	1024	0.004
cube_14	14	28	16811.1	3.020	16384	0.160
cube_20	20	40	1.01008e+6	10.869	—	—
cube_30	30	60	1.08628e+9	54.257	—	—
cube_40	40	80	1.06866e+12	174.059	—	—
rh_8_25	8	25	766.744	0.628	785.989	0.884
rh_10_20	10	20	13711.1	1.164	13882.7	0.284
rh_10_25	10	25	5737.29	1.120	5729.52	5.100
rh_10_30	10	30	2051.68	1.154	—	—
cross_7	7	128	0.0250643	0.968	0.0253968	0.088
fm_6	15	59	296501	5.988	—	—
cc_8_10	8	70	153128	1.068	156816	6.764
cc_8_11	8	88	1.42154e+6	1.284	1.39181e+6	34.430

Thanks