Constrained Sampling and Counting

Moshe Y. Vardi
Rice University

Joint work with: Supratik Chakraborty (IIT Bombay), Daniel Fremont (UC Berkeley), Kuldeep Meel (Rice), and Sanjit Seshia (UC Berkeley)
How do we guarantee that systems work **correctly**?

**Functional Verification**

- Formal verification
  - Challenges: formal requirements, scalability
  - ~10-15% of verification effort

- Dynamic verification: *dominant approach*
Dynamic Verification

- Design is simulated with test vectors
- Test vectors represent different verification scenarios
- Results from simulation compared to intended results
- **Challenge**: Exceedingly large test spaces!
Motivating Example

How do we test the circuit works?

- Try for all values of $a$ and $b$
  - $2^{128}$ possibilities
  - Sun will go nova before done!
  - Not scalable
Constrained-Random Simulation

Sources for Constraints

- Designers:
  1. \( a +_{64} 11 \times_{32} b = 12 \)
  2. \( a <_{64} (b >> 4) \)

- Past Experience:
  1. \( 40 <_{64} 34 + a <_{64} 5050 \)
  2. \( 120 <_{64} b <_{64} 230 \)

- Users:
  1. \( 232 \times_{32} a + b \neq 1100 \)
  2. \( 1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200 \)

- Test vectors: solutions of constraints
Constrained-Random Simulation

Sources for Constraints

• Designers:
  1. $a +_{64} 11 \times_{32} b = 12$
  2. $a <_{64} (b \gg 4)$

• Past Experience:
  1. $40 <_{64} 34 + a <_{64} 5050$
  2. $120 <_{64} b <_{64} 230$

• Users:
  1. $232 \times_{32} a + b \neq 1100$
  2. $1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200$

Problem: How can we uniformly sample the values of $a$ and $b$ satisfying the above constraints?
Problem Formulation

Set of Constraints

SAT Formula

Sample satisfying assignments uniformly at random

SAT Sampling
Roadmap

- SAT Sampling
- SAT Counting
- Extensions
- Future Directions
Diverse Applications

- Search-based Synthesis
- Probabilistic Inference
- Planning under uncertainty
- Automatic Problem Generation
- Constrained Random Simulation
- SAT Sampling
Prior Work

- BGP
- BDD

- UniGen
- MCMC
- SAT-Based

Performance vs. Guarantees
Core Idea
Partitioning into cells

Cells should be roughly equal in size and small enough to enumerate completely
Partitioning into cells

Pick a random cell

Pick a random solution from this cell
How to Partition?

How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

Universal Hashing
[Carter-Wegman 1979]
Universal Hashing

- Hash functions: mapping $\{0,1\}^n$ to $\{0,1\}^m$
  - $(2^n$ elements to $2^m$ cells)

- Random inputs => All cells are roughly equal (in expectation)

- Universal family of hash functions:
  - Choose hash function randomly from family
  - For arbitrary distribution on inputs => All cells are roughly equal (in expectation)
Strong Universality

- $H(n,m,r)$: Family of $r$-universal hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$ ($2^n$ elements to $2^m$ cells)
  - $r$: degree of independence of hashed inputs

- Higher $r$ => Stronger guarantee on range of size of cells

- $r$-wise universality => Polynomials of degree $r-1$

- Stronger universality => Higher complexity
Hashing-based Approaches

Solution space

n-universal hashing

All cells required to be small

Uniform Generation

BGP Algorithm (Bellare et al, 2000)
Scaling to \(\sim 0.8M\) Variables

From tens of variables to \(\sim 0.8M\) variables!

BGP Algorithm

UniGen

All cells are small

Only a randomly chosen cell needs to be “small”

Uniform Generation

Almost-Uniform Generation
Underlying Hash Functions

- A cell can be represented as the conjunction of:
  - Input formula F
  - $m$ random XOR constraints

- $2^m$ is the number of cells desired

- Use CryptoMiniSAT for CNF + XOR formulas
Strong Theoretical Guarantees

- Uniformity

\[ \Pr[y \text{ is output}] = \frac{1}{|R_F|} \]

- Almost-Uniformity

\[ \forall y \in R_F, \frac{1}{(1 + \varepsilon)|R_F|} \leq \Pr[y \text{ is output}] \leq (1 + \varepsilon)\frac{1}{|R_F|} \]

- UniGen succeeds with probability $> 0.52$
2-3 Orders of Magnitude Faster

Timeout: 18000 seconds

Time(s)

Benchmarks

UniGen
XORSample'
Results: Uniformity

- Benchmark: case110.cnf;  #var: 287;  #clauses: 1263
- Total Runs: $4 \times 10^6$; Total Solutions: $16384$
Results: Uniformity

- Benchmark: case110.cnf; \#var: 287; \#clauses: 1263
- Total Runs: $4 \times 10^6$; Total Solutions: 16384
Roadmap

- SAT Sampling
- SAT Counting
- Works inspired from core ideas
- Future Directions
What is SAT Counting?

- Given a SAT formula $F$
- $R_F$: Set of all solutions of $F$
- Problem ($\#SAT$): Estimate the number of solutions of $F$ ($\#F$) i.e., what is the cardinality of $R_F$?
- E.g., $F = (a \lor b)$
- $R_F = \{(0,1), (1,0), (1,1)\}$
- The number of solutions ($\#F$) = 3

$\#P$: The class of counting problems for decision problems in NP!
Practical Applications

Wide range of applications!

- Estimating coverage achieved
- Probabilistic reasoning/Bayesian inference
- Planning with uncertainty
- Multi-agent/ adversarial reasoning

[Roth 96, Sang 04, Bacchus 04, Domshlak 07]
Counting through Partitioning
Counting through Partitioning

Pick a random cell

Total # of solutions = #solutions in the cell * total # of cells
Strong Theoretical Results

ApproxMC (CNF: F, tolerance: $\varepsilon$, confidence: $\delta$)

Suppose ApproxMC($F, \varepsilon, \delta$) returns $C$. Then,

$$Pr\left[\frac{|R_F|}{1 + \varepsilon} \leq C \leq (1 + \varepsilon)|R_F|\right] \geq \delta$$

ApproxMC runs in time polynomial in log $(1-\delta)^{-1}$, $|F|, \varepsilon^{-1}$ relative to SAT oracle
Mean Error: Only 4% (\(\varepsilon: 0.75\))

Mean error: 4% – much smaller than the theoretical guarantee of 75%
Roadmap

- SAT Sampling
- Model Counting
- Extensions
- Future Directions
Extensions

- Improved performance of UniGen
  - Distributed sampling after preprocessing
  - ~20 SAT calls per sample

- Weighted (non-uniform) sampling and counting
Roadmap

- SAT Sampling
- Model Counting
- Extensions
- Future Directions
Extension to More Expressive Domains (SMT, CSP, ASP)

- Efficient strongly-universal hashing schemes
- Solvers to handle F + Hash efficiently
  - CryptoMiniSAT has fueled progress for SAT domain
  - Similar solvers for other domains?
Key Takeaways

- Constrained sampling and counting are fundamental problems with wide variety of applications
- Prior methods failed to scale or offered very weak theoretical guarantees
- **UniGen**: The first scalable generator with theoretical guarantees of almost-uniformity
- **ApproxMC**: The first scalable approximate SAT counter
- Extensions of underlying techniques in different contexts
Backup Slides
Can Solve a Large Class of Problems

Large class of problems that lie beyond the exact counters but can be computed by ApproxMC
Exploring CNF+XOR

- Very little understanding as of now

- Eager/Lazy approach for XORs?

- How to reduce size of XORs further?
Weighted Counting

Weighted Counting

Given
- CNF Formula $F$
- Weight Function $W$ over assignments

Problem
- What is the sum of weights of *satisfying* assignments?

Example
- $F = (a \lor b)$
- $W([0,1]) = W([1,0]) = 1/3 \quad W([1,1]) = W([0,0]) = 1/6$
- $W(F) = 1/3 + 1/3 + 1/6 = 5/6$
Partition into (weighted) equal "small" cells
 Partition into (weighted) equal “small” cells

Pick a random cell

Pick (by weight) a random solution from this cell
Can you always achieve partitioning?

What if one solution dominates the entire solution space

Tilt = \( \frac{w_{\text{max}}}{w_{\text{min}}} \)

Small tilt \( \rightarrow \) All solutions contribute
How to handle large tilt?

Tilt = 992
Handling Large Tilt

Can be achieved with Pseudo-Boolean Solver
Still a SAT problem not Optimization