SAT Sampling and Counting: From Theory to Practice

Kuldeep Meel
Rice University

M.S. Thesis: Sampling Techniques for Boolean Satisfiability (2014)
Advisors: Moshe Vardi (Rice) and Supratik Chakraborty (IIT Bombay)
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 space!
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. \( 232 \times_{32} a + b != 1100 \)
  2. \( 1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200 \)

- Test vectors: solutions of constraints
Problem: How can we uniformly sample the values of a and b satisfying the above constraints?

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. $1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200$
Problem Formulation

Set of Constraints

SAT Formula

Sample satisfying assignments uniformly at random

SAT Sampling
Roadmap

- SAT Sampling
- Model Counting
- Works inspired from core ideas
- Future Directions
Diverse Applications

Search-based Synthesis

Constrained Random Simulation

SAT Sampling

Probabilistic Inference

Planning under uncertainty

Automatic Problem Generation
Prior Work

- BGP
- BDD
- UniGen
- MCMC
- SAT-Based

Guarantees vs. Performance
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?

r-Universal Hashing
[Carter-Wegman 1979]
Universal Hashing

- Hash functions: mapping \( \{0,1\}^n \) to \( \{0,1\}^m \)
  - \((2^n \text{ elements to } 2^m \text{ cells})\)
- Random inputs => All cells are \textit{roughly} equal (in \textit{expectation})

- Universal family of hash functions:
  - Choose hash function randomly from family
  - For \textit{arbitrary} distribution on inputs => All cells are \textit{roughly equal} (in \textit{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

n-universal hashing

All cells required to be small

Uniform Generation

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

From tens of variables to ~0.8M variables!

BGP Algorithm

UniGen

All cells required to be small

Uniform Generation

Only a randomly chosen cell needs to be “small”

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 (Previous best known: 0.125)
2-3 Orders of Magnitude Faster

Timeout: 18000 seconds

Time(s)

Benchmarks

- case47
- case3_b14_3
- case105
- case8
- case203
- case145
- case61
- case9
- case140
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
- case145
- case61
- case9
- case15
- case140
- case203
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
- Model Counting
- Works inspired from core ideas
- Future Directions
What is Model 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% (ε: 0.75)

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

- SAT Sampling
- Model Counting
- Works inspired from core ideas
- Future Directions
Extensions

2015
WeightCount
IJCAI 2015

2014
WeightMC
AAAI 2014

WeightMC
AAAI 2014

MIS
CP 2015 (in submission)

WeightGen
AAAI 2014

UniGen2
TACAS 2015

UniGen
DAC 2014

M.S. Thesis

ApproxMC
CP 2013

UniWit
CAV 2013

Sampling Techniques
Applications and follow up

- Quantified Information flow (Fremont et al, 2014)

- Hashing-based integration (Ermon et al, 2014)

- Control Improvisation (Fremont et al, 2014)

- Probabilistic programming (Chistikov et al, 2015)
Roadmap

- SAT Sampling
- Model Counting
- Works inspired from core ideas
- Future Directions
Extension to More Expressive Domains (SMT, CSP, ASP)

- Efficient 3-universal/2-universal hashing schemes

- Solvers to handle F + Hash efficiently
  - CryptoMiniSAT has fueled progress for SAT domain
  - Similar solvers for other domains?
Deeper understanding of hashing

- Improved works on sampling require 3-universal hash functions while 2-universal is sufficient for counting.
- Sampling and counting are inter-reducible via Jerrum, Valiant & Vazirani (1986).
Key Takeaways

- 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 model counter
- Extensions of underlying techniques in different contexts
Acknowledgements

- **Advisors**
  - Moshe Vardi (Rice)

- **Collaborators**
  - Daniel Fremont (UCB)
  - Dror Fried (Rice)
  - Alexander Ivrii (IBM, Haifa)
  - Sharad Malik (Princeton)
  - Sanjit Seshia (UCB)

- Supratik Chakraborty (IITB)
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