Sampling Techniques for Constraint Satisfaction and Beyond

Kuldeep S Meel$^2$

(Joint work with Supratik Chakraborty$^1$, Moshe Y Vardi$^2$)

Part of this work has been published in CAV 2013 and CP 2013

$^1$Indian Institute of Technology Bombay, India
$^2$Department of Computer Science, Rice University
Life in The 21\textsuperscript{st} Century!

How do we guarantee that the systems work correctly?
Motivating Example

How do we verify that this circuit works?

• Try for all values of a and b
  • \(2^{128}\) possibilities (\(10^{22}\) years)
  • Not scalable

• Randomly sample some a’s and b’s
  • Wait! None of the circuits in the past faulted when \(10 < b < 40\)
  • Finite resources!

• Let’s sample from regions where it is likely to fault
Designing Verification Scenarios

Designing Constraints

- Designers:
  1. $100 < b < 200$
  2. $300 < a < 451$
  3. $40 < a < 50$ and $30 < b < 40$

- Past Experience:
  1. $400 < a < 2000$
  2. $120 < b < 230$

- Users:
  1. $1000 < a < 1100$
  2. $20000 < b < a < 22000$

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

Given a SAT formula, can one uniformly sample solutions without enumerating all solutions while scaling to real world problems?

Scalable Uniform Generation of SAT-Witnesses
Outline

- Uniform Generation of SAT-witnesses
- Approximate Model Counting
- Future Directions
Outline

- Uniform Generation of SAT-witnesses
- Approximate Model Counting
- Future Directions
Prior Work

**Theoretical Work**
- **Guarantees**: strong
- **Performance**: weak
  
  *BGP Algorithm*  
  (Bellare, Goldreich & Petrank, 98)

**Heuristic Work**
- **Guarantees**: weak
- **Performance**: strong

*XORSample'*
  (Gomes, Sabhrawal & Selman, 07)

**Prior Work**
- **Guarantees**: weak
- **Performance**: strong

*Heuristic Work*  
*Bellare, Goldreich & Petrank, 98*

**Prior Work**
- **Guarantees**: strong
- **Performance**: weak

*Theoretical Work*

**INDUSTRY**
- BDD-based
  - Poor performance

**ACADEMIA**
- SAT-based heuristics
  - No guarantees
Our Contribution

**BGP Algorithm**
(Bellare, Goldreich & Petrank, 98)

**XORSample’**
(Gomes, Sabhrawal & Selman, 07)

---

**Theoretical Work**

- **Guarantees:** strong
- **Performance:** weak

---

**Heuristic Work**

- **Guarantees:** weak
- **Performance:** strong

---

**UniGen**

- **Guarantees:** strong
- **Performance:** strong

---

**INDUSTRY**

- **BDD-based**
  - Poor performance

---

**ACADEMIA**

- **SAT-based heuristics**
  - No guarantees
Central Idea
Partitioning into equal “small” cells
Partitioning into equal “small” 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, Sipser 1983]
Universal Hashing

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

- Random inputs \( \rightarrow \) All cells are *roughly* small

- Universal hash functions:
  - Arbitrary distribution on inputs \( \rightarrow \) All cells are *roughly* small

- Need stronger bounds on distribution of the size of cells
Universality v/s Complexity

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

- Higher the $r$ $\Rightarrow$ Stronger guarantees on distribution of size of cells

- $r$-wise universality $\Rightarrow$ Polynomials of degree $r-1$

- Lower universality $\Rightarrow$ lower complexity
Hashing-Based Approaches

n-universal hashing

BGP Algorithm

All cells should be small

Uniform Generation
Scaling to Thousands of Variables

n-universal hashing

BGP Algorithm

All cells should be small

Uniform Generation

Solution space

2-universal hashing

Random

RF : Solution space

Our Approach

Small :

uniGen

Only a randomly chosen cells needs to be “small”

Near-Uniform Generation
Scaling to Thousands of Variables

From tens of variables to thousands of variables!

BGP Algorithm

Universe

Solution space

Uniform Generation

UniGen

All cells should be small

Only a randomly chosen cells needs to be “small”

Near-Uniform Generation
Employs XOR-based hash functions instead of computationally infeasible algebraic hash functions

Uses off-the-shelf SAT solver CryptoMiniSAT (MiniSAT+XOR support)
Strong Theoretical Guarantees

- Uniformity

For every solution $y$ of $R_F$

$$\Pr [y \text{ is output}] = \frac{1}{|R_F|}$$
Strong Theoretical Guarantees

- Near Uniformity

  For every solution $y$ of $R_F$
  \[ \Pr [y \text{ is output}] \geq \frac{1}{8} \times \frac{1}{|R_F|} \]

- Success Probability

  Algorithm UniWit succeeds with probability at least $1/8$

- Polynomial calls to SAT Solver
Experimental Methodology

- Benchmarks (over 300)
  - Bit-blasted versions of word-level constraints from VHDL designs, SMTLIB, ISCAS’85
  - Bit-blasted versions from program synthesis
  - Largest benchmark with 486,193 variables

- Objectives
  - Comparison with algorithms BGP & XORSample’
    - Uniformity
    - Performance
Results: Uniformity

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

![Bar chart showing performance results for UniGen and XORSample'](image-url)
2-3 Orders of Magnitude Faster

- UniWit is 2-3 orders of magnitude faster than XORSample.
- Observed success probability = 0.6 (> theoretical guarantee of 0.125)
The Story So Far

- Theoretical guarantees of almost uniformity
- Major improvements in running time and uniformity compared to existing generators
- But . . . .
  How many samples should I test my system to achieve desired coverage?
- Are $10^5$ samples enough?
  - Case A: Total solutions - $10^6$
  - Case B: Total solutions - $10^{60}$
What is the total number of satisfying assignments to system of constraints?
Outline

- Uniform Generation of SAT-witnesses
- Approximate Model Counting
- 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

Exciting range of applications!

- Probabilistic reasoning/Bayesian inference
- Planning with uncertainty
- Multi-agent/ adversarial reasoning
  [Roth 96, Sang 04, Bacchus 04, Domshlak 07]
But it is hard!

- #SAT is #P-complete
  - Even for counting solutions of 2-CNF SAT

- #P is really hard!
  - Believed to be much harder than NP-complete problems
  - $\text{PH} \subseteq \text{P}^\#P$
## Prior Work

**Input Formula:** $F$;  **Total Solutions:** $\#F$;  **Return Value:** $C$

<table>
<thead>
<tr>
<th>Counters</th>
<th>Guarantee</th>
<th>Confidence</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact counter (e.g. sharpSAT, Cachet)</td>
<td>$C = #F$</td>
<td>1</td>
<td>Poor Scalability</td>
</tr>
<tr>
<td>Lower bound counters (e.g. MBound, SampleCount)</td>
<td>$C \leq #F$</td>
<td>$\delta$</td>
<td>Very weak guarantees</td>
</tr>
<tr>
<td>Upper bound counters (e.g. MiniCount)</td>
<td>$C \geq #F$</td>
<td>$\delta$</td>
<td>Very weak guarantees</td>
</tr>
</tbody>
</table>
Design an approximate model counter $G$:

- **inputs:**
  - CNF formula $F$
  - tolerance $\varepsilon$
  - confidence $\delta$

- the count returned by it is within $\varepsilon$ of the $\#F$ with confidence at least $\delta$
Approximate Model Counting

Design an approximate model counter $G$:

- inputs:
  - CNF formula $F$
  - tolerance $\varepsilon$
  - confidence $\delta$

- the count returned by it is within $\varepsilon$ of the $\#F$ with confidence at least $\delta$ and scales to real world problems

Scalable Approximate Model Counting

Lies in the 2\textsuperscript{nd} level of Polynomial hierarchy: $\Sigma_2^P$
Our Contribution

Input Formula: $F$; Total Solutions: $\#F$

<table>
<thead>
<tr>
<th>Counters</th>
<th>Guarantee</th>
<th>Confidence</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact counter (e.g. sharpSAT, Cachet)</td>
<td>$C = #F$</td>
<td>1</td>
<td>Poor Scalability</td>
</tr>
<tr>
<td>ApproxMC</td>
<td>$\frac{#F}{(1+\varepsilon)\delta} \leq C \leq (1+\varepsilon)#F$</td>
<td>$\delta$</td>
<td>Scalability + Strong guarantees</td>
</tr>
</tbody>
</table>

The First Scalable Approximate Model Counter
How do we count?
Naïve Enumeration: Not Scalable

- Enumerate all solutions
- Exact Counting!
- Cachet, Relsat, sharpSAT

Not Scalable! (Think of enumerating $2^{100}$ solutions)
Counting through Partitioning
Counting through Partitioning

Pick a random cell

Total # of solutions = #solutions in the cell * total # of cells
Algorithm in Action
Algorithm in Action

Algorithm

Median

690  710  730  730  731  831

834

\[ \hat{t} \]
Partitioning

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

Linear hash functions (2-universal hash functions)
Strong Theoretical Results

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

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

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

ApproxMC runs in time polynomial in $\log (1-\delta)^{-1}$, $|F|$, $\varepsilon^{-1}$ relative to SAT oracle
Experimental Methodology

- **Benchmarks (over 200)**
  - Grid networks, DQMR networks, Bayesian networks
  - Plan recognition, logistics problems
  - Circuit synthesis

- **Tolerance:** \( \varepsilon = 0.75 \), **Confidence:** \( \delta = 0.9 \)

- **Objectives**
  - Comparison with exact counters (Cachet) & bounding counters (MiniCount, Hybrid-MBound, SampleCount)
    - Performance
    - Quality of bounds
Results: Performance Comparison

- ApproxMC
- Cachet
Results: Performance Comparison

The graph compares the performance of ApproxMC and Cachet. The x-axis represents the time in steps, while the y-axis shows the performance metric. ApproxMC shows a more volatile performance with frequent spikes, whereas Cachet maintains a more stable path towards higher performance values.
Can Solve a Large Class of Problems

Large class of problems that lie beyond the exact counters but can be computed by ApproxMC
Mean Error: Only 4% (allowed: 75%)

Mean error: 4% — much smaller than the theoretical guarantee of 75%
Key Takeaways

- Prior work either offered no/weak guarantees or poor performance
- Limited independence hash functions for partitioning

- Our Technique provides
  - Scalability
  - Theoretical guarantees of almost uniformity (UniGen)
  - The first approximate model counter (ApproxMC)

- Tools are available online! Go and Try them out!
UniGen: Uniform generator for the next Generation

- Efficient hash functions
  - With smaller XOR lengths
  - Scales to hundreds of thousands of variables
- Stronger guarantees

For every solution $y$ of $R_F$

$$\frac{1}{8} \times \frac{1}{|R_F|} \leq \Pr[y \text{ is output}]$$
Looking Forward

- **UniGen**: Uniform generator for the next Generation
  - Efficient hash functions
    - With smaller XOR lengths
    - Scales to hundreds of thousands of variables
  - Stronger guarantees

For every solution $y$ of $R_F$

$$\frac{1}{(2.7) \times 1/|R_F|} \leq \Pr[y \text{ is output}] \leq 2.7 \times \frac{1}{|R_F|}$$

- Extension to other domains: SMT
- Distribution-aware sampling and counting
Discussion

Acknowledgments

• NSF
• ExCAPE
• Intel
• BRNS, India
• Sun Microsystems
• Sigma Solutions, Inc

Thank You for your attention!
Range of count from bounding counters = $C_2 - C_1$
- $C_1$: From lower bound counters (MBound/SampleSAT)
- $C_2$: From upper bound counters (MiniCount)

Range from ApproxMC: $[C/(1+\varepsilon), (1+\varepsilon)C]$

Smaller the range, better the algorithm!
ApproxMC improved the upper bounds significantly while also improving the lower bounds.