Constrained Sampling and Counting: When Practice Drives Theory

Supratik Chakraborty
IIT Bombay

Joint work with Kuldeep Meel and Moshe Y. Vardi
(Rice University)
How do we infer useful information from the data filled with uncertainty?

Modeling Attendance for Today’s Talk

Roth, 1996
Smart Cities

• Alarm system in every house that responds to either burglary or earthquake

• Every alarm system is connected to the central dispatcher (of course, automated!)

• Suppose one of the alarm goes off

• Important to predict whether its earthquake or burglary
What is the probability of earthquake \((E)\) given that alarm sounded \((A)\)?

\[
\Pr[\text{event} \mid \text{evidence}]
\]

Bayes’ rule to the rescue

\[
\Pr[E \mid A] = \frac{\Pr[E \cap A]}{\Pr[A]}
\]

How do we calculate these probabilities?
Probabilistic Models

Graphical Models
## Graphical Models

<table>
<thead>
<tr>
<th>$B$</th>
<th>$Pr$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.8</td>
</tr>
<tr>
<td>$F$</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$E$</th>
<th>$Pr$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.1</td>
</tr>
<tr>
<td>$F$</td>
<td>0.9</td>
</tr>
</tbody>
</table>

### Table 1: Pr(A|E, B)

| $B$ | $E$ | $A$ | $Pr(A|E, B)$ |
|-----|-----|-----|-------------|
| $T$ | $T$ | $T$ | 0.3         |
| $T$ | $T$ | $F$ | 0.7         |
| $T$ | $F$ | $T$ | 0.4         |
| $T$ | $F$ | $F$ | 0.6         |
| $F$ | $T$ | $T$ | 0.2         |
| $F$ | $F$ | $F$ | 0.8         |
| $F$ | $F$ | $T$ | 0.1         |
Calculating $\Pr[E \cap A]$

<table>
<thead>
<tr>
<th>$B$</th>
<th>$\Pr$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.8</td>
</tr>
<tr>
<td>$F$</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$E$</th>
<th>$\Pr$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.1</td>
</tr>
<tr>
<td>$F$</td>
<td>0.9</td>
</tr>
</tbody>
</table>

| $B$ | $E$ | $A$ | $\Pr(A|E, B)$ |
|-----|-----|-----|--------------|
| $T$ | $T$ | $T$ | 0.3          |
| $T$ | $T$ | $F$ | 0.7          |
| $T$ | $F$ | $T$ | 0.4          |
| $T$ | $F$ | $F$ | 0.6          |
| $F$ | $T$ | $T$ | 0.2          |
| $F$ | $F$ | $F$ | 0.8          |
| $F$ | $F$ | $T$ | 0.1          |

$\Pr[E \cap A] = \Pr[E] \times \Pr[\neg B] \times \Pr[A|E, \neg B] + \Pr[E] \times \Pr[B] \times \Pr[A|E, B]$
Calculating $\Pr[E \cap A]$

\[ \Pr[E \cap A] = \Pr[E] \times \Pr[B] \times \Pr[A|E, B] + \Pr[E] \times \Pr[\neg B] \times \Pr[A|E, \neg B] \]
Calculating $\Pr[E \cap A]$

<table>
<thead>
<tr>
<th>$B$</th>
<th>$Pr$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.8</td>
</tr>
<tr>
<td>$F$</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$E$</th>
<th>$Pr$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.1</td>
</tr>
<tr>
<td>$F$</td>
<td>0.9</td>
</tr>
</tbody>
</table>

| $B$ | $E$ | $A$ | $Pr(A|E, B)$ |
|-----|-----|-----|-------------|
| $T$ | $T$ | $T$ | 0.3         |
| $T$ | $T$ | $F$ | 0.7         |
| $T$ | $F$ | $T$ | 0.4         |
| $T$ | $F$ | $F$ | 0.6         |
| $F$ | $T$ | $T$ | 0.2         |
| $F$ | $F$ | $F$ | 0.8         |
| $F$ | $F$ | $T$ | 0.1         |

$$
\Pr[E \cap A] = \Pr[E] \times \Pr[B] \times \Pr[A|E, B] \\
+ \Pr[E] \times \Pr[\neg B] \times \Pr[A|E, \neg B]
$$
Moving from Probability to Logic

- $X = \{A, B, E\}$
- $F = E \land A$
- $W(B = 0) = 0.2, W(B = 1) = 1 - W(B = 0) = 0.8$
- $W(A = 0) = 0.1, W(A = 1) = 0.9$
- $W(E = 0| A = 0, B = 0) = \ldots$
- $W(A = 1, E = 1, B = 1) = W(B = 1) \ast W(E = 1) \ast W(A = 1|E = 1, B = 1)$
- $R_F = \{(A = 1, E = 1, B = 0), (A = 1, E = 1, B = 1)\}$
- $W(F) = W(A = 1, E = 1, B = 1) + W(A = 1, E = 1, B = 1)$

$W(F) = Pr[E \land A]$
Probabilistic Inference to WMC to Unweighted Model Counting

Weighted Model Counting $\rightarrow$ Unweighted Model Counting

Polynomial time reductions

Roth, 1996
Model Counting

• Given a SAT formula F
• \(R_F\): Set of all solutions of F
• Problem (\(#\text{SAT}\))\: 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!
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!
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. \( 1020 <_{64} (b /_{64} 2) +_{64} a <_{64} 2200 \)

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

Scalable Uniform Generation of SAT Witnesses
Agenda

Design **Scalable** Techniques for Uniform Generation and Model Counting with **Strong** Theoretical Guarantees
Agenda

Design **Scalable** Techniques for Almost-Uniform Generation and Approximate-Model Counting with **Strong Theoretical Guarantees**
Formal Definitions

- $F$: CNF Formula; $R_F$: Solution Space of $F$
- Input: $F$  
  Output: $y \in R_F$

- Uniform Generator:
  - Guarantee: $\forall y \in R_F$, $\Pr[y$ is output$] = \frac{1}{|R_F|}$
- Almost-Uniform Generator
  - Guarantee: $\forall y \in R_F$, $\frac{1}{(1+\epsilon)|R_F|} \leq \Pr[y$ is output$] \leq \frac{(1+\epsilon)}{|R_F|}$
Formal Definitions

- $F$: CNF Formula;  $R_F$: Solution Space of $F$

- Probably Approximately Correct (PAC) Counter
  - Input: $F$  
  - Output: $C$

\[
\text{Pr} \left[ \frac{|R_F|}{(1 + \varepsilon)} \leq C \leq |R_F|(1 + \varepsilon) \right] \geq 1 - \delta
\]
Uniform Generation
Rich History of Theoretical Work

- Jerrum, Valiant and Vazirani (1986):
  - Uniform Generator: Polynomial time PTM (Probabilistic Turing Machine) given access to $\Sigma^P_2$ oracle

Can be used to design a BPP$^\text{NP}$ procedure -- too large NP instances

No Practical Algorithms
Rich History of Theoretical Work

• Bellare, Goldreich, and Petrank (2000)
  • Uniform Generator: Polynomial time PTM given access to NP oracle
  • Employs n-universal hash functions
Universal Hashing

- \( H(n, m, r) \): Set of \( r \)-universal hash functions from \( \{0,1\}^n \rightarrow \{0,1\}^m \)

\[ \forall y_1, y_2, \ldots, y_r \text{ (distinct)} \in \{0,1\}^n \text{ and } \forall \alpha_1, \alpha_2 \ldots, \alpha_r \in \{0,1\}^m \]

\[ \Pr[h(y_i = \alpha_i)] = \frac{1}{2^m} \quad \text{(Uniformity)} \]

\[ \Pr[ h(y_1 = \alpha_1) \land \cdots \land (h(y_r) = \alpha_r) ] = 2^{-(mr)} \quad \text{(Independence)} \]

- \((r-1)\) degree polynomials \(\rightarrow\) \(r\)-universal hash functions
Concentration Bounds

• t-wise \((t \geq 4)\) random variables \(X_1, X_2, \ldots, X_n \in [0,1]\)

\[
X = \sum X_i \ ; \ \mu = E[X]
\]

\[
\Pr[|X - \mu| \leq A] \geq 1 - 8 \left(\frac{t\mu + t^2}{A^2}\right)^t
\]

• For \(t = 2\)

\[
\Pr[|X - \mu| \leq A] \geq 1 - \frac{\sigma^2[X]}{A^2}
\]
BGP Method

- Choose $m$
- Choose $h \in H(n, m, n)$

• For right choice of $m$, all the cells are small ($\#$ of solutions $\leq 2n^2$)
• Check if all the cells are small (NP-Query)
• If yes, pick a solution randomly from randomly picked cell

In practice, the query is too long and can not be handled by SAT Solvers!
To Recap

• **Jerrum, Valiant and Vazirani (1986):**
  - Uniform Generator: Polynomial time PTM given access to $\Sigma_2^P$ oracle
  - Almost-Uniform Generation is inter-reducible to PAC counting

• **Bellare, Goldreich, and Petrank (2000)**
  - Uniform Generator: Polynomial time PTM given access to NP oracle

Does not work in practice!
Prior Work

Guarantees

- BGP
- BDD

Performance

- MCMC
- SAT-Based
Desires

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: XORSample’</td>
<td>50000</td>
</tr>
<tr>
<td>Ideal Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
</tbody>
</table>

Experiments over 200+ benchmarks
*: According to EDA experts
Our Contribution

Guarantees

BGP
BDD

Performance

UniGen

MCMC
SAT-Based
Key Ideas

- For right choice of m, large number of cells are “small”
  - “almost all” the cells are “roughly” equal
- Check if a randomly picked cell is “small”
- If yes, pick a solution randomly from randomly picked cell
Key Challenges

• F: Formula     X: Set of variables     R_F: Solution space

• R_{F,h,\alpha}: Set of solutions for F \land (h(X) = \alpha) where
  • h \in H(n, m, *) ; \alpha \in \{0,1\}^m

1. How large is “small” cell?
2. How much universality do we need?
3. What is the value of m?
Size of cell

Pr[ y is output ] = \( \frac{1}{2^m} \cdot \Pr[ \text{Cell is small} | \ y \ is \ in \ the \ cell] \cdot \frac{1}{\text{Size of cell}} \)

Let Size of cell \( \in [loThresh, hiThresh] \), Then:

\[
\frac{1}{2^m} \cdot q \cdot \frac{1}{hiThresh} \leq \Pr[ y \ is \ output ] \leq \frac{1}{2^m} \cdot q \cdot \frac{1}{loThresh}
\]

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

\[hiThresh = (1 + \varepsilon) \cdot pivot; \quad loThresh = \frac{pivot}{1 + \varepsilon}\]

\[pivot = k \left( 1 + \frac{1}{\varepsilon^2} \right)\]
Losing Independence

Our desire:

\[ \Pr \left[ \text{loThresh} \leq |R_{F,h,\alpha}| \leq \text{hiThresh} \right] \geq p \left( \geq \frac{1}{2} \right) \]

\[ \Pr \left[ \frac{\text{pivot}}{1 + \varepsilon} \leq |R_{F,h,\alpha}| \leq (1 + \varepsilon)\text{pivot} \right] \geq p \left( \geq \frac{1}{2} \right) \]

Suppose \( h \in H(n, m, \ast) \) and \( m = \log \frac{|R_F|}{\text{pivot}} \)

Then, \( E[|R_{F,h,\alpha}|] = \frac{|R_F|}{2^m} = \text{pivot} \)

Concentration bound \( \Rightarrow \) \( k \)-universal (small constant)
How many cells?

• Our desire: \( m = \log \frac{|R_F|}{\text{pivot}} \)
  • But determining \(|R_F|\) is expensive (\#P complete)

• How about approximation?
  • \( \text{ApproxMC}(F, \varepsilon, \delta) \) returns \( C: \)
    \[
    \Pr\left[ \frac{|R_F|}{1+\varepsilon} \leq C \leq (1 + \varepsilon)|R_F| \right] \geq 1 - \delta
    \]
  • \( q = \log C - \log \text{pivot} \)
  • Concentrate on \( m = q-1, q, q+1 \)
UniGen(F, \varepsilon)

1. C = \text{ApproxMC}(F, \varepsilon) \quad \text{One time execution}
2. Compute pivot, loThresh, hiThresh
3. q = \log|C| - \log\text{pivot}
4. for i in \{q-1, q, q+1\}:
   5. Choose h \textbf{randomly*} from H(n,i,3)
   6. Choose \alpha randomly from \{0,1\}^m
   7. If (loThresh \leq |R_{F,h,\alpha}| \leq hiThresh):
      8. Pick \ y \in R_{F,h,\alpha} \ \text{randomly}
Are we back to JVV?

NOT Really

• JVV makes linear (in n) calls to Approximate counter compared to just 1 in UniGen

• # of calls to ApproxMC is only 1 regardless of the number of samples required unlike JVV
PAC Counter: ApproxMC(F, ε, δ)

- For right choice of m, large number of cells are “small”
  - “almost all” the cells are “roughly” equal
- Check if a randomly picked cell is “small”
- If yes, then estimate = # of solutions in cell * 2^m
$\text{ApproxMC}(F, \varepsilon, \delta)$
ApproxMC(F, \varepsilon, \delta)
ApproxMC(F, ε, δ)

Estimate:
# of sols * 2^m

#sols < pivot

YES
ApproxMC(F, ε, δ)

Key Lemmas

Let $m^* = \log|R_F| - \log\text{pivot}$

Lemma 1: The algorithm terminates with $m \in [m^* - 1, m^*]$ with high probability

Lemma 2: The estimate from a randomly picked cell for $m \in [m^* - 1, m^*]$ is correct with high probability
Results: Performance Comparison

![Graph showing performance comparison between ApproxMC and Cachet]

- X-axis: Benchmarks
- Y-axis: Time (seconds)
Results: Performance Comparison
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%
Runtime Performance of UniGen
1-2 Orders of Magnitude Faster

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
So far

• The first scalable approximate model counter
• The first scalable uniform generator
• Outperforms state-of-the-art generators/counters

Are we done?
### Where are we?

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: XORSample’</td>
<td>50000</td>
</tr>
<tr>
<td>UniGen</td>
<td>~5000</td>
</tr>
<tr>
<td>Ideal Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
</tbody>
</table>

Experiments over 200+ benchmarks
*: According to EDA experts
XOR-Based Hashing

- Partition $2^n$ space into $2^m$ cells
- Variables: $X_1, X_2, X_3, \ldots, X_n$
- Pick every variable with prob. $\frac{1}{2}$, XOR them and add 0/1 with prob. $\frac{1}{2}$
- $X_1 + X_3 + X_6 + \ldots + X_{n-1} + 0$
- To construct $h$: $\{0,1\}^n \rightarrow \{0,1\}^m$, choose $m$ random XORs
- $\alpha \in \{0,1\}^m \rightarrow$ Set every XOR equation to 0 or 1 randomly
- The cell: $F \land \text{XOR (CNF+XOR)}$
XOR-Based Hashing

- **CryptoMiniSAT**: Efficient for CNF+XOR

- Avg Length : n/2

- Smaller XORs ➔ better performance

How to shorten XOR clauses?
Independent Support

• Set I of variables such that assignments to these uniquely determine assignments to rest of variables (for satisfying assignments)

• If $^1$ and $^2$ agree on I then $^1 = ^2$

• $c \leftrightarrow (a \lor b) ; \text{Independent Support I: } \{a, b\}$

• **Key Idea:** Hash only on the independent variables
Independent Support

- Hash only on the Independent Support
- Average size of XOR: $n/2$ to $|I|/2$
Formal Definition

Input Formula: $F$, Solution space: $R_F$

$\forall \sigma_1, \sigma_2 \in R_F$, If $\sigma_1$ and $\sigma_2$ agree on $I$, then $\sigma_1 = \sigma_2$

$$F(x_1, \ldots, x_n) \land F(y_1, \ldots, y_n) \land \bigwedge_{i \mid x_i \in I} (x_i = y_i) \Rightarrow \bigwedge_j (x_j = y_j)$$

where $F(y_1, \ldots, y_n) = F(x_1 \rightarrow y_1, \ldots, x_n \rightarrow y_n)$
Minimal Unsatisfiable Subset

- Given $\Psi = H_1 \land H_2 \cdots H_m$

- Find subset $\{H_{i_1}, H_{i_2}, \cdots H_{i_k}\}$ of $\{H_1, H_2, \cdots H_m\}$ such that $H_{i_1} \land H_{i_2} \cdots H_{i_k} \land \Omega$ is UNSAT

  Unsatisfiable subset

- Find \textbf{minimal} subset $\{H_{i_1}, H_{i_2}, \cdots H_{i_k}\}$ of $\{H_1, H_2, \cdots H_m\}$ such that $H_{i_1} \land H_{i_2} \cdots H_{i_k}$ is UNSAT

  Minimal Unsatisfiable subset
Key Idea

\[ F(x_1, \ldots, x_n) \land F(y_1, \ldots, y_n) \land \bigwedge_{i \mid x_i \in I} (x_i = y_i) \implies \bigwedge_j (x_j = y_j) \]

\[ Q_{F,I} = F(x_1, \ldots, x_n) \land F(y_1, \ldots, y_n) \land \bigwedge_{i \mid x_i \in I} (x_i = y_i) \land \neg \left( \bigwedge_j (x_j = y_j) \right). \]

Theorem: \( Q_{F,I} \) is unsatisfiable if and only if I is independent support
Key Idea

\[ H_1 = \{x_1 = y_1\}, \ldots, H_n = \{x_n = y_n\} \]

\[ \Omega = F(x_1, \ldots, x_n) \land F(y_1, \ldots, y_n) \land (\neg \bigwedge_j (x_j = y_j)) \]

\( I = \{x_i\} \) is Independent Support iff \( H^I \land \Omega \) is unsatisfiable where \( H^I = \{H_i | x_i \in I\} \)
Group-Oriented Minimal Unsatisfiable Subset

- Given $\Psi = H_1 \land H_2 \cdots H_m \land \Omega$

- Find subset $\{H_{i_1}, H_{i_2}, \cdots H_{i_k}\}$ of $\{H_1, H_2, \cdots H_m\}$ such that $H_{i_1} \land H_{i_2} \cdots H_{i_k} \land \Omega$ is UNSAT

  Group Oriented Unsatisfiable subset

- Find **minimal** subset $\{H_{i_1}, H_{i_2}, \cdots H_{i_k}\}$ of $\{H_1, H_2, \cdots H_m\}$ such that $H_{i_1} \land H_{i_2} \cdots H_{i_k} \land \Omega$ is UNSAT

  Group Oriented Minimal Unsatisfiable subset
Minimal Independent Support

\[ H_1 = \{x_1 = y_1\}, \ldots, H_n = \{x_n = y_n\} \]

\[ \Omega = F(x_1, \ldots, x_n) \land F(y_1, \ldots, y_n) \land (\neg \bigwedge_j (x_j = y_j)) \]

\[ I = \{x_i\} \text{ is minimal Independent Support iff } H^I \text{ is minimal unsatisfiable subset where } H^I = \{H_i \mid x_i \in I\} \]
Key Idea

Minimal Independent Support (MIS) → Minimal Unsatisfiable Subset (MUS)
Impact on Sampling and Counting Techniques
What about complexity

- Computation of MUS: $FP^{NP}$

- Why solve a $FP^{NP}$ for almost-uniform generation/approximate counter (PTIME PTM with NP Oracle)

Settling the debate through practice!
Performance Impact on Approximate Model Counting

![Graph showing performance impact on approximate model counting with bars for approxMC and IApproxMC across various test cases.](Image)
Performance Impact on Uniform Sampling

![Graph showing performance impact on uniform sampling](image)
### Where are we?

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: XORSample’</td>
<td>50000</td>
</tr>
<tr>
<td>UniGen</td>
<td>5000</td>
</tr>
<tr>
<td>UniGen1</td>
<td>470</td>
</tr>
<tr>
<td>Ideal Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
</tbody>
</table>
Back to basics

# of solutions in “small” cell ∈ [lo\text{Thresh}, hi\text{Thresh}]
We pick one solution
“Wastage” of lo\text{Thresh} solutions
Pick lo\text{Thresh} samples!
3-Universal and Independence of Samples

3-Universal hash functions:
  • Choose hash function randomly
  • For arbitrary distribution on solutions => All cells are roughly equal in expectation

  • But:
    • While each input is hashed uniformly
    • And each 3-solutions set is hashed independently
    • A 4-solutions set might not be hashed independently
Balancing Independence

For \( h \in H(n, m, 3) \)

- Choosing up to 3 samples => Full independence among samples

- Choosing loThresh (>> 3) samples => Loss of independence
Why care about Independence

Convergence requires multiplication of probabilities

If every sample is independent => Faster convergence
The principle of principled compromise!

- Choosing up to 3 samples $\Rightarrow$ Full independence among samples

- Choosing loThresh ($>> 3$) samples $\Rightarrow$ Loss of independence
  - “Almost-Independence” among samples
  - Still provides strong theoretical guarantees of coverage
Strong Guarantees

- \( L = \# \text{ of samples} < |R_F| \)

\[
\frac{L}{(1 + \varepsilon)|R_F|} \leq \Pr[y \text{ is output}] \leq 1.02(1 + \varepsilon) \frac{L}{|R_F|}
\]

**Polynomial** Constant number of SAT calls per sample
- After one call to ApproxMC
Bug-finding effectiveness

\[
\text{bug frequency } f = \frac{|B|}{|R_F|}
\]

<table>
<thead>
<tr>
<th>relative number of SAT calls</th>
<th>UniGen</th>
<th>UniGen2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\frac{3 \cdot hiThresh(1+\nu)(1+\varepsilon)}{0.52}$</td>
<td>$\frac{3 \cdot hiThresh}{0.62 \cdot loThresh} \frac{(1+\tilde{\nu})(1+\varepsilon)}{1-\tilde{\nu}}$</td>
</tr>
</tbody>
</table>

Simply put, 
# of SAT calls for UniGen2 << # of SAT calls for UniGen
Bug-finding effectiveness

bug frequency $f = 1/10^4$
find bug with probability $\geq 1/2$

<table>
<thead>
<tr>
<th></th>
<th>UniGen</th>
<th>UniGen2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected number of SAT calls</td>
<td>$4.35 \times 10^7$</td>
<td>$3.38 \times 10^6$</td>
</tr>
</tbody>
</table>

An order of magnitude difference!
~20 times faster than UniGen1

![Benchmark Comparison Graph]

- UniGen2
- UniGen1

**Benchmarks:**
- case_1_b12_2
- s1238a_3_2
- s1196a_3_2
- s832a_15_7
- squaring16
- squaring7
- LoginService2
- Sort
- doublyLinkedList
- enqueue
- Karatsuba
- llreverse
- lltraversal
- diagStencil_new
- tutorial3
- demo2_new

**Time per sample (s):**

- 1000
- 100
- 10
- 1
- 0.1
- 0.01
- 0.001
## Where are we?

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: XORSample’</td>
<td>50000</td>
</tr>
<tr>
<td>UniGen</td>
<td>5000</td>
</tr>
<tr>
<td>UniGen1</td>
<td>470</td>
</tr>
<tr>
<td>UniGen2</td>
<td>20</td>
</tr>
<tr>
<td>Ideal Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
</tbody>
</table>
The Final Push....

- UniGen requires one time computation of ApproxMC
- Generation of samples in fully distributed fashion (Previous algorithms lacked the above property)
- New paradigms!
Current Paradigm of Simulation-based Verification

- Can not be parallelized since test generators maintain “global state”
- Loses theoretical guarantees (if any) of uniformity
New Paradigm of Simulation-based Verification

- Preprocessing needs to be done only once
- No communication required between different copies of the test generator
- Fully distributed!
## Closing in...

<table>
<thead>
<tr>
<th>Generator</th>
<th>Relative runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>State-of-the-art: XORSample’</td>
<td>50000</td>
</tr>
<tr>
<td>UniGen</td>
<td>5000</td>
</tr>
<tr>
<td>UniGen1</td>
<td>470</td>
</tr>
<tr>
<td>UniGen2</td>
<td>20</td>
</tr>
<tr>
<td>Multi-core UniGen2</td>
<td>10 (two cores)</td>
</tr>
<tr>
<td>Ideal Uniform Generator*</td>
<td>10</td>
</tr>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
</tbody>
</table>
So what happened....

Sampling and Counting
Important Applications

New Applications (Theory drives practice)

Beautiful Theory
But does not work in practice

Theoretical Contributions
(Practice drives theory)
Future Directions
Extension to More Expressive domains

• Efficient hashing schemes
  • Extending bit-wise XOR to richer constraint domains provides guarantees but no advantage of SMT progress

• Solvers to handle F + Hash efficiently
  • CryptoMiniSAT has fueled progress for SAT domain
  • Similar solvers for other domains?
Handling Distributions

- Given: CNF formula F and Weight function W over assignments
- Weighted Counting: sum the weight of solutions
- Weighted Sampling: Sample according to weight of solution
- Wide range of applications in Machine Learning
- Extending universal hashing works only in theory so far