Constrained Counting and Sampling:
Bridging the Gap between Theory and Practice

Kuldeep S. Meel

Rice University, USA

www.kuldeepmeel.com
kuldeep@rice.edu
• Data
• Uncertainty
• Data
• Uncertainty
• Data
• Uncertainty
- Data
- Uncertainty
• Data
• Uncertainty
• Scalable
• Guarantees of Accuracy
Boolean Satisfiability (SAT): Given a Boolean expression, using “and” (∧), “or” (∨), and “not” (¬) is there a solution, i.e., an assignment of 0’s and 1’s to the variables that makes the expression equal 1?

Example: \((x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor \neg x_3)\)

\(x_1 = 1, x_2 = 1, x_3 = 1\)
A Tale of Constraints

Boolean Satisfiability (SAT): Given a Boolean expression, using “and” (\(\land\)), “or” (\(\lor\)), and “not” (\(\neg\)) is there a solution, i.e., an assignment of 0’s and 1’s to the variables that makes the expression equal 1?

Example: \((x_1 \land \neg x_2 \lor \neg x_3) \land (x_2 \lor \neg x_3)\)

\(x_1 = 1, x_2 = 1, x_3 = 1\)

Ernst Schröder, 1841-1902: “Getting a handle on the consequences of any premises, or at least the fastest method for obtaining these consequences, seems to me to be one of the noblest, if not the ultimate goal of mathematics and logic.”

Cook, 1971; Levin, 1973: SAT is NP-complete
Modern SAT solvers are able to deal routinely with practical problems that involve many thousands of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)
Modern SAT solvers are able to deal routinely with practical
problems that involve many thousands of variables, although
such problems were regarded as hopeless just a few years ago.
(Donald Knuth, 2016)

Industrial usage of SAT Solvers: hardware verification, planning,
Genome Rearrangement, Telecom Feature Subscription, Resource
Constrained Scheduling, Noise Analysis, Games, · · ·
Modern SAT solvers are able to deal routinely with practical problems that involve many thousands of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)

Industrial usage of SAT Solvers: hardware verification, planning, Genome Rearrangement, Telecom Feature Subscription, Resource Constrained Scheduling, Noise Analysis, Games, · · ·

Now that SAT is “easy”, it is time to look beyond satisfiability
Constrained Counting and Sampling

• **Given**
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula $F$ over $X_1, X_2, \cdots X_n$

• $\text{Sol}(F) = \{ \text{solutions of } F \}$

• **Constrained Counting**: Determine $|\text{Sol}(F)|$

• **Constrained Sampling**: Randomly sample from $\text{Sol}(F)$ such that
  $$\Pr[y \text{ is sampled}] = \frac{1}{|\text{Sol}(F)|}$$
Constrained Counting and Sampling

- **Given**
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula $F$ over $X_1, X_2, \cdots X_n$
  - Weight Function $W : \{0, 1\}^n \mapsto [0, 1]$

- $\text{Sol}(F) = \{ \text{solutions of } F \}$

- $W(F) = \sum_{y \in \text{Sol}(F)} W(y)$

- **Constrained Counting:** Determine $W(F)$

- **Constrained Sampling:** Randomly sample from $\text{Sol}(F)$ such that
  \[
  \Pr[y \text{ is sampled}] = \frac{W(y)}{W(F)}
  \]
Constrained Counting and Sampling

- **Given**
  - Boolean variables $X_1, X_2, \ldots, X_n$
  - Formula $F$ over $X_1, X_2, \ldots, X_n$
  - Weight Function $W: \{0, 1\}^n \mapsto [0, 1]$

- **Sol($F$)** = \{ solutions of $F$ \}

- $W(F) = \sum_{y \in \text{Sol}(F)} W(y)$

- **Constrained Counting**: Determine $W(F)$

- **Constrained Sampling**: Randomly sample from Sol($F$) such that
  $\Pr[y \text{ is sampled}] = \frac{W(y)}{W(F)}$

- $F := (X_1 \vee X_2)$;
  $W[(0, 0)] = W[(1, 1)] = \frac{1}{6}; W[(1, 0)] = W[(0, 1)] = \frac{1}{3}$

- **Sol($F$)** = \{(0, 1), (1, 0), (1, 1)\}
Constrained Counting and Sampling

- **Given**
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula $F$ over $X_1, X_2, \cdots X_n$
  - Weight Function $W: \{0, 1\}^n \mapsto [0, 1]$

- $\text{Sol}(F) = \{ \text{solutions of } F \}$
- $W(F) = \sum_{y \in \text{Sol}(F)} W(y)$

- **Constrained Counting**: Determine $W(F)$
- **Constrained Sampling**: Randomly sample from $\text{Sol}(F)$ such that $\Pr[y \text{ is sampled}] = \frac{W(y)}{W(F)}$

- $F := (X_1 \lor X_2)$;
  - $W[(0, 0)] = W[(1, 1)] = \frac{1}{6}; W[(1, 0)] = W[(0, 1)] = \frac{1}{3}$

- $\text{Sol}(F) = \{(0, 1), (1, 0), (1, 1)\}$

- $W(F) = \frac{1}{3} + \frac{1}{3} + \frac{1}{6} = \frac{5}{6}$
Network Reliability

Probabilistic Inference
Today’s Menu

Network Reliability

Probabilistic Inference

Constrained Counting
Today’s Menu

Network Reliability

Probabilistic Inference  Constrained Counting  Hashing Framework
Network Reliability

Probabilistic Inference

Constrained Counting

Hardware Validation

Constrained Sampling

Hashing Framework
Can we reliably predict the effect of natural disasters on critical infrastructure such as power grids?
Can we reliably predict the effect of natural disasters on critical infrastructure such as power grids?
Can we predict likelihood of a region facing blackout?
Reliability of Critical Infrastructure Networks

- $G = (V, E)$; source node: $s$ and terminal node $t$
- failure probability $g : E \rightarrow [0, 1]$
- Compute $\Pr[ s \text{ and } t \text{ are disconnected}]$?

Plantersville, SC
\( G = (V, E); \) source node: \( s \) and terminal node \( t \)

- failure probability \( g : E \rightarrow [0, 1] \)
- Compute \( \Pr[ s \text{ and } t \text{ are disconnected}] \)?
- \( \pi \): Configuration (of network) denoted by a 0/1 vector of size \( |E| \)
- \( W(\pi) = \Pr(\pi) \)

Plantersville, SC
Reliability of Critical Infrastructure Networks

- \( G = (V, E) \); source node: \( s \) and terminal node \( t \)
- failure probability \( g : E \rightarrow [0, 1] \)
- Compute \( \Pr[s \text{ and } t \text{ are disconnected}] \)?
- \( \pi : \) Configuration (of network) denoted by a 0/1 vector of size \(|E|\)
- \( W(\pi) = \Pr(\pi) \)
- \( \pi_{s,t} : \) configuration where \( s \) and \( t \) are disconnected
  - Represented as a solution to set of constraints over edge variables
Reliability of Critical Infrastructure Networks

- $G = (V, E)$; source node: $s$ and terminal node $t$
- Failure probability $g : E \rightarrow [0, 1]$
- Compute $\Pr[\text{s and t are disconnected}]$?
- $\pi$: Configuration (of network) denoted by a $0/1$ vector of size $|E|$ of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where $s$ and $t$ are disconnected
  - Represented as a solution to set of constraints over edge variables
- $\Pr[\text{s and t are disconnected}] = \sum_{\pi_{s,t}} W(\pi_{s,t})$
• $G = (V, E)$; source node: $s$ and terminal node $t$
• failure probability $g : E \rightarrow [0, 1]$
• Compute $\Pr[\text{s and t are disconnected}]$?
• $\pi : \text{Configuration (of network) denoted by a 0/1 vector of size } |E|$
• $W(\pi) = \Pr(\pi)$
• $\pi_{s,t}$: configuration where $s$ and $t$ are disconnected
  – Represented as a solution to set of constraints over edge variables
• $\Pr[\text{s and t are disconnected}] = \sum_{\pi_{s,t}} W(\pi_{s,t})$

(Constrained Counting, DMPV, AAAI 17)
### Probabilistic Models

<table>
<thead>
<tr>
<th>Patient</th>
<th>Cough</th>
<th>Smoker</th>
<th>Asthma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Randee</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tova</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Azucena</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Georgine</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Shoshana</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Lina</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hermine</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Graph showing relationships between Smoker (S), Asthma (A), and Cough (C) with nodes and edges]
## Probabilistic Models

<table>
<thead>
<tr>
<th>Patient</th>
<th>Cough</th>
<th>Smoker</th>
<th>Asthma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Randee</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tova</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Azucena</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Georgine</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Shoshana</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Lina</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hermine</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Bayesian Network Diagram](image-url)

- **Smoker (S)**
- **Asthma (A)**
- **Cough (C)**
### Probabilistic Models

<table>
<thead>
<tr>
<th>Patient</th>
<th>Cough</th>
<th>Smoker</th>
<th>Asthma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Randee</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tova</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Azucena</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Georgine</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Shoshana</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Lina</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hermine</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\Pr[\text{Asthma}(A) \mid \text{Cough}(C)] = \frac{\Pr[A \cap C]}{\Pr[C]}
\]
Probabilistic Models

<table>
<thead>
<tr>
<th>Patient</th>
<th>Cough</th>
<th>Smoker</th>
<th>Asthma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Randee</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tova</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Azucena</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Georgine</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Shoshana</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Lina</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hermine</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\Pr[\text{Asthma}(A) \mid \text{Cough}(C)] = \frac{\Pr[A \cap C]}{\Pr[C]}
\]

\[
F = A \land C
\]
Pr[Asthma(A) | Cough(C)] = \frac{\Pr[A \cap C]}{\Pr[C]}

F = A \land C

Sol(F) = \{(A, C, S), (A, C, \bar{S})\}
Probabilistic Models

<table>
<thead>
<tr>
<th>Patient</th>
<th>Cough</th>
<th>Smoker</th>
<th>Asthma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Randee</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tova</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Azucena</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Georgine</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Shoshana</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Lina</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hermine</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \Pr[\text{Asthma}(A) \mid \text{Cough}(C)] = \frac{\Pr[A \cap C]}{\Pr[C]} \]

\( F = A \land C \)

\( \text{Sol}(F) = \{(A, C, S), (A, C, \bar{S})\} \)

\[ \Pr[A \cap C] = \sum_{y \in \text{Sol}(F)} W(y) = W(F) \]
Prior Work

Strong guarantees but poor scalability

- Hashing-based approach (Stockmeyer 1983, Jerrum Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Sampling-based techniques (Wei and Selman 2005, Rubinstein 2012, Gogate and Dechter 2011)
Prior Work

Strong guarantees but poor scalability

- Hashing-based approach (Stockmeyer 1983, Jerrum Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Sampling-based techniques (Wei and Selman 2005, Rubinstein 2012, Gogate and Dechter 2011)

How to bridge this gap?
Constrained Counting

- **ExactCount**($F$, $W$): Compute $W(F)$?
  - \#P-complete (Valiant 1979)
Constrained Counting

- **ExactCount**($F$, $W$): Compute $W(F)$?
  - #P-complete

- **ApproxCount**($F$, $W$, $\varepsilon$, $\delta$): Compute $C$ such that
  \[
  \Pr\left[\frac{W(F)}{1 + \varepsilon} \leq C \leq W(F)(1 + \varepsilon)\right] \geq 1 - \delta
  \]

(Valiant 1979)
Boolean Formula $F$ and weight function $W : \{0, 1\}^n \rightarrow \mathbb{Q}_{\geq 0}$

\[ W(F) = c(W) \times |\text{Sol}(F')| \]

- Key Idea: Encode weight function as a set of constraints

(CFMV, IJCAI15)
Boolean Formula $F$ and weight function $W : \{0, 1\}^n \rightarrow \mathbb{Q}^\geq 0$

$$W(F) = c(W) \times |\text{Sol}(F')|$$

- Key Idea: Encode weight function as a set of constraints

How do we estimate $|\text{Sol}(F')|$?
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique ($n = 23$) 23 bit identifier ($2^n = 5.6M$)
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique ($n = 23$) bit identifier ($2^n = 5.6M$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by $5.6M/50$
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique \( n = 23 \) bit identifier \( 2^n = 5.6M \)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 5.6M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique \( n = 23 \) bit identifier \( 2^n = 5.6M \)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 5.6M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP Query: Find a person who likes coffee
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique \((n =)\) 23 bit identifier \((2^n = 5.6M)\)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 5.6M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person \(y\)
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique \( (n =) \) 23 bit identifier \( (2^n = 5.6M) \)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 5.6M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person \( y \)
- Attempt #2: Enumerate every person who likes coffee
How many people in Singapore like coffee?

- Population of NUS = 5.6M
- Assign every person a unique \((n =)\) 23 bit identifier \(2^n = 5.6M\)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 5.6M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person \(y\)
- Attempt #2: Enumerate every person who likes coffee
  - Potentially \(2^n\) queries

Can we do with lesser \# of SAT queries – \(O(n)\) or \(O(\log n)\)?
As Simple as Counting Dots
As Simple as Counting Dots
As Simple as Counting Dots

Pick a random cell

Estimate = Number of solutions in a cell \times Number of cells
Challenge 1  How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?
Challenges

**Challenge 1**  How to partition into *roughly equal small* cells of solutions without knowing the distribution of solutions?

**Challenge 2**  How many cells?
Challenge 1  How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Designing function $h : \text{assignments} \rightarrow \text{cells}$ (hashing)
- Solutions in a cell $\alpha$: $\text{Sol}(F) \cap \{y \mid h(y) = \alpha\}$
Challenge 1  How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Designing function $h : \text{assignments} \rightarrow \text{cells}$ (hashing)
- Solutions in a cell $\alpha$: $\text{Sol}(F) \cap \{y \mid h(y) = \alpha\}$
- Deterministic $h$ unlikely to work
Challenge 1  How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Designing function $h : \text{assignments} \rightarrow \text{cells}$ (hashing)
- Solutions in a cell $\alpha$: $\text{Sol}(F) \cap \{y \mid h(y) = \alpha\}$
- Deterministic $h$ unlikely to work
- Choose $h$ randomly from a large family $H$ of hash functions

Universal Hashing (Carter and Wegman 1977)
2-Universal Hashing

- Let $H$ be family of 2-universal hash functions mapping $\{0, 1\}^n$ to $\{0, 1\}^m$

\[ \forall y_1, y_2 \in \{0, 1\}^n, \alpha_1, \alpha_2 \in \{0, 1\}^m, h \leftarrow^R H \]

\[ \Pr[h(y_1) = \alpha_1] = \Pr[h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right) \]

\[ \Pr[h(y_1) = \alpha_1 \land h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)^2 \]
2-Universal Hashing

- Let $H$ be family of 2-universal hash functions mapping $\{0, 1\}^n$ to $\{0, 1\}^m$

$$\forall y_1, y_2 \in \{0, 1\}^n, \alpha_1, \alpha_2 \in \{0, 1\}^m, h \leftarrow H \Pr[h(y_1) = \alpha_1] = \Pr[h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)$$

$$\Pr[h(y_1) = \alpha_1 \land h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)^2$$

- The power of 2-universality
  - $Z$ be the number of solutions in a randomly chosen cell
  - $E[Z] = \frac{|\text{Sol}(F)|}{2^m}$
  - $\sigma^2[Z] \leq E[Z]$
2-Universal Hashing

- Let $H$ be family of 2-universal hash functions mapping $\{0, 1\}^n$ to $\{0, 1\}^m$

\[
\forall y_1, y_2 \in \{0, 1\}^n, \alpha_1, \alpha_2 \in \{0, 1\}^m, h \leftarrow H \quad Pr[h(y_1) = \alpha_1] = Pr[h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)
\]

\[
Pr[h(y_1) = \alpha_1 \land h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)^2
\]

- The power of 2-universality
  - $Z$ be the number of solutions in a randomly chosen cell
  - $E[Z] = \frac{|\text{Sol}(F)|}{2^m}$
  - $\sigma^2[Z] \leq E[Z]$
2-Universal Hash Functions

- Variables: \( X_1, X_2, \ldots, X_n \)
- To construct \( h : \{0, 1\}^n \rightarrow \{0, 1\}^m \), choose \( m \) random XORs
- Pick every \( X_i \) with prob. \( \frac{1}{2} \) and XOR them
  - \( X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} \)
  - Expected size of each XOR: \( \frac{n}{2} \)
2-Universal Hash Functions

- Variables: $X_1, X_2, \cdots X_n$
- To construct $h: \{0, 1\}^n \rightarrow \{0, 1\}^m$, choose $m$ random XORs
- Pick every $X_i$ with prob. $\frac{1}{2}$ and XOR them
  - $X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
  - Expected size of each XOR: $\frac{n}{2}$
- To choose $\alpha \in \{0, 1\}^m$, set every XOR equation to 0 or 1 randomly
  \[
  X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} = 0 \quad (Q_1)
  \]
  \[
  X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} = 1 \quad (Q_2)
  \]
  \[
  \cdots \\
  \]
  \[
  X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} = 1 \quad (Q_m)
  \]
- Solutions in a cell: $F \land Q_1 \cdots \land Q_m$
2-Universal Hash Functions

- Variables: $X_1, X_2, \cdots X_n$
- To construct $h : \{0, 1\}^n \rightarrow \{0, 1\}^m$, choose m random XORs
- Pick every $X_i$ with prob. $\frac{1}{2}$ and XOR them
  - $X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
  - Expected size of each XOR: $\frac{n}{2}$
- To choose $\alpha \in \{0, 1\}^m$, set every XOR equation to 0 or 1 randomly
  \[
  \begin{align*}
  X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} &= 0 \\
  X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} &= 1 \\
  \cdots \\
  X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} &= 1
  \end{align*}
  \]
- Solutions in a cell: $F \land Q_1 \cdots \land Q_m$
- Performance of state of the art SAT solvers degrade with increase in the size of XORs (SAT Solvers $\neq$ SAT oracles)
2-Universal Hash Functions

- Variables: $X_1, X_2, \cdots, X_n$
- To construct $h : \{0, 1\}^n \rightarrow \{0, 1\}^m$, choose $m$ random XORs
- Pick every $X_i$ with prob. $\frac{1}{2}$ and XOR them
  - $X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
  - Expected size of each XOR: $\frac{n}{2}$
- To choose $\alpha \in \{0, 1\}^m$, set every XOR equation to 0 or 1 randomly
  \[
  \begin{align*}
  X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} &= 0 \quad (Q_1) \\
  X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} &= 1 \quad (Q_2) \\
  \vdots & \quad (\cdots) \\
  X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} &= 1 \quad (Q_m)
  \end{align*}
  \]

- Solutions in a cell: $F \land Q_1 \cdots \land Q_m$
- Performance of state of the art SAT solvers degrade with increase in the size of XORs (SAT Solvers $\neq$ SAT oracles)
- Two orders of magnitude reduction in the size of XORs by embedding formula into smaller dimension (“Independent Support")
  (IMMV CP15, Best Student Paper)  (IMMV Constraints16, Invited Paper)
Challenges

Challenge 1  How to partition into \textit{roughly equal small} cells of solutions without knowing the distribution of solutions?
  
  \begin{itemize}
    \item Independent Support-based 2-Universal Hash Functions
  \end{itemize}

Challenge 2  How many cells?
Question 2: How many cells?

- A cell is small if it has less than $\text{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions
Question 2: How many cells?

- A cell is small if it has less than $\text{thresh} = 5\left(1 + \frac{1}{\epsilon}\right)^2$ solutions.
- We want to partition into $2^{m^*}$ cells such that $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$. 
Question 2: How many cells?

- A cell is small if it has less than \( \text{thresh} = 5(1 + \frac{1}{\varepsilon})^2 \) solutions.
- We want to partition into \( 2^{m^*} \) cells such that \( 2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}} \)
  - Check for every \( m = 0, 1, \cdots, n \) if the number of solutions \( \leq \text{thresh} \).
ApproxMC\((F, \varepsilon, \delta)\)

\[
\text{# of sols} \leq \text{thresh}?
\]
ApproxMC($F, \varepsilon, \delta$)

# of sols $\leq$ thresh? No

# of sols $\leq$ thresh?
ApproxMC($F, \varepsilon, \delta$)

- No
- No
- No
ApproxMC($F, \varepsilon, \delta$)

No

# of sols $\leq$ thresh?

No

# of sols $\leq$ thresh?

No

# of sols $\leq$ thresh?

No

# of sols $\leq$ thresh?
ApproxMC($F, \varepsilon, \delta$)

Estimate = \# of sols $\times$ \# of cells

# of sols $\leq$ thresh?
ApproxMC($F, \varepsilon, \delta$)

- We want to partition into $2^{m^*}$ cells such that $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$
  
  - Query 1: Is $\#(F \land Q_1) \leq \text{thresh}$
  - Query 2: Is $\#(F \land Q_1 \land Q_2) \leq \text{thresh}$
  - ...
  - Query $n$: Is $\#(F \land Q_1 \land Q_2 \cdots \land Q_n) \leq \text{thresh}$

- Stop at the first $m$ where Query $m$ returns YES and return estimate as $\#(F \land Q_1 \land Q_2 \cdots \land Q_m) \times 2^m$

- Observation: $\#(F \land Q_1 \cdots \land Q_i \land Q_{i+1}) \leq \#(F \land Q_1 \cdots \land Q_i)$
  
  - If Query $i$ returns YES, then Query $i + 1$ must return YES
ApproxMC\((F, \varepsilon, \delta)\)

- We want to partition into \(2^{m^*}\) cells such that \(2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}\)
  - Query 1: Is \(#(F \land Q_1) \leq \text{thresh}\)
  - Query 2: Is \(#(F \land Q_1 \land Q_2) \leq \text{thresh}\)
  - \ldots
  - Query \(n\): Is \(#(F \land Q_1 \land Q_2 \cdots \land Q_n) \leq \text{thresh}\)

- Stop at the first \(m\) where Query \(m\) returns YES and return estimate as \(#(F \land Q_1 \land Q_2 \cdots \land Q_m) \times 2^m\)

- Observation: \(#(F \land Q_1 \cdots \land Q_i \land Q_{i+1}) \leq #(F \land Q_1 \cdots \land Q_i)\)
  - If Query \(i\) returns YES, then Query \(i + 1\) must return YES
  - Logarithmic search (\# of SAT calls: \(O(\log n)\))
ApproxMC\((F, \varepsilon, \delta)\)

- We want to partition into \(2^{m^*}\) cells such that \(2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}\)
  - Query 1: Is \(#(F \land Q_1) \leq \text{thresh}\)
  - Query 2: Is \(#(F \land Q_1 \land Q_2) \leq \text{thresh}\)
  - \ldots
  - Query \(n\): Is \(#(F \land Q_1 \land Q_2 \cdots \land Q_n) \leq \text{thresh}\)
- Stop at the first \(m\) where Query \(m\) returns YES and return estimate as \(#(F \land Q_1 \land Q_2 \cdots \land Q_m) \times 2^m\)
- **Observation:** \(#(F \land Q_1 \cdots \land Q_i \land Q_{i+1}) \leq #(F \land Q_1 \cdots \land Q_i)\)
  - If Query \(i\) returns YES, then Query \(i + 1\) must return YES
  - Logarithmic search (\(#\) of SAT calls: \(O(\log n))\)
- **Will this work? Will the “\(m\)” where we stop be close to \(m^*\)?
ApproxMC\((F, \varepsilon, \delta)\)

- We want to partition into \(2^{m^*}\) cells such that \(2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}\)
  - Query 1: Is \(#(F \land Q_1) \leq \text{thresh}\)
  - Query 2: Is \(#(F \land Q_1 \land Q_2) \leq \text{thresh}\)
  - \ldots
  - Query \(n\): Is \(#(F \land Q_1 \land Q_2 \cdots \land Q_n) \leq \text{thresh}\)

- Stop at the first \(m\) where Query \(m\) returns YES and return estimate as \(#(F \land Q_1 \land Q_2 \cdots \land Q_m) \times 2^m\)

- Observation: \(#(F \land Q_1 \cdots \land Q_i \land Q_{i+1}) \leq #(F \land Q_1 \cdots \land Q_i)\)
  - If Query \(i\) returns YES, then Query \(i + 1\) must return YES
  - Logarithmic search (\# of SAT calls: \(O(\log n)\))

- Will this work? Will the “\(m\)” where we stop be close to \(m^*\)?
  - Challenge Query \(i\) and Query \(j\) are not independent
  - Independence crucial to analysis (Stockmeyer 1983, \ldots)
ApproxMC($F, \varepsilon, \delta$)

- We want to partition into $2^{m^*}$ cells such that $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$
  - Query 1: Is $\#(F \land Q_1) \leq \text{thresh}$
  - Query 2: Is $\#(F \land Q_1 \land Q_2) \leq \text{thresh}$
  - ...
  - Query $n$: Is $\#(F \land Q_1 \land Q_2 \cdots \land Q_n) \leq \text{thresh}$

- Stop at the first $m$ where Query $m$ returns YES and return estimate as $\#(F \land Q_1 \land Q_2 \cdots \land Q_m) \times 2^m$

- Observation: $\#(F \land Q_1 \cdots \land Q_i \land Q_{i+1}) \leq \#(F \land Q_1 \cdots \land Q_i)$
  - If Query $i$ returns YES, then Query $i + 1$ must return YES
  - Logarithmic search (# of SAT calls: $O(\log n)$)

- Will this work? Will the “$m$” where we stop be close to $m^*$?
  - Challenge Query $i$ and Query $j$ are not independent
  - Independence crucial to analysis (Stockmeyer 1983, ···)
  - Key Insight: The probability of making a bad choice of $Q_i$ is very small for $i \ll m^*$

(CMV, IJCAI16)
Let \( 2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}} \) \((m^* = \log(\frac{|\text{Sol}(F)|}{\text{thresh}}))\)

**Lemma (1)**

\(\text{ApproxMC}(F, \varepsilon, \delta)\) terminates with \(m \in \{m^* - 1, m^*\}\) with probability \(\geq 0.8\)

**Lemma (2)**

For \(m \in \{m^* - 1, m^*\}\), estimate obtained from a randomly picked cell lies within a tolerance of \(\varepsilon\) of \(|\text{Sol}(F)|\) with probability \(\geq 0.8\)
**Theorem (Correctness)**

\[
\Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq \text{ApproxMC}(F, \varepsilon, \delta) \leq |\text{Sol}(F)|\left(1 + \varepsilon\right) \right] \geq 1 - \delta
\]

**Theorem (Complexity)**

\(\text{ApproxMC}(F, \varepsilon, \delta)\) makes \(O\left(\frac{\log n \log(\frac{1}{\delta})}{\varepsilon^2}\right)\) calls to SAT oracle.

- Prior work required \(O\left(\frac{n \log n \log(\frac{1}{\delta})}{\varepsilon}\right)\) calls to SAT oracle  \((\text{Stockmeyer 1983})\)
Reliability of Critical Infrastructure Networks

Plaintersville, SC

- $G = (V, E)$; source node: $s$
- Compute $\text{Pr}[t \text{ is disconnected}]$?

Timeout = 1000 seconds

(DMPV, AAAI17)
Reliability of Critical Infrastructure Networks

Plantersville, SC

- $G = (V, E)$; source node: $s$
- Compute $\Pr[\text{t is disconnected}]$?

Timeout = 1000 seconds

(DMPV, AAAI17)
G = (V, E); source node: s

Compute Pr[t is disconnected]?

Timeout = 1000 seconds

(DMPV, AAAI17)
Beyond Network Reliability

ApproxMC

- Probabilistic Inference
- Network Reliability
- Quantified Information Flow
- Program Synthesis

(CFMSV, AAAI14), IMMV, CP15), (CFMV, IJCAI15), (CMMV, AAAI16), (CMV, IJCAI16)

(Fremont, Rabe and Seshia 2017)

(CFMSV, AAAI14), Fremont et al 2017, Ellis et al 2017
Network Reliability

Probabilistic Inference  Constrained Counting
Network Reliability

Probabilistic Inference

Constrained Counting

Hashing Framework
Network Reliability

Probabilistic Inference

Constrained Counting

Hardware Validation

Hashing Framework
Hardware Validation

- Design is simulated with test vectors (values of $a$ and $b$)
- Results from simulation compared to intended results
Hardware Validation

- Design is simulated with test vectors (values of $a$ and $b$)
- Results from simulation compared to intended results
- Challenge: How do we generate test vectors?
  - $2^{128}$ combinations for a toy circuit
• Design is simulated with test vectors (values of $a$ and $b$)
• Results from simulation compared to intended results
• Challenge: How do we generate test vectors?
  - $2^{128}$ combinations for a toy circuit
• Use constraints to represent *interesting* verification scenarios
Constrained-Random Simulation

Constraints

- Designers:
  - $a +_{64} 11 \times 32b = 12$
  - $a <_{64} (b >> 4)$

- Past Experience:
  - $40 <_{64} 34 + a <_{64} 5050$
  - $120 <_{64} b <_{64} 230$

- Users:
  - $232 \times 32a +_{64} b! = 1100$
  - $1020 <_{64} (b/_{64}2) +_{64} a <_{64} 2200$

Test vectors: random solutions of constraints
Constrained Sampling

- Given:
  - Set of Constraints $F$ over variables $X_1, X_2, \cdots X_n$
- Uniform Sampler
  \[
  \forall y \in \text{Sol}(F), \Pr[y \text{ is output}] = \frac{1}{|\text{Sol}(F)|}
  \]
- Almost-Uniform Sampler
  \[
  \forall y \in \text{Sol}(F), \frac{1}{(1 + \varepsilon)|\text{Sol}(F)|} \leq \Pr[y \text{ is output}] \leq \frac{(1 + \varepsilon)}{|\text{Sol}(F)|}
  \]
Approximate counting and almost-uniform sampling are inter-reducible (Jerrum, Valiant and Vazirani, 1986)

Is the reduction efficient?
Approximate counting and almost-uniform sampling are inter-reducible \footnote{Jerrum, Valiant and Vazirani, 1986}

Is the reduction efficient?
- Almost-uniform sampler (JVV) require linear number of approximate counting calls
Prior Work

Strong guarantees but poor scalability

- Polynomial calls to NP oracle (Bellare, Goldreich and Petrank, 2000)
- BDD-based techniques (Yuan et al. 1999, Yuan et al. 2004, Kukula and Shiple 2000)
- Reduction to approximate counting (Jerrum, Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Randomization in SAT solvers (Moskiewicz 2001, Nadel 2011)
- Belief Networks (Dechter 2002, Gogate and Dechter 2006)
Prior Work

Strong guarantees but poor scalability

- Polynomial calls to NP oracle  
  (Bellare, Goldreich and Petrank, 2000)
- BDD-based techniques  
- Reduction to approximate counting  
  (Jerrum, Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Randomization in SAT solvers  
  (Moskewicz 2001, Nadel 2011)
- MCMC-based approaches  
- Belief Networks  
  (Dechter 2002, Gogate and Dechter 2006)

How to bridge this gap?
Key Ideas

- For right choice of number of cells, 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 Ideas

- For right choice of number of cells, 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

**Challenge:** How many cells?
How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}} \quad (m^* = \log \frac{|\text{Sol}(F)|}{\text{thresh}})$
  - $\text{ApproxMC}(F, \varepsilon, \delta)$ returns $C$ such that
    $$\Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq C \leq |\text{Sol}(F)|(1 + \varepsilon) \right] \geq 1 - \delta$$
  - $\tilde{m} = \log \frac{C}{\text{thresh}}$
Desired Number of cells: $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$ ( $m^* = \log \frac{|\text{Sol}(F)|}{\text{thresh}}$ )

- ApproxMC$(F, \varepsilon, \delta)$ returns $C$ such that
  \[
  \Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq C \leq |\text{Sol}(F)|(1 + \varepsilon) \right] \geq 1 - \delta
  \]

- $\tilde{m} = \log \frac{C}{\text{thresh}}$
- Check for $m = \tilde{m} - 1, \tilde{m}, \tilde{m} + 1$ if a randomly chosen cell is small
How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$ ($m^* = \log \frac{|\text{Sol}(F)|}{\text{thresh}}$)
  - $\text{ApproxMC}(F, \varepsilon, \delta)$ returns $C$ such that
    \[
    \Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq C \leq |\text{Sol}(F)|(1+\varepsilon) \right] \geq 1 - \delta
    \]
  - $\tilde{m} = \log \frac{C}{\text{thresh}}$
  - Check for $m = \tilde{m} - 1, \tilde{m}, \tilde{m} + 1$ if a randomly chosen cell is small
  - Not just a practical hack required non-trivial proof

(CMV, CAV13)
(CMV, DAC14)
(CFMSV, TACAS15)
Theoretical Guarantees

Theorem (Almost-Uniformity)

\[ \forall y \in \text{Sol}(F), \quad \frac{1}{(1+\varepsilon)|\text{Sol}(F)|} \leq \Pr[y \text{ is output}] \leq \frac{1+\varepsilon}{|\text{Sol}(F)|} \]
Theoretical Guarantees

Theorem (Almost-Uniformity)
\[
\forall y \in \text{Sol}(F), \quad \frac{1}{(1+\varepsilon)|\text{Sol}(F)|} \leq \Pr[y \text{ is output}] \leq \frac{1+\varepsilon}{|\text{Sol}(F)|}
\]

Theorem (Query)

For a formula $F$ over $n$ variables UniGen makes one call to approximate counter
Theoretical Guarantees

**Theorem (Almost-Uniformity)**

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

**Theorem (Query)**

For a formula \( F \) over \( n \) variables \( \text{UniGen} \) makes **one call** to approximate counter

- JVV (Jerrum, Valiant and Vazirani 1986) makes \( n \) calls
### Three Orders of Improvement

<table>
<thead>
<tr>
<th>SAT Solver</th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired Uniform Generator</td>
<td>10</td>
</tr>
</tbody>
</table>

Experiments over 200+ benchmarks

(CMV, CAV13)
(CMV, DAC14)
(CFMSV, TACAS15)
### Three Orders of Improvement

<table>
<thead>
<tr>
<th></th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
<tr>
<td>Desired Uniform Generator</td>
<td>10</td>
</tr>
<tr>
<td>XORSample (2012 state of the art)</td>
<td>50000</td>
</tr>
</tbody>
</table>

Experiments over 200+ benchmarks

(CMv, CAV13)

(CMv, DAC14)

(CFMSv, TACAS15)
Three Orders of Improvement

<table>
<thead>
<tr>
<th></th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
<tr>
<td>Desired Uniform Generator</td>
<td>10</td>
</tr>
<tr>
<td>XORSample (2012 state of the art)</td>
<td>50000</td>
</tr>
<tr>
<td>UniGen (2015)</td>
<td>21</td>
</tr>
</tbody>
</table>

(CMV, CAV13)
(CMV, DAC14)
(CFMSV, TACAS15)

Experiments over 200+ benchmarks
## Three Orders of Improvement

<table>
<thead>
<tr>
<th></th>
<th>Relative Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT Solver</td>
<td>1</td>
</tr>
<tr>
<td>Desired Uniform Generator</td>
<td>10</td>
</tr>
<tr>
<td>XORSample (2012 state of the art)</td>
<td>50000</td>
</tr>
<tr>
<td>UniGen (2015)</td>
<td>21</td>
</tr>
</tbody>
</table>

Experiments over 200+ benchmarks

*Closer to technical transfer*
• Benchmark: case110.cnf; #var: 287; #clauses: 1263
• Total Runs: $4 \times 10^6$; Total Solutions : 16384
Benchmark: case110.cnf; \#var: 287; \#clauses: 1263
Total Runs: $4 \times 10^6$; Total Solutions: 16384
Usages of Open Source Tool: UniGen

- Music Improvisation
- Pattern Mining
- Quantified Information Flow
- Problem Generation
- Hardware Validation
Speedup over 2012 state of the art

- 2012
- 2013: CAV 13
- 2014: DAC 14, AAAI 14
- 2015: CP 15, IJCAI 15, TACAS 15
- 2016: IJCAI 16a, IJCAI 16b, AAAI 16, Constraints

The speedup is shown on a logarithmic scale.
Requires combinations of ideas from theory, statistics and systems
• Tighter integration between solvers and algorithms
• Tighter integration between solvers and algorithms
• Exploring solution space structure of CNF+XOR formulas (DMV, IJCAI16),
Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms
- Exploring solution space structure of CNF+XOR formulas
  
  \[(DMV, IJCAI16),\]

- Beyond Boolean variables – without *bit blasting*
Challenge Problems
Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid
Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid
Privacy  Leakage Measurement for C++ program with 100 lines
Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid
Privacy  Leakage Measurement for C++ program with 100 lines
Artificial Intelligence  Inference for Bayesian network with 1K nodes
Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid
Privacy  Leakage Measurement for C++ program with 100 lines
Artificial Intelligence  Inference for Bayesian network with 1K nodes
The Potential of Hashing-based Framework
Machine Learning  Probabilistic programming
Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid

Privacy  Leakage Measurement for C++ program with 100 lines

Artificial Intelligence  Inference for Bayesian network with 1K nodes

The Potential of Hashing-based Framework

Machine Learning  Probabilistic programming

  Theory  Classification of Approximate counting complexity
Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid
  Privacy  Leakage Measurement for C++ program with 100 lines

Artificial Intelligence  Inference for Bayesian network with 1K nodes

The Potential of Hashing-based Framework

Machine Learning  Probabilistic programming
  Theory  Classification of Approximate counting complexity

Databases  Streaming algorithms
Part I

Backup
Highly Accurate Estimates
Highly Accurate Estimates

![Diagram showing error values ranging from 0 to 1 for different terminals.]
Highly Accurate Estimates

Terminal Error

Allowed

ApproxMC