Approximate Counting and Sampling

Kuldeep S. Meel

National University of Singapore

Simons Bootcamp
The Amazing Collaborators

S. Akshay (IITB, India), Teodora Baluta (NUS, SG), Fabrizio Biondi (Avast, CZ), Supratik Chakraborty (IITB, India), Alexis de Colnet (NUS, SG), Remi Delannoy (NUS, SG), Jeffrey Dudek (Rice, US), Leonardo Duenas-Osorio (Rice, US), Mike Enescu (Inria, France) Daniel Fremont (UCB, US), Dror Fried (Open U., Israel), Stephan Gocht (Lund U., Sweden), Rahul Gupta (IITK, India), Annelie Heuser (Inria, France), Alexander Ivrii (IBM, Israel), Alexey Ignatiev (IST, Portugal), Axel Legay (UCL, Belgium), Sharad Malik (Princeton, US), Joao Marques Silva (IST, Portugal), Rakesh Mistry (IITB, India), Nina Narodytska ((VMWare, US), Roger Paredes (Rice, US), Yash Pote (NUS, SG), Jean Quilbeuf (Inria, France), Subhajit Roy (IITK, India), Mate Soos (NUS, SG), Prateek Saxena (NUS, SG), Sanjit Seshia (UCB, US), Shubham Sharma (IITK, India), Aditya Shrotri (Rice, US), Moshe Vardi (Rice, US)

Special shout out to Mate Soos, maintainer of ApproxMC and UniGen
Modern SAT solvers are able to deal routinely with practical problems that involve millions 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 millions of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)

Industrial usage of SAT Solvers: Model Checking, 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 millions of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)

Industrial usage of SAT Solvers: Model Checking, Planning, Genome Rearrangement, Telecom Feature Subscription, Resource Constrained Scheduling, Noise Analysis, Games, · · ·

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

- Given
  - Boolean variables $X_1, X_2, \ldots X_n$
  - Formula $F$ over $X_1, X_2, \ldots X_n$
- $\text{Sol}(F) = \{ \text{solutions of } F \}$
Counting and Sampling

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

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

- **Counting**: Determine $|\text{Sol}(F)|$
  - Approximation: $\Pr \left[ \frac{|\text{Sol}(F)|}{1+\epsilon} \leq c \leq |\text{Sol}(F)|(1+\epsilon) \right] \geq 1 - \delta$
Counting and Sampling

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

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

- **Counting**: Determine $|\text{Sol}(F)|$
  - Approximation: $\Pr\left[\frac{|\text{Sol}(F)|}{1+\varepsilon} \leq c \leq |\text{Sol}(F)|(1+\varepsilon)\right] \geq 1 - \delta$

- **Uniform Sampling** $\Pr[y \text{ is output}] = \frac{1}{|\text{Sol}(F)|}$
  - Almost-Uniform: $\frac{1}{(1+\varepsilon)|\text{Sol}(F)|} \leq \Pr[y \text{ is output}] \leq \frac{1+\varepsilon}{|\text{Sol}(F)|}$
Counting and Sampling

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula $F$ over $X_1, X_2, \cdots X_n$
- $\text{Sol}(F) = \{ \text{solutions of } F \}$
- Counting: Determine $|\text{Sol}(F)|$
  - Approximation: $\Pr\left[\frac{|\text{Sol}(F)|}{1+\varepsilon} \leq c \leq |\text{Sol}(F)|(1 + \varepsilon)\right] \geq 1 - \delta$
- Uniform Sampling $\Pr[y \text{ is output}] = \frac{1}{|\text{Sol}(F)|}$
  - Almost-Uniform: $\frac{1}{(1+\varepsilon)|\text{Sol}(F)|} \leq \Pr[y \text{ is output}] \leq \frac{1+\varepsilon}{|\text{Sol}(F)|}$
- Given
  - $F := (X_1 \lor X_2)$
- $\text{Sol}(F) = \{(0, 1), (1, 0), (1, 1)\}$
- $|\text{Sol}(F)| = 3$
Applications across Computer Science

Counting & Sampling

Network Reliability

Hardware Validation

Explainable AI

Quantified Information Flow

Neural Network Robustness
Obs 1  SAT Oracle \( \neq \) NP Oracle

- Returns UNSAT with a proof
- Return a satisfying assignment if satisfiable
Obs 1  SAT Oracle $\neq$ NP Oracle
  - Returns UNSAT with a proof
  - Return a satisfying assignment if satisfiable

Obs 2  SAT Solver $\neq$ SAT oracle
  - The performance of solver depends on the formulas
Obs 1 SAT Oracle $\neq$ NP Oracle
- Returns UNSAT with a proof
- Return a satisfying assignment if satisfiable

Obs 2 SAT Solver $\neq$ SAT oracle
- The performance of solver depends on the formulas

Obs 3 Memoryfulness
- Incremental Solving: Often easier to solve $F$ followed by $G$ if we $G$ can be written as $G = F \land H$
- If $F \rightarrow C$ then $(F \land H) \implies C$
Today’s Menu

Constrained Counting
The Rise of Hashing-based Approach: Promise of Scalability and Guarantees
(S83,GSS06,GHSS07,CMV13b,EGSS13b,CMV14,CDR15,CMV16,ZCSE16,AD16 KM18,ATD18,SM19,ABM20,SGM20)
Constrained Counting  Hashing Framework
Constrained Sampling

The Rise of Hashing-based Approach: Promise of Scalability and Guarantees
(S83, GSS06, GHSS07, CMV13b, EGSS13b, CMV14, CDR15, CMV16, ZCSE16, AD16, KM18, ATD18, SM19, ABM20, SGM20)
Counting in Berkeley

How many people in Berkeley like coffee?

- Population of Berkeley = 112K
- Assign every person a unique \( n = 17 \) bit identifier \( 2^n = 112K \)
How many people in Berkeley like coffee?

- Population of Berkeley = 112K
- Assign every person a unique \( n \) (17 bit identifier \( 2^n = 112K \))
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 112K/50
How many people in Berkeley like coffee?

- Population of Berkeley = 112K
- Assign every person a unique \( n \) = 17 bit identifier \( 2^n = 112K \)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 112K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
Counting in Berkeley

How many people in Berkeley like coffee?

• Population of Berkeley = 112K
• Assign every person a unique \( n = 17 \) bit identifier \( 2^n = 112K \)
• Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 112K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
• SAT Query: Find a person who likes coffee

• SAT Query: 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 \( O(\log n) \)?
How many people in Berkeley like coffee?

- Population of Berkeley = 112K
- Assign every person a unique \((n =)\) 17 bit identifier \((2^n = 112K)\)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 112K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT 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 \(y\)
Counting in Berkeley

How many people in Berkeley like coffee?

- Population of Berkeley = 112K
- Assign every person a unique \( n \) = 17 bit identifier \( (2^n = 112K) \)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 112K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT 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 Berkeley like coffee?

- Population of Berkeley = 112K
- Assign every person a unique \( n \) 17 bit identifier \( 2^n = 112K \)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiply by 112K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT 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

Pick a random cell
Estimate = Number of solutions in a cell × Number of cells
As Simple as Counting Dots

Pick a random cell

Estimate = Number of solutions in a cell × Number of cells
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 3  What is exactly a *small cell*?
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)
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-wise independent Hashing

- Let $H$ be a family of 2-wise independent hash functions mapping \( \{0, 1\}^n \) to \( \{0, 1\}^m \)

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

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

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

- Let $H$ be family of 2-wise independent 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-wise independency
  - $Z$ 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-wise independent 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: \( \frac{n}{2} \)

- 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)

12/33
2-wise independent 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)
  
  \vdots \quad \cdots \quad (\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-wise independent 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)
Improved 2-wise Independent Hash Functions

• Not all variables are required to specify solution space of $F$
  - $F := X_3 \iff (X_1 \lor X_2)$
  - $X_1$ and $X_2$ uniquely determine rest of the variables (i.e., $X_3$)

• Formally: if $I$ is independent support, then $\forall \sigma_1, \sigma_2 \in \text{Sol}(F)$, if $\sigma_1$ and $\sigma_2$ agree on $I$ then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
• Not all variables are required to specify solution space of $F$
  - $F := X_3 \iff (X_1 \lor X_2)$
  - $X_1$ and $X_2$ uniquely determines rest of the variables (i.e., $X_3$)
• Formally: if $I$ is independent support, then $\forall \sigma_1, \sigma_2 \in \text{Sol}(F)$, if $\sigma_1$ and $\sigma_2$ agree on $I$ then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
• Random XORs need to be constructed only over $I$  (CMV DAC14)
• Not all variables are required to specify solution space of $F$
  - $F := X_3 \iff (X_1 \lor X_2)$
  - $X_1$ and $X_2$ uniquely determines rest of the variables (i.e., $X_3$)
• Formally: if $I$ is independent support, then $\forall \sigma_1, \sigma_2 \in \text{Sol}(F)$, if $\sigma_1$ and $\sigma_2$ agree on $I$ then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
• Random XORs need to be constructed only over $I$ (CMV DAC14)
• Typically $I$ is 1-2 orders of magnitude smaller than $X$
• Auxiliary variables introduced during encoding phase are dependent (Tseitin 1968)
Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of $F$
  - $F := X_3 \iff (X_1 \lor X_2)$
  - $X_1$ and $X_2$ uniquely determines rest of the variables (i.e., $X_3$)
- Formally: if $I$ is independent support, then $\forall \sigma_1, \sigma_2 \in \text{Sol}(F)$, if $\sigma_1$ and $\sigma_2$ agree on $I$ then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over $I$ (CMV DAC14)
- Typically $I$ is 1-2 orders of magnitude smaller than $X$
- Auxiliary variables introduced during encoding phase are dependent (Tseitin 1968)

Algorithmic procedure to determine $I$?
Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of $F$
  - $F := X_3 \iff (X_1 \lor X_2)$
  - $X_1$ and $X_2$ uniquely determines rest of the variables (i.e., $X_3$)
- Formally: if $I$ is independent support, then $\forall \sigma_1, \sigma_2 \in \text{Sol}(F)$, if $\sigma_1$ and $\sigma_2$ agree on $I$ then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over $I$ (CMV DAC14)
- Typically $I$ is 1-2 orders of magnitude smaller than $X$
- Auxiliary variables introduced during encoding phase are dependent (Tseitin 1968)

Algorithmic procedure to determine $I$?
- $FP^{NP}$ procedure via reduction to Minimal Unsatisfiable Subset
• Not all variables are required to specify solution space of $F$
  - $F := X_3 \iff (X_1 \lor X_2)$
  - $X_1$ and $X_2$ uniquely determines rest of the variables (i.e., $X_3$)
• Formally: if $I$ is independent support, then $\forall \sigma_1, \sigma_2 \in \text{Sol}(F)$, if $\sigma_1$ and $\sigma_2$ agree on $I$ then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
• Random XORs need to be constructed only over $I$ (CMV DAC14)
• Typically $I$ is 1-2 orders of magnitude smaller than $X$
• Auxiliary variables introduced during encoding phase are dependent (Tseitin 1968)

Algorithmic procedure to determine $I$?
• $FP^{NP}$ procedure via reduction to Minimal Unsatisfiable Subset
• Two orders of magnitude runtime improvement
  (IMMV; CP15, Constraints16)
• CNF + Sparse XORs are still CNF+XOR formulas.
• Translating XORs to CNF and performing CDCL is not sufficient
• CNF + Sparse XORs are still CNF+XOR formulas.
• Translating XORs to CNF and performing CDCL is not sufficient
  – XORs can be solved by Gaussian elimination
• CryptoMiniSAT: Solver designed to perform CDCL and Gaussian
  Elimination in tandem (SNC09; SM19, SGM20)
• BIRD (Blast, Inprocess, Recover, and Detach): Tighter integration
Challenges

Challenge 1  How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?
  • Independent Support-based XORs
  • Specialized CNF Solvers

Challenge 2  How many cells?

Challenge 3  What is exactly a small cell?
Challenge 2: How many cells?

• We want to partition into $2^{m^*}$ cells such that $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$
Challenge 2: How many cells?

- 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

\[ \text{# of sols} \leq \text{thresh} ? \]
ApproxMC

$\text{# of sols } \leq \text{thresh?}$

No

$\text{# of sols } \leq \text{thresh?}$
No

# of sols
≤ thresh?

No

# of sols
≤ thresh?
ApproxMC

No

No

No

No

No

No

No

No

No

No

No

No

No

No

No

No

No

No
ApproxMC

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

\# of sols \leq \text{thresh}?

No

\# of sols \leq \text{thresh}?

No

\# of sols \leq \text{thresh}?

\# of sols \leq \text{thresh}?

No

Yes

\# of sols \leq \text{thresh}?

\# of sols \leq \text{thresh}?

\# of sols \leq \text{thresh}?

\# of sols \leq \text{thresh}?
ApproxMC

• 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) $\leq$ thresh
  - Query 2: Is #(F $\land$ Q_1 $\land$ Q_2) $\leq$ thresh
  - ...
  - Query n: Is #(F $\land$ Q_1 $\land$ Q_2 $\cdots$ $\land$ Q_n) $\leq$ 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

- 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: $O(\log n)$)
  - Incremental Search
ApproxMC

• 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: $O(\log n)$)
  - Incremental Search

• Will this work? Will the “$m$” where we stop be close to $m^*$?
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: $O(\log n)$)
- Incremental Search

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, ...
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: $\mathcal{O}(\log n)$)
- Incremental Search

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 $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}} \quad (m^* = \log(\frac{|\text{Sol}(F)|}{\text{thresh}}))$

**Lemma (1)**

ApproxMC 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 $\geq 0.8$

Repeat $\mathcal{O}(\log(1/\delta))$ times and return the median
Challenge 3 What is a small cell?

A cell is small cell if it has $\approx$ solutions.

Approach 1: $\text{thresh} = \text{constant} \rightarrow 4$-factor approximation
- From 4 to 2-factor
Let $G = F_1 \wedge F_2$ (i.e., two identical copies of $F$)
$|\text{Sol}(G)| = 4 \leq C \leq 4 \cdot |\text{Sol}(G)| = \Rightarrow |\text{Sol}(F)| \leq 2 \leq 2 \cdot |\text{Sol}(F)|$ - From 4 to $(1 + \varepsilon)$-factor
- Construct $G = F_1 \wedge F_2 ... F_1^{\varepsilon}$ And then we can take $1^{\varepsilon}$-root.

Approach 2: $\text{thresh} = O(1^{\varepsilon^2})$ gives $(1 + \varepsilon)$-approximation directly

Techniques based on $\text{thresh} = O(1^{\varepsilon^2})$, despite worse complexity, e.g., ApproxMC scale significantly better than those based on $\text{thresh} = \text{constant}$.
Challenge 3 What is a small cell?

- A cell is small cell if it has \( \approx \) thresh solutions.
- **Approach 1**: thresh = constant \( \rightarrow \) 4-factor approximation
  - From 4 to 2-factor
    
    Let \( G = F_1 \land F_2 \) (i.e., two identical copies of \( F \))

    \[
    \frac{|\text{Sol}(G)|}{4} \leq C \leq 4 \cdot |\text{Sol}(G)| \implies \frac{|\text{Sol}(F)|}{2} \leq \sqrt{C} \leq 2 \cdot |\text{Sol}(F)|
    \]
Challenges

Challenge 3 What is a small cell?
- A cell is small cell if it has \( \approx \) thresh solutions.
- **Approach 1**: thresh = constant \( \rightarrow \) 4-factor approximation
  - From 4 to 2-factor
    Let \( G = F_1 \land F_2 \) (i.e., two identical copies of \( F \))
    \[
    \frac{|\text{Sol}(G)|}{4} \leq C \leq 4 \cdot |\text{Sol}(G)| \implies \frac{|\text{Sol}(F)|}{2} \leq \sqrt{C} \leq 2 \cdot |\text{Sol}(F)|
    \]
  - From 4 to \((1 + \varepsilon)\)-factor
    Construct \( G = F_1 \land F_2 \ldots F_{1/\varepsilon} \) And then we can take \( \frac{1}{\varepsilon} \)-root
Challenge 3 What is a small cell?

- A cell is small cell if it has $\approx$ thresh solutions.
- **Approach 1:** thresh $= \text{constant} \rightarrow 4$-factor approximation
  - From 4 to 2-factor
    Let $G = F_1 \land F_2$ (i.e., two identical copies of $F$)
    \[
    \frac{|\text{Sol}(G)|}{4} \leq C \leq 4 \cdot |\text{Sol}(G)| \implies \frac{|\text{Sol}(F)|}{2} \leq \sqrt{C} \leq 2 \cdot |\text{Sol}(F)|
    \]
  - From 4 to $(1 + \varepsilon)$-factor
    Construct $G = F_1 \land F_2 \ldots F_{\frac{1}{\varepsilon}}$ And then we can take $\frac{1}{\varepsilon}$-root
- **Approach 2:** thresh $= \mathcal{O}(\frac{1}{\varepsilon^2})$ gives $(1 + \varepsilon)$-approximation directly
Challenges

Challenge 3 What is a small cell?

- A cell is small cell if it has \( \approx \) thresh solutions.
- **Approach 1:** thresh = constant \( \rightarrow \) 4-factor approximation
  - From 4 to 2-factor
    Let \( G = F_1 \land F_2 \) (i.e., two identical copies of \( F \))
    \[
    \frac{|\text{Sol}(G)|}{4} \leq C \leq 4 \cdot |\text{Sol}(G)| \implies \frac{|\text{Sol}(F)|}{2} \leq \sqrt{C} \leq 2 \cdot |\text{Sol}(F)|
    \]
    - From 4 to \( (1 + \epsilon) \)-factor
      Construct \( G = F_1 \land F_2 \ldots F_{1/\epsilon} \) And then we can take \( \frac{1}{\epsilon} \)-root
- **Approach 2:** thresh = \( \mathcal{O}\left(\frac{1}{\epsilon^2}\right) \) gives \( (1 + \epsilon) \)-approximation directly

Techniques based on thresh = \( \mathcal{O}\left(\frac{1}{\epsilon^2}\right) \), despite worse complexity, e.g., ApproxMC scale significantly better than those based on thresh = constant.

The performance of SAT solvers depend on the formulas
ApproxMC

**Theorem (Correctness)**

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

**Theorem (Complexity)**

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

Theorem (Correctness)

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

Theorem (Complexity)

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

Theorem (FPRAS for DNF; (MSV, FSTTCS 17; CP 18, IJCAI-19))

If \( F \) is a DNF formula, then \( \text{ApproxMC} \) is FPRAS – different from the Monte-Carlo based FPRAS for DNF (Karp, Luby 1983)
Improvements Over the Years

Solved instances

Constrained Counting ✓  Hashing Framework ✓

Constrained Sampling
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)]
• Approximate counting and almost-uniform sampling are inter-reducible (Jerrum, Valiant and Vazirani, 1986)

• Is the reduction efficient?
  – Almost-uniform sampler (JVV) require linear number of approximate counting calls
• Check if a randomly picked cell is *small*
  - If yes, pick a solution randomly from the randomly picked cell
• Check if a randomly picked cell is *small*
  – If yes, pick a solution randomly from randomly picked cell

Challenge: How many cells?
• Desired Number of cells: \( 2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}} \) (\( m^* = \log \frac{|\text{Sol}(F)|}{\text{thresh}} \))
• 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
    $$\Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq C \leq |\text{Sol}(F)|(1+\varepsilon) \right] \geq 1 - \delta$$
  - $\tilde{m} = \log \frac{C}{\text{thresh}}$
How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\text{Sol}(F)|}{\text{thresh}}$ ($m^* = \log_2 \frac{|\text{Sol}(F)|}{\text{thresh}}$)
  - $\text{ApproxMC}(F, \varepsilon, \delta)$ returns $C$ such that
    $$\Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq C \leq |\text{Sol}(F)|(1+\varepsilon) \right] \geq 1 - \delta$$
  - $\tilde{m} = \log_2 \frac{C}{\text{thresh}}$
  - Check for $m = \tilde{m} - 1, \tilde{m}, \tilde{m} + 1$ if a randomly chosen cell is small

- $\Pr[y \text{ is output }] = \Pr[y \text{ is chosen}] \Pr[\text{Cell is small } | \text{ y is in cell}]$

- The conditioning in $\Pr[\text{Cell is small } | \text{ y is in cell}]$ leads to requirement of 3-wise independence of 2-wise independence.

(CMV14, CFMSV14, CFMSV15, SGM20)
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)**

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

**Theorem (Query)**

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

- Prior work required $n$ calls to approximate counter (Jerrum, Valiant and Vazirani, 1986)
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

- Prior work required \(n\) calls to approximate counter \((\text{Jerrum, Valiant and Vazirani, 1986})\)
- JVV employs 2-wise independent hash functions
- UniGen employs 3-wise independent hash functions
Theoretical Guarantees

Theorem (Almost-Uniformity)

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

Theorem (Query)

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

- Prior work required \( n \) calls to approximate counter \( (\text{Jerrum, Valiant and Vazirani, 1986}) \)

- JVV employs 2-wise independent hash functions
- UniGen employs 3-wise independent hash functions

Random XORs are 3-wise independent
Quiz Time: Uniformity

- Benchmark: case110.cnf; \#var: 287; \#clauses: 1263
- Total Runs: \(4 \times 10^6\); Total Solutions: 16384
Statistically Indistinguishable

- Benchmark: case110.cnf; \#var: 287; \#clauses: 1263
- Total Runs: $4 \times 10^6$; Total Solutions: 16384
Now that SAT is "easy", it is time to look beyond satisfiability
Improvements Over the Years

ApproxMC2 (2016)  
ApproxMC3 (2019)  
ApproxMC4 (2020)  
ApproxMC4.2 (2021)  
UniGen4 (2021)  
UniGen3 (2020)  
UniGen2.5 (2019)  
UniGen2 (2015)  

Time (s)  
Solved instances
Enabling “Beyond NP” Revolution

Challenge Problems
Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid

Security    Leakage Measurement for C++ program with 1K lines

Hardware Verification  Handling SMT formulas with 10K nodes
Enabling “Beyond NP” Revolution

Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid

Security  Leakage Measurement for C++ program with 1K lines

Hardware Verification  Handling SMT formulas with 10K nodes

Technical Directions

• Tighter integration between solvers and algorithms

• Handling weighted distributions: Connections to theory of integration

• Verification of sampling and counting
Enabling “Beyond NP” Revolution

Challenge Problems

Civil Engineering  Reliability for Los Angeles Transmission Grid

Security  Leakage Measurement for C++ program with 1K lines

Hardware Verification  Handling SMT formulas with 10K nodes

Technical Directions

• Tighter integration between solvers and algorithms
• Handling weighted distributions: Connections to theory of integration
• Verification of sampling and counting

Questions?
Reliability of Critical Infrastructure Networks

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

**Figure:** 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 $\Pr[\text{s and t are disconnected}]$?
- $\pi : \text{Configuration (of network) denoted by a 0/1 vector of size } |E|$
- $W(\pi) = \Pr(\pi)$

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

• failure probability $g : E \rightarrow [0, 1]$

• Compute $\Pr[\text{s and t are disconnected}]$?

• $\pi : \text{Configuration (of network) denoted by a 0/1 vector of size } |E|$

• $W(\pi) = \Pr(\pi)$

• $\pi_{s,t} : \text{configuration where } s \text{ and } t \text{ are disconnected}$
  – Represented as a solution to set of constraints over edge variables
• \( G = (V, E) \); 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
• \( \Pr[s \text{ and } t \text{ are disconnected}] = \sum_{\pi_{s,t}} W(\pi_{s,t}) \)
Reliability of Critical Infrastructure Networks

- $G = (V, E)$; source node: $s$ and terminal node $t$
- Failure probability $g : E \rightarrow [0, 1]$
- Compute $\Pr[s$ and $t$ 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
- $\Pr[s$ and $t$ are disconnected$] = \sum_{\pi_{s,t}} W(\pi_{s,t})$
  (DMPV, AAAI 17, ICASP-13, RESS 2019)
Reliability of Critical Infrastructure Networks

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

Timeout = 1000 seconds

( DMPV, AAAI17)
Reliability of Critical Infrastructure Networks

Figure: 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 \( \Pr[ t \text{ is disconnected}] \)?

Timeout = 1000 seconds