On Hashing-Based Approaches to Approximate DNF-Counting

Kuldeep S. Meel[1], **Aditya A. Shrotri[2]**, Moshe Y. Vardi[2]

[1] School of Computing, National University of Singapore

[2] Department of Computer Science, Rice University

At a glance

- Boolean formulas
  - Variables take values T/F
  - Logical Operations AND, OR, NOT, XOR …

- Formula encodings:
  - Conjunctive Normal Form (CNF)
    - $(X_1 \vee \neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee X_4) \wedge ..$
  - **Disjunctive Normal Form (DNF)**
    - $(X_1 \wedge \neg X_2 \wedge \neg X_3) \vee (\neg X_1 \wedge X_4) \vee ..$

At a glance

- **Counting Problem**
  - number of satisfying assignments?

- DNF-Counting Applications
  - Probabilistic Databases
  - Network Reliability

- CNF-Counting Applications
  - Probabilistic Inference
  - Information Leakage

At a glance

- Counting problem is hard [Valiant,'79]
  - "#P-Complete"

- Relaxation: **Approximate Counting**
  - Challenge: *Efficiently* find an approximate answer that is guaranteed to be close

At a glance

- Hash functions show promise!
- Hashing-based Approximate CNF-Counting
  - Improvements over 5+ years
    - [Chakraborty et al. ’13,’14,’16]
  - Scales to large formulas
- Hashing-based Approximate DNF-Counting [Chakraborty et al. ‘16]

At a glance

- Hash functions show promise!

- Hashing-based Approximate CNF-Counting
  - Improvements over 5+ years
    - [Chakraborty et al. ’13,’14,’16]
  - Scales to large formulas

- Hashing-based Approximate DNF-Counting [Chakraborty et al. ‘16]
  - Poor time complexity

At a glance

- Problem:
  - Approximate DNF-Counting
- Our Solution Strategy:
  - Design *efficient* hashing techniques
- Contributions
  - 3 algorithmic improvements to the hashing framework
- Result
  - Reduction in complexity from cubic to linear
- Significance
  - Power and versatility of hashing

Boolean Formulas and the Counting problem

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- # variables n = 3

Disjunctive Normal Form

$$\varphi = \underbrace{(\neg X_1 \wedge X_2)}_{C_1} \vee \underbrace{(X_2 \wedge X_3)}_{C_2} \vee \underbrace{(X_1 \wedge X_3)}_{C_3}$$

Cubes: C_1 C_2 C_3

Disjunctive Normal Form

$$\varphi = \underbrace{(\neg X_1 \wedge X_2)}_{C_1} \vee \underbrace{(X_2 \wedge X_3)}_{C_2} \vee \underbrace{(X_1 \wedge X_3)}_{C_3}$$

Cubes: C_1 C_2 C_3

- # cubes m = 3

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- Assignment: X_1 = F , X_2 = T , X_3 = T

Disjunctive Normal Form

$$\varphi = (\neg X_1 \land X_2) \lor (X_2 \land X_3) \lor (X_1 \land X_3)$$

- Assignment: $X_1 = F$, $X_2 = T$, $X_3 = T$

$(\neg F \land T) \lor (T \land T) \lor (F \land T)$

$= T \lor T \lor F$

$= T$

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- Assignment: X_1 = F , X_2 = T , X_3 = T
  - σ = <0, 1, 1>
  - σ satisfies C_2 ⇒ σ satisfies φ
    - $\sigma \vDash C_2$
    - $\sigma \vDash \varphi$

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- Assignment: X_1 = F , X_2 = T , X_3 = T
  - σ = <0, 1, 1>
  - σ satisfies C_2 ⇒ σ satisfies φ
    - $\sigma \models C_2$
    - $\sigma \models \varphi$
- Checking $\sigma \models \varphi$ takes linear time

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- Universe of assignments U = $\{0,1\}^n$

Disjunctive Normal Form

$$\varphi = (\neg X_1 \land X_2) \lor (X_2 \land X_3) \lor (X_1 \land X_3)$$

- Universe of assignments $U = \{0,1\}^n$
- Set of solutions $S_\varphi = \{ \sigma \in U : \sigma \models \varphi \}$

Disjunctive Normal Form

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- Universe of assignments $U = \{0,1\}^n$
- Set of solutions $S_\varphi = \{ \sigma \in U : \sigma \vDash \varphi \}$
- DNF-Counting: Determine $|S_\varphi|$ "Count of φ"

DNF Counting Problem

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$

- $|U| = 2^3 = 8$
- $S_\varphi = \{ \langle 0, 1, 0 \rangle, \langle 0, 1, 1 \rangle, \langle 1,0,1 \rangle, \langle 1, 1, 1 \rangle \}$
- $|S_\varphi| = 4$

Hardness of DNF-Counting

- Checking satisfiability
  - UNSAT: $.. \vee (.. \wedge X_i \wedge \neg X_i \wedge ..) \vee ..$

Hardness of DNF-Counting

- Enumerating satisfying assignments
  - $\varphi = (X_1 \land \neg X_2) \lor (X_3 \land X_4) \lor \ldots$
    - <1 , 0, 0, 0>
    - <1 , 0, 0, 1>
    - <1 , 0, 1, 0>
    - <1 , 0, 1, 1>
  - S_{c_1} = {<1 , 0, 0, 0>, <1 , 0, 0, 1>, <1 , 0, 1, 0>, <1 , 0, 1, 1>}

Hardness of DNF-Counting

- Enumerating satisfying assignments
  - $\varphi = (X_1 \wedge \neg X_2) \vee (X_3 \wedge X_4) \vee \ldots$
    - <1 , 0, 0, 0>
    - <1 , 0, 0, 1>
    - <1 , 0, 1, 0>
    - <1 , 0, 1, 1>
  - S_{c_1} = {<1 , 0, 0, 0>, <1 , 0, 0, 1>, <1 , 0, 1, 0>, <1 , 0, 1, 1>}
  - $S_{c_1} \cap S_{c_2}$ is not empty

Hardness of DNF-Counting

- Enumerating satisfying assignments
  - $\varphi = (X_1 \wedge \neg X_2) \vee (X_3 \wedge X_4) \vee \ldots$
    - <1 , 0, 0, 0>
    - <1 , 0, 0, 1>
    - <1 , 0, 1, 0>
    - <1 , 0, 1, 1>
  - S_{c_1} = {<1 , 0, 0, 0>, <1 , 0, 0, 1>, <1 , 0, 1, 0>, <1 , 0, 1, 1>}
  - $S_{c_1} \cap S_{c_2}$ is not empty $\Rightarrow$ $|S_\varphi| = |S_{c_1} \cup S_{c_2} \cup S_{c_3} \ldots \cup S_{c_m}|$

Hardness of DNF-Counting

- DNF-Counting is #P-Complete [Valiant,’79]
  - Class contains entire Polynomial Hierarchy!

- Need to Approximate!

Approximate DNF-Counting

Input: DNF Formula φ

Tolerance ε $\quad 0 < \varepsilon < 1$

Confidence δ $\quad 0 < \delta < 1$

Output: Approximate Count C s.t.

$$\Pr\left[\, |S_\varphi| \cdot (1-\varepsilon) < C < |S_\varphi| \cdot (1+\varepsilon) \,\right] > 1-\delta$$

Approximate DNF-Counting

Input: DNF Formula φ

Tolerance ε $0 < \varepsilon < 1$

Confidence δ $0 < \delta < 1$

Output: Approximate Count C s.t.

$$\Pr\left[\, |S_{\varphi}| \cdot (1-\varepsilon) < C < |S_{\varphi}| \cdot (1+\varepsilon) \,\right] > 1-\delta$$

Challenge: Design a poly(m, n, $\frac{1}{\epsilon}$, $\log(\frac{1}{\delta})$) time algorithm "**FPRAS**"

where m = #cubes n = #vars

Previous Work

1. Monte Carlo Sampling for Approximate DNF-Counting
   1. [Karp, Luby ‘83]
   2. [Karp, Luby, Madras ‘89]
   3. [Dagum, Karp, Luby, Ross ‘00]

2. Hashing-Based Algorithms for Approximate DNF-Counting
   1. Originated with [Sipser, ‘83], [Bellare et al. ‘98], [Gomes et al. ‘06]

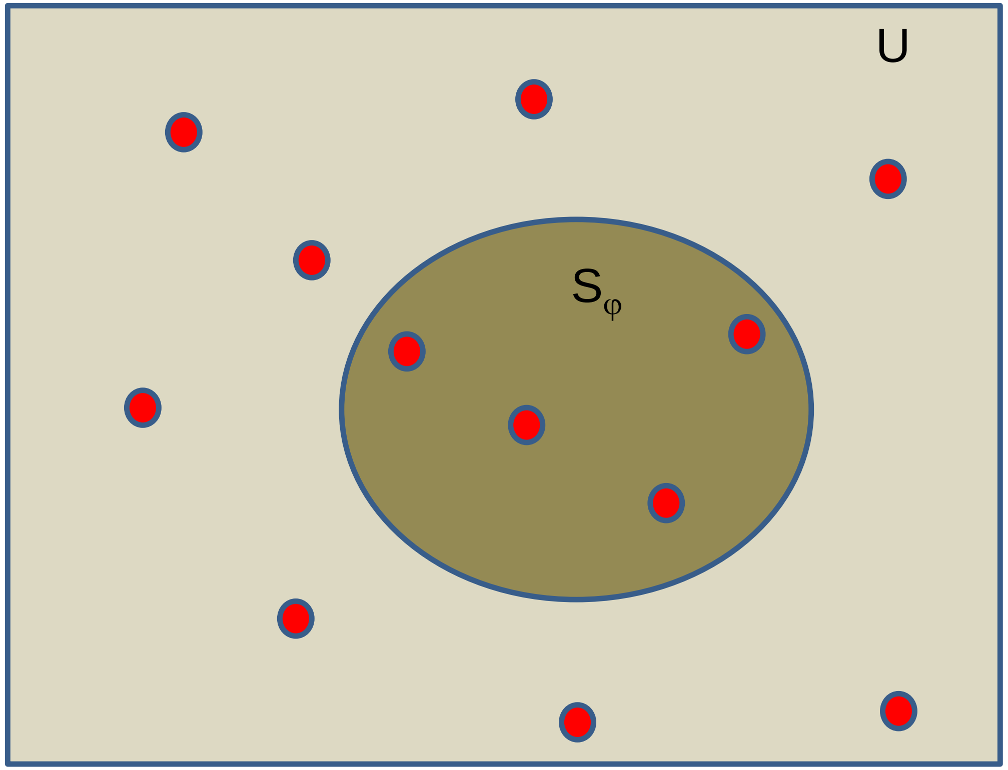
   2. **DNF-Counting FPRAS:** [Chakraborty et al., ‘16]

Method 1: Monte Carlo Sampling

- Randomized algorithms based on drawing independent samples

Method 1: Monte Carlo Sampling

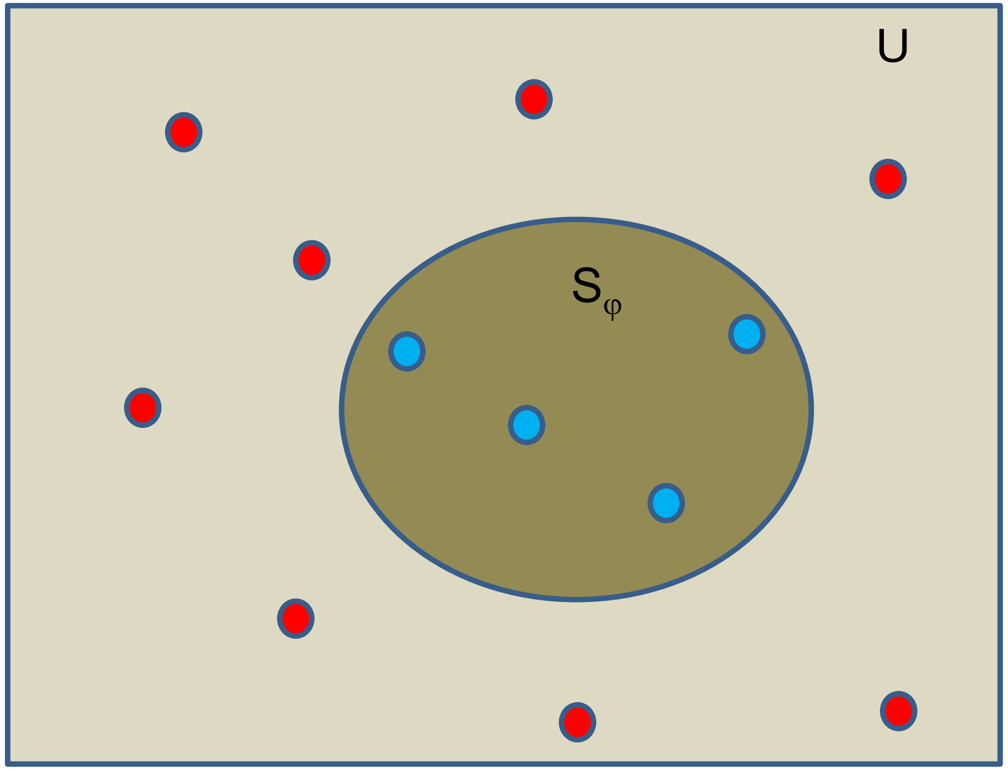
- Randomized algorithms based on drawing independent samples



- Randomly sample from U
- "Throwing darts"

Method 1: Monte Carlo Sampling

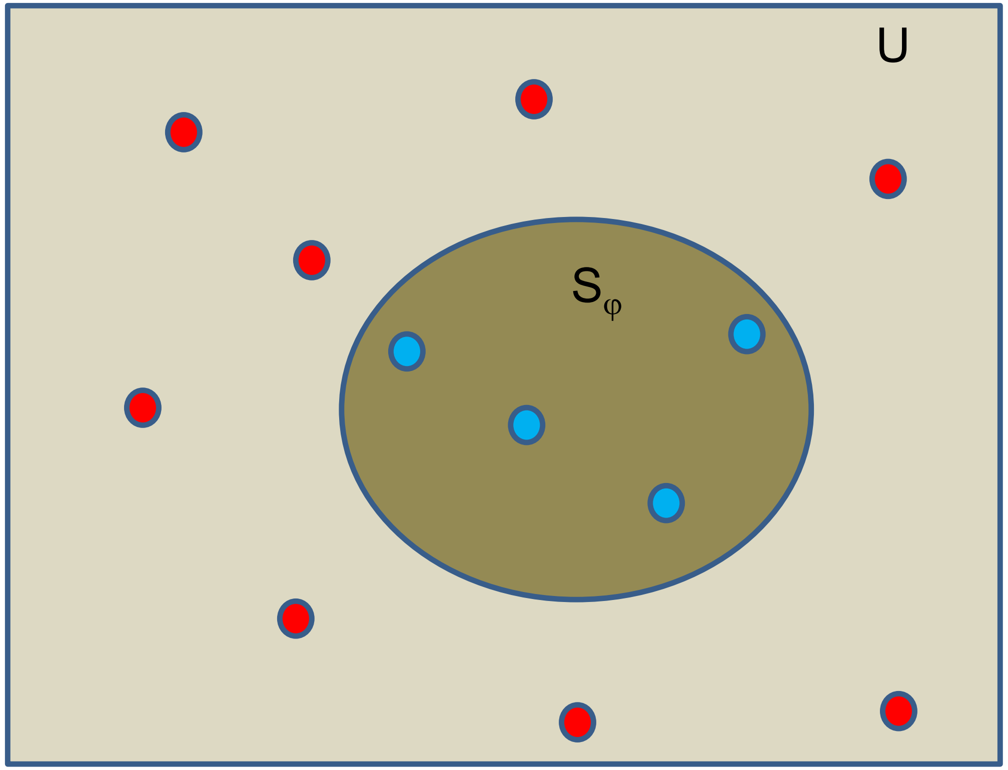
- Randomized algorithms based on drawing independent samples



- Randomly sample from U
- "Throwing darts"
- Count darts on target
- $|S_\varphi| \approx \frac{\#\bullet}{\#\bullet + \#\bullet} * |U|$

Method 1: Monte Carlo Sampling

- Randomized algorithms based on drawing independent samples

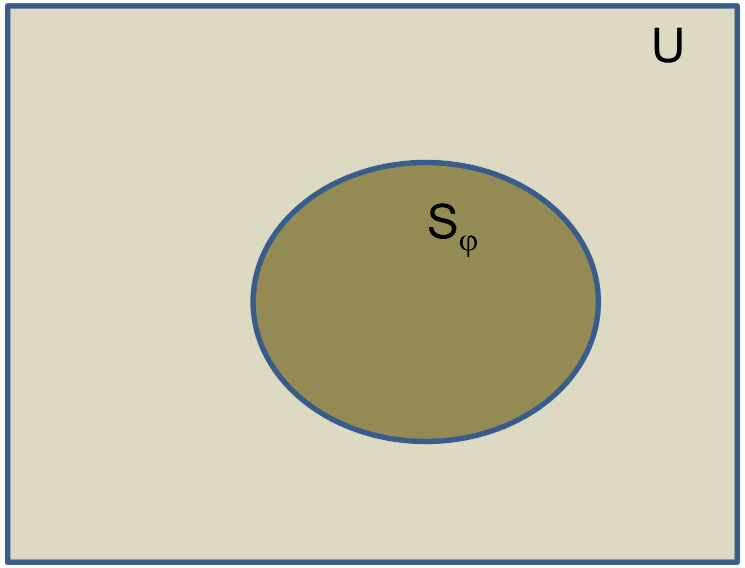


- Randomly sample from U
- "Throwing darts"
- Count darts on target
- $|S_\varphi| \approx \frac{\#\text{blue}}{\#\text{blue} + \#\text{red}} * |U|$
- $O(m \cdot n \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$ [Karp et al., '89]

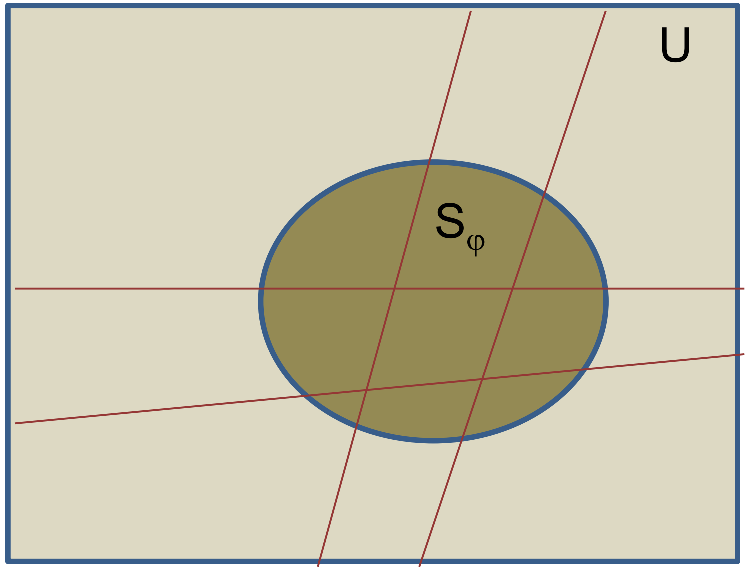
where m = #cubes n = #vars

Method 2: Hashing - Based

"Cutting slices of a pie"

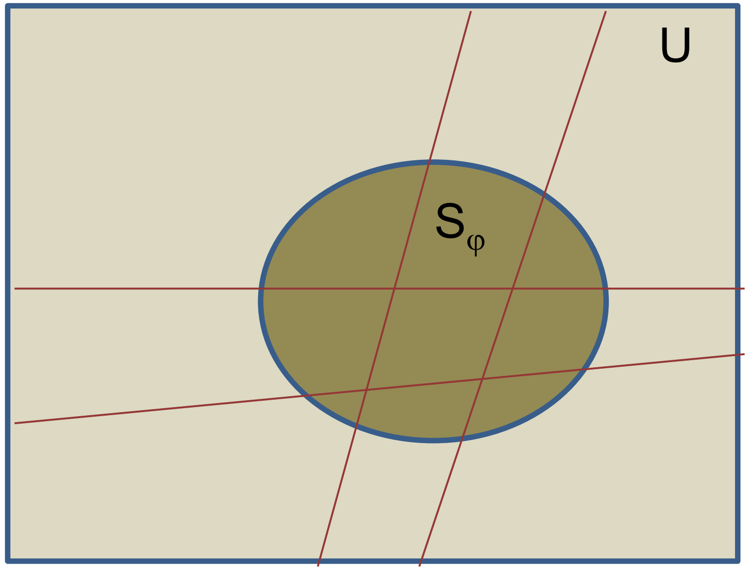


Method 2: Hashing - Based



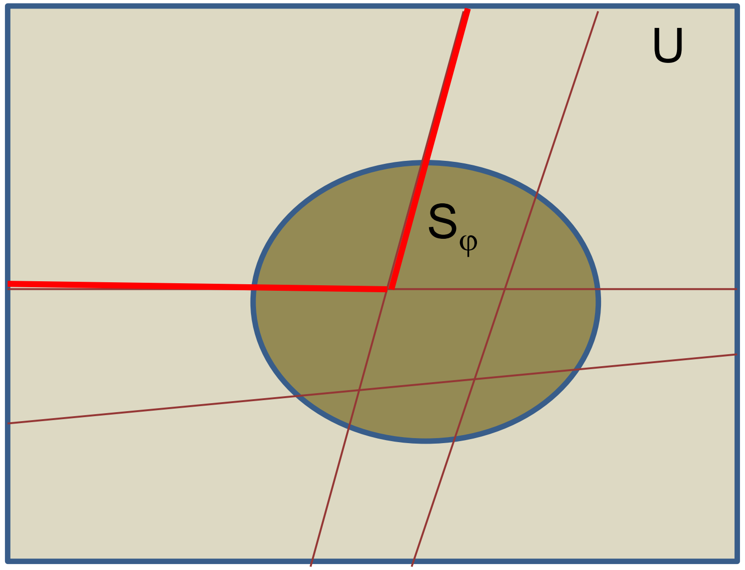
- Partition universe into ‘cells’

Method 2: Hashing - Based



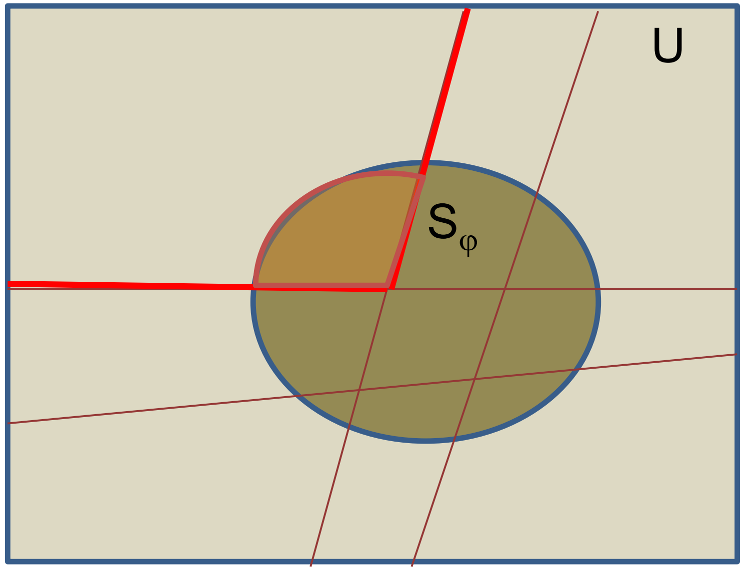
- Partition universe into 'cells'
- Ensure 'roughly' equal solutions to φ in each cell

Method 2: Hashing - Based



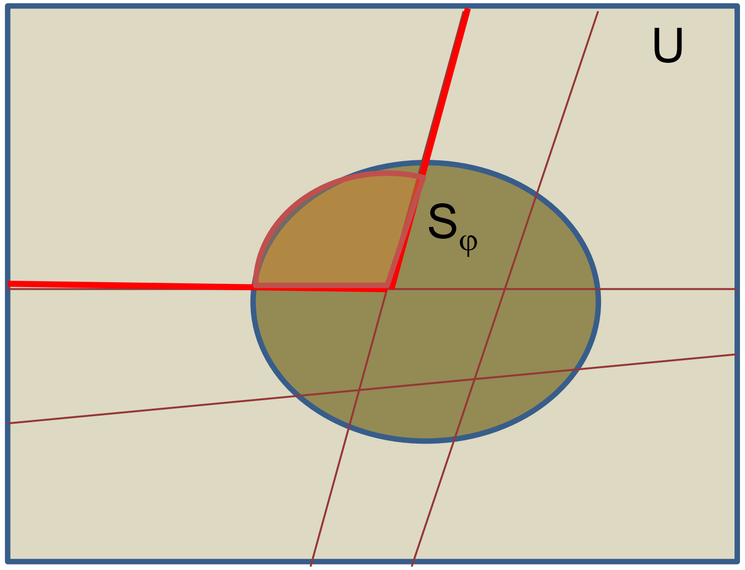
- Partition universe into 'cells'
- Ensure 'roughly' equal solutions to φ in each cell
- Pick a cell at random

Method 2: Hashing - Based



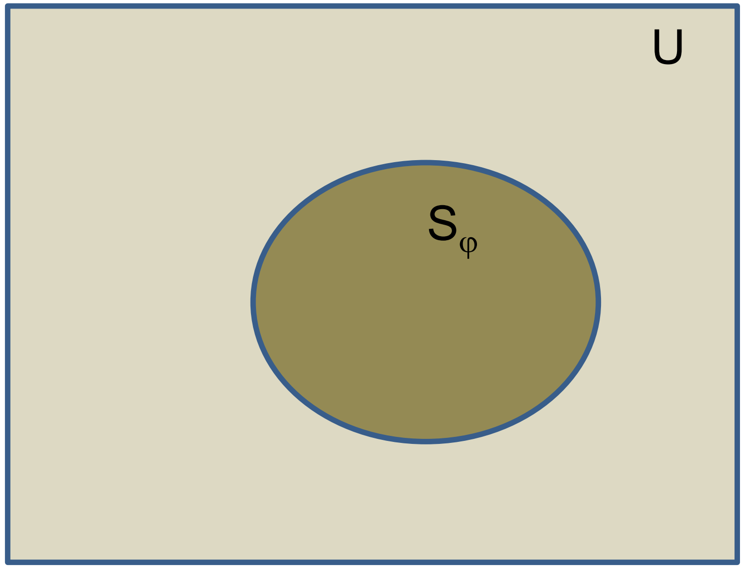
- Partition universe into 'cells'
- Ensure 'roughly' equal solutions to φ in each cell
- Pick a cell at random
- Count solutions only in the picked cell

Method 2: Hashing - Based



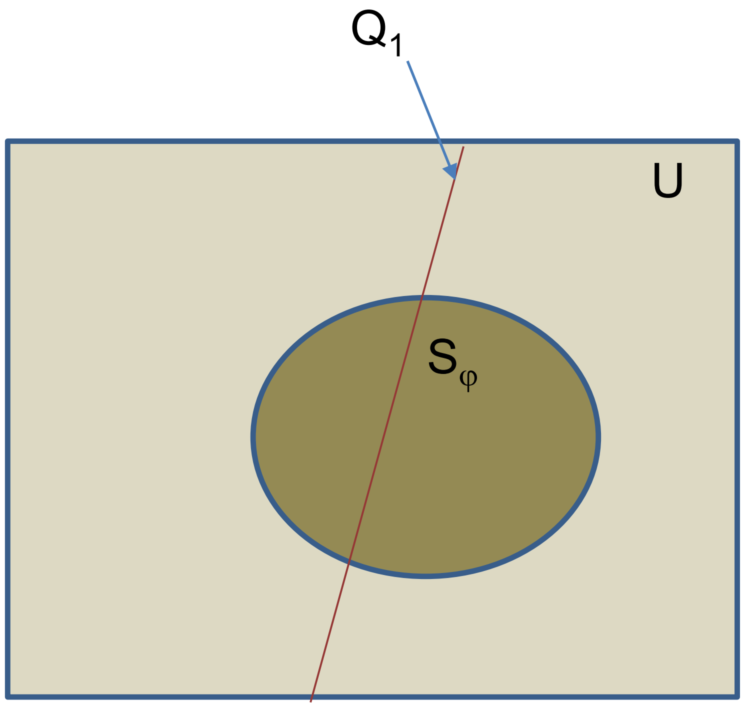
- Partition universe into ‘cells’
- Ensure ‘roughly’ equal solutions to φ in each cell
- Pick a cell at random
- Count solutions only in the picked cell
- Estimate count as #(solutions to φ in cell) x #cells

Method 2: Hashing - Based



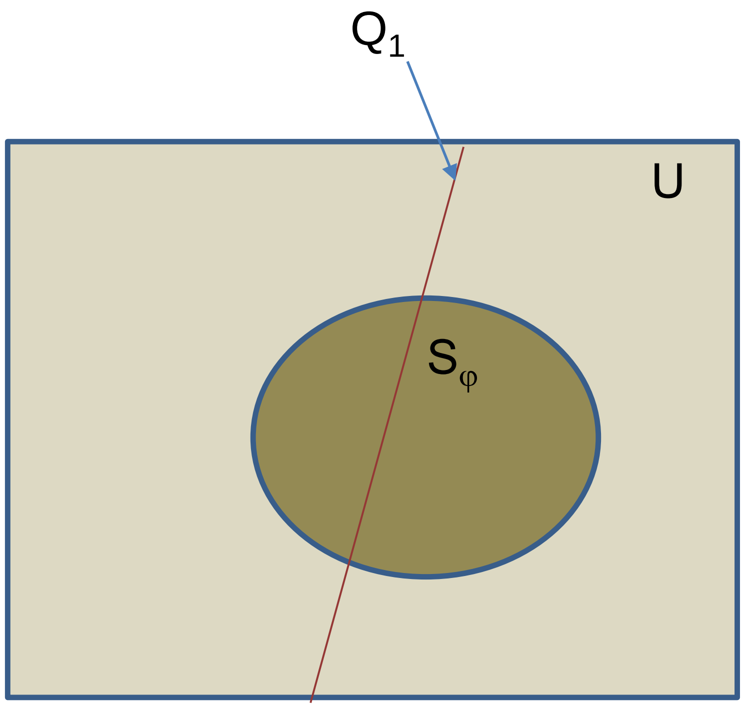
- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)

Method 2: Hashing - Based



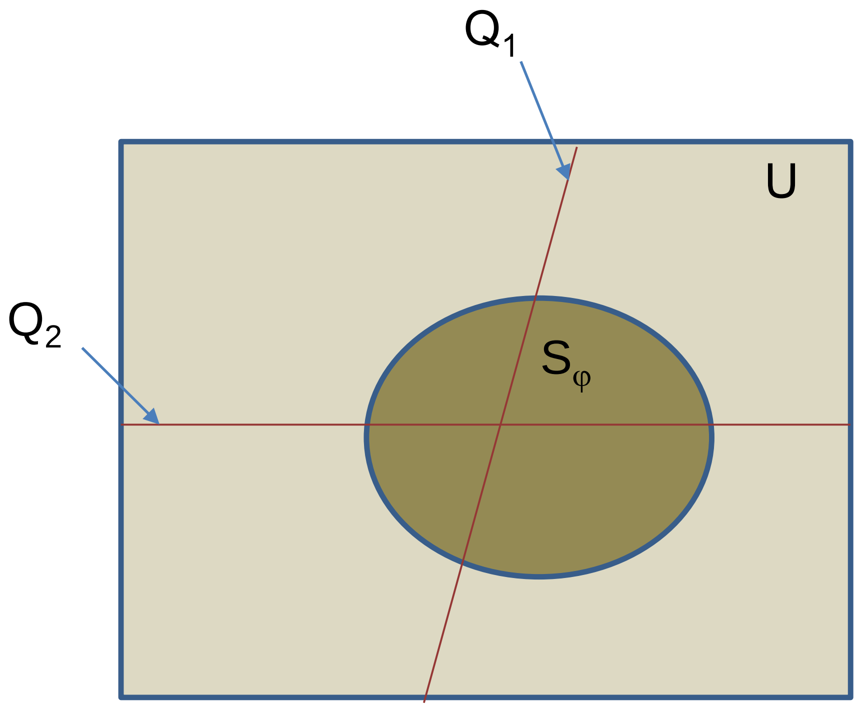
- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)

Method 2: Hashing - Based



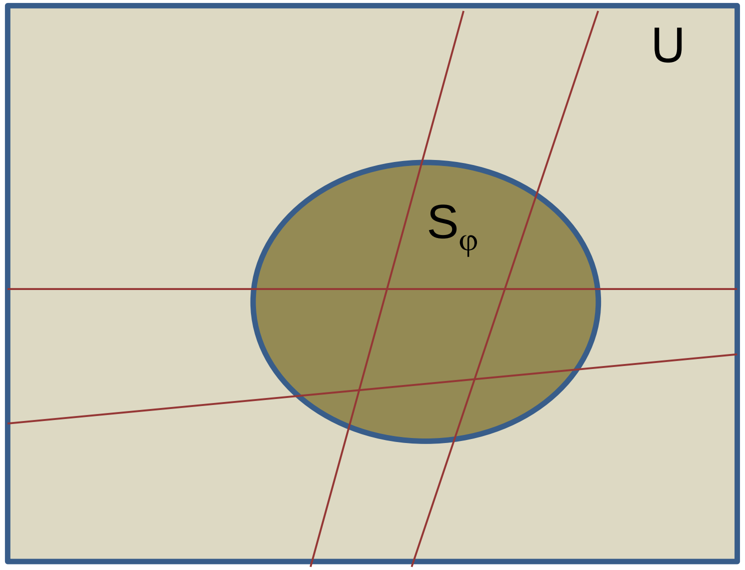
- Partition universe into ‘cells’
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - Q_1 partitions U into 2 ‘cells’
      - $Q_1 = 0$
      - $Q_1 = 1$

Method 2: Hashing - Based



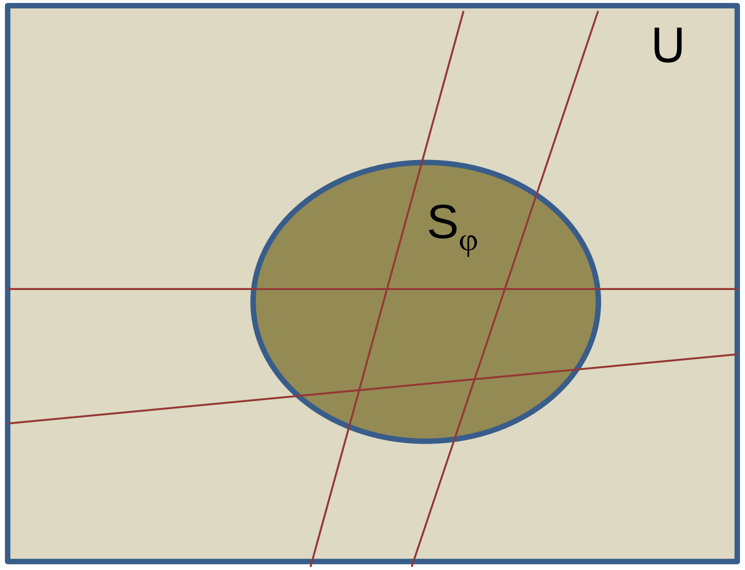
- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - Q_1 , Q_2 partitions U into 4 'cells'
      - $Q_{1\ ,}\ Q_2\ \in \{00\ ,\ 01,\ 10,\ 11\}$

Method 2: Hashing - Based



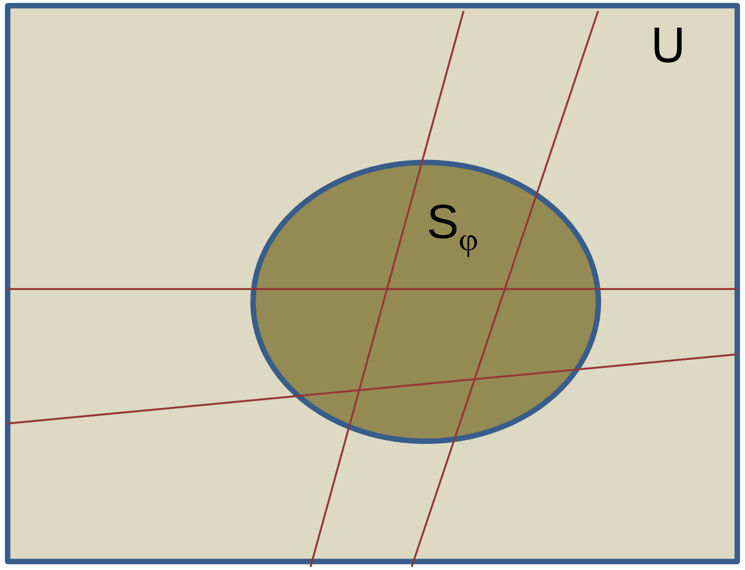
- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - Q_1 , Q_2 , Q_3 …. , Q_L partitions U into 2^L 'cells'

Method 2: Hashing - Based



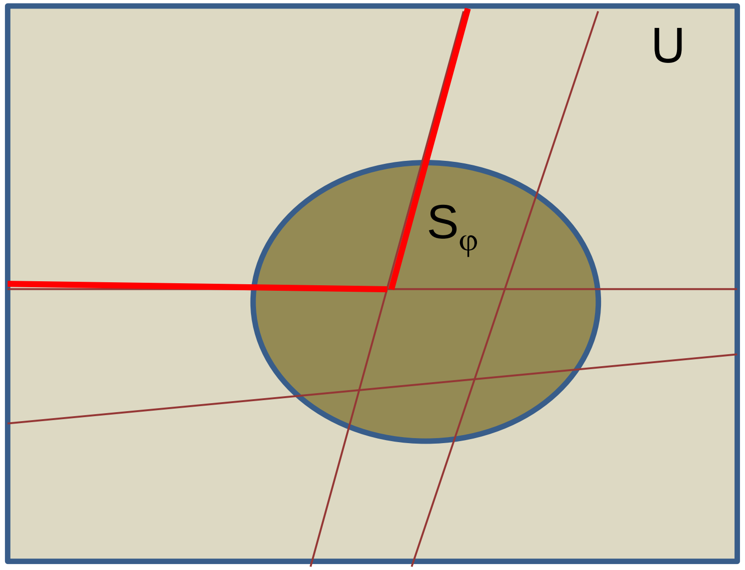
- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - Q_1 , Q_2 , Q_3 …. , Q_L partitions U into 2^L 'cells'
- Ensure 'roughly' equal solutions to φ in each cell

Method 2: Hashing - Based



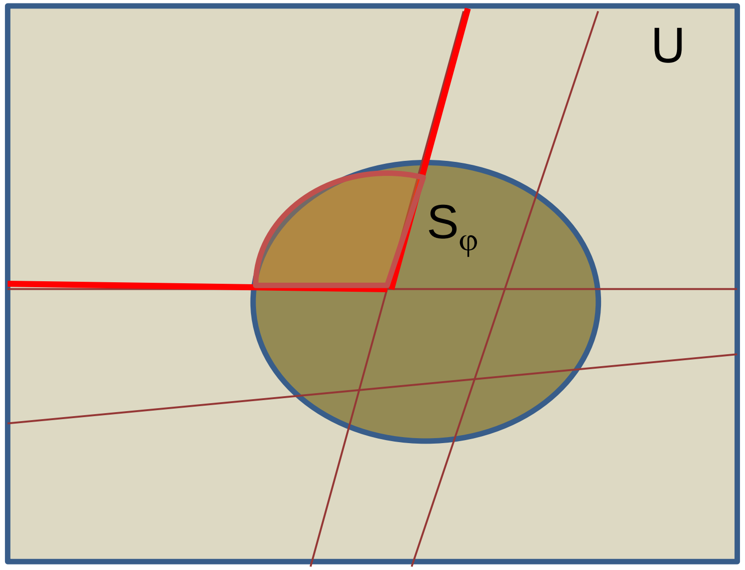
- Partition universe into ‘cells’
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - Q_1 , Q_2 , Q_3 …. , Q_L partitions U into 2^L ‘cells’
- Ensure ‘roughly’ equal solutions to φ in each cell
  - Use 2-universal Hash Families

Method 2: Hashing - Based



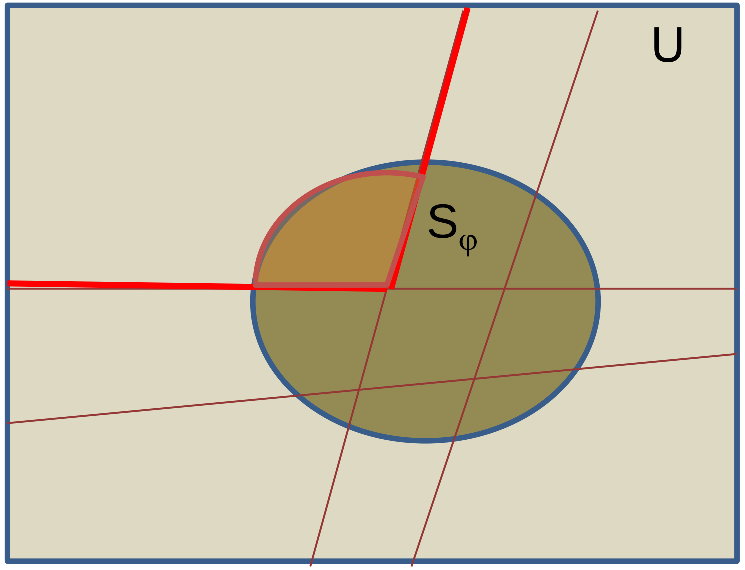
- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - $Q_1, Q_2, Q_3 \ldots, Q_L$ partitions U into 2^L 'cells'
- Ensure 'roughly' equal solutions to φ in each cell
  - Use 2-universal Hash Families
- Pick cell at random
  - $Q_1Q_2Q_3 \ldots Q_L = 0100\ldots1$

Method 2: Hashing - Based



- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - Q_1 , Q_2 , Q_3 …. , Q_L partitions U into 2^L 'cells'
- Ensure 'roughly' equal solutions to φ in each cell
  - Use 2-universal Hash Families
- Pick cell at random
  - $Q_1Q_2Q_3 \ldots. Q_L = 0100\ldots1$
- Count solutions in the picked cell
  - $\varphi' = \varphi \wedge Q$
    - $Q = (Q_1 \Leftrightarrow 0) \wedge (Q_2 \Leftrightarrow 1) \wedge (Q_3 \Leftrightarrow 0) \; .. \; (Q_L \Leftrightarrow 1)$
  - Calculate $|S_{\varphi'}|$

Method 2: Hashing - Based



- Partition universe into 'cells'
  - **Hash function**: Conjunction of Hash Constraints
  - **Hash Constraint:** Boolean formula randomly chosen from special set (Hash family)
    - $Q_1, Q_2, Q_3 \ldots, Q_L$ partitions U into 2^L 'cells'
- Ensure 'roughly' equal solutions to φ in each cell
  - Use 2-universal Hash Families
- Pick cell at random
  - $Q_1Q_2Q_3 \ldots Q_L = 0100\ldots1$
- Count solutions in the picked cell
  - $\varphi' = \varphi \wedge Q$
    - $Q = (Q_1 \Leftrightarrow 0) \wedge (Q_2 \Leftrightarrow 1) \wedge (Q_3 \Leftrightarrow 0) \; .. \; (Q_L \Leftrightarrow 1)$
  - Calculate $|S_{\varphi'}|$
- Estimate count as #(solutions to φ in cell) x #cells
  - $|S_\varphi| \approx |S_{\varphi'}|$ x (# cells)

Comparison of Approaches

Monte Carlo Sampling

- "Dart Throwing"
- DNF-Counting
  - Complexity: $O(m \cdot n \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$
    - [Karp et al., '89]

Hashing Based

- "Pie Slicing"
- DNF-Counting
  - Complexity: $O(m \cdot n^3 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$
    - '**ApproxMC**' [Chakraborty et al., '16]

where m = #cubes n = #vars

Comparison of Approaches

Monte Carlo Sampling

- "Dart Throwing"
- DNF-Counting
  - Complexity: $O(m \cdot n \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$
    - [Karp et al., '89]
- CNF-Counting
  - Does not scale well

Hashing Based

- "Pie Slicing"
- DNF-Counting
  - Complexity: $O(m \cdot n^3 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$
    - '**ApproxMC**' [Chakraborty et al., '16]
- CNF-Counting
  - Very successful in practice

where m = #cubes   n = #vars

Motivating Questions

- *Power of Hashing?*
- *Can ApproxMC be made competitive with state-of-the-art?*

Contributions

Better Hashing techniques for Approximate DNF-Counting

- Row Echelon Hash Family
- Symbolic Hashing
- Stochastic Cell-Counting

Row Echelon Hash Family

XOR Family of Hash Functions

- Boolean Formulas with only XOR $\oplus$
- 2-Universal
- Sampling procedure:
  - Pick each variable with probability 0.5
  - XOR picked variables

XOR Family: Hash Matrix

- Sample each constraint at random (n=7)
  - $Q_1 = X_1 \oplus X_4 \oplus X_5$
  - $Q_2 = X_3 \oplus X_4 \oplus X_6 \oplus X_7$
  - …
  - $Q_L = X_2 \oplus X_4 \oplus X_6$

XOR Family: Hash Matrix

- Sample each constraint at random (n=7)
  - $Q_1 = X_1 \oplus X_4 \oplus X_5$
  - $Q_2 = X_3 \oplus X_4 \oplus X_6 \oplus X_7$
  - …
  - $Q_L = X_2 \oplus X_4 \oplus X_6$
- Equivalent to sampling a 0/1 matrix of dimension L x 7

$$
\begin{array}{l}
X_1 \oplus X_4 \oplus X_5 \\
\wedge\ X_3 \oplus X_4 \oplus X_6 \oplus X_7 \\
\wedge\ \quad \ldots \\
\wedge\ X_2 \oplus X_4 \oplus X_6
\end{array}
\Rightarrow
\underbrace{\left|\begin{array}{ccccccc}
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 \\
 & & & & & & \ldots \\
0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}\right|}_{N=7}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\end{array}}\right\} L
$$

XOR Family: Counting using XORs

1. Sample a hash matrix
   - $Q_1 = X_1 \oplus X_4 \oplus X_5$
   - $Q_2 = X_3 \oplus X_4 \oplus X_6 \oplus X_7$
   - …
   - $Q_L = X_2 \oplus X_4 \oplus X_6$
2. Pick a cell (L bits) at random
   - 10…1
3. Count solutions to φ in cell
   - $\varphi' = \varphi \wedge Q$ where $Q = (Q_1 \Leftrightarrow 1) \wedge (Q_2 \Leftrightarrow 0) \wedge .. (Q_L \Leftrightarrow 1)$
   - Need to find $|S_{\varphi'}|$

XOR Family: Counting using XORs

1. Sample a hash matrix
   - $Q_1 = X_1 \oplus X_4 \oplus X_5$
   - $Q_2 = X_3 \oplus X_4 \oplus X_6 \oplus X_7$
   - …
   - $Q_L = X_2 \oplus X_4 \oplus X_6$
2. Pick a cell (L bits) at random
   - 10…1
3. Count solutions to φ in cell
   - $\varphi' = \varphi \wedge Q$ where $Q = (Q_1 \Leftrightarrow 1) \wedge (Q_2 \Leftrightarrow 0) \wedge .. (Q_L \Leftrightarrow 1)$
   - Need to find $|S_{\varphi'}|$: Enumerate solutions to Q and check if satisfy φ

XOR Family: Enumerating Solutions

Constraints		Cell
$X_1 \oplus X_4 \oplus X_5$	=	1
$\wedge\ X_3 \oplus X_4 \oplus X_6 \oplus X_7$	=	0
$\wedge\ \ldots$		..
$\wedge\ X_2 \oplus X_4 \oplus X_6$	=	1

$\Rightarrow$

Hash Matrix, Cell Vector:

$$\begin{vmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ \ldots \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{vmatrix} \begin{vmatrix} X_1 \\ X_2 \\ .. \\ X_7 \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ .. \\ 1 \end{vmatrix}$$

- Q is a system of linear equations (mod 2)
  - Simplify using Gaussian Elimination!

XOR Hash: Gaussian Elimination

- $\begin{bmatrix} 1 & 0 & 1 & .. & 1 \\ 1 & 1 & 0 & .. & 1 \\ 0 & 1 & 0 & .. & 1 \\ .. & .. & .. & .. & .. \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_{n'} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ .. \\ .. \\ 1 \end{bmatrix} \xrightarrow[\Longrightarrow]{\substack{Gaussian \\ Elimi.}} \begin{bmatrix} 1 & 0 & 0 & 0 & .. & 1 \\ 0 & 1 & 0 & 1 & .. & 1 \\ 0 & 0 & 1 & 1 & .. & 1 \\ 0 & 0 & 0 & 0 & .. & 0 \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_{n'} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ .. \\ 1 \end{bmatrix}$

Row Echelon Form

- Row Echelon Form:
  - First block is the identity matrix
  - Zero rows after non-zero rows

XOR Hash: Gaussian Elimination

- $\begin{bmatrix} 1 & 0 & 1 & .. & 1 \\ 1 & 1 & 0 & .. & 1 \\ 0 & 1 & 0 & .. & 1 \\ .. & .. & .. & .. & .. \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_{n'} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ .. \\ .. \\ 1 \end{bmatrix} \xrightarrow[Elimi.]{Gaussian} \begin{bmatrix} 1 & 0 & 0 & 0 & .. & 1 \\ 0 & 1 & 0 & 1 & .. & 1 \\ 0 & 0 & 1 & 1 & .. & 1 \\ 0 & 0 & 0 & 0 & .. & 0 \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_{n'} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ .. \\ 1 \end{bmatrix}$

Row Echelon Form

- Row Echelon Form:
  - First block is the identity matrix
  - Zero rows after non-zero rows
- Drawback: Gaussian Elimination is $O(n^3)$
  - ApproxMC Complexity $O(m \cdot n^3 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

Row Echelon Hash Family

n

L

$$\begin{bmatrix} 1 & 0 & 0.. & 0 & 1 & 1 \\ 0 & 1 & 0.. & 0 & 1 & 1 \\ 0 & 0 & 1.. & 0 & 0 & 0 \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0.. & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_n \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ 1 \end{bmatrix}$$

Sample a hash matrix in Row-Echelon form directly!

Row Echelon Hash Family

$$\overbrace{\left.L\right\{\begin{bmatrix} 1 & 0 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 1.. & 0 & \mathbf{0} & \mathbf{0} \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0.. & 1 & \mathbf{1} & \mathbf{1} \end{bmatrix}}^{n} \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_n \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ 1 \end{bmatrix}$$

Sample a hash matrix in Row-Echelon form directly!

- Only sample $L \times (n - L)$ non-identity part

Row Echelon Hash Family

$$\underbrace{}_{n} \quad L\left\{\begin{bmatrix} 1 & 0 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 1.. & 0 & \mathbf{0} & \mathbf{0} \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0.. & 1 & \mathbf{1} & \mathbf{1} \end{bmatrix}\right. \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_n \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ 1 \end{bmatrix}$$

Sample a hash matrix in Row-Echelon form directly!

- Only sample $L \times (n - L)$ non-identity part
- Eliminates expensive Gaussian Elimination step

Row Echelon Hash Family

$$L\left\{\overbrace{\begin{bmatrix} 1 & 0 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 1.. & 0 & \mathbf{0} & \mathbf{0} \\ .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0.. & 1 & \mathbf{1} & \mathbf{1} \end{bmatrix}}^{n}\right. \begin{bmatrix} X_1 \\ X_2 \\ .. \\ X_n \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ 1 \end{bmatrix}$$

Sample a hash matrix in Row-Echelon form directly!

- Only sample $L \times (n - L)$ non-identity part
- Eliminates expensive Gaussian Elimination step
- *Is it 2-Universal?*

Row Echelon Hash Family

$$\underbrace{}_{n}\quad L\left\{\begin{bmatrix} 1 & 0 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 0.. & 0 & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 1.. & 0 & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0.. & 1 & \mathbf{1} & \mathbf{1} \end{bmatrix}\right. \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Sample a hash matrix in Row-Echelon form directly!

- Only sample $L \times (n - L)$ non-identity part
- Eliminates expensive Gaussian Elimination step
- *Is it 2-Universal?* *Yes!*

- **Theorem:** Row-Echelon Family is 2-Universal

Hash Family Comparison

XOR Hash Family

- $O(n^2)$ space
- $O(n^3)$ time to generate (Gaussian Elim.)
- $O(n^2)$ time to enumerate one solution

Row Echelon Hash Family

- $O(n^2)$ space
- $O(n^2)$ time (Sampling the matrix)
- $O(n)$ time to enumerate one solution

where m = #cubes n = #vars

Hash Family Comparison

XOR Hash Family

- $O(n^2)$ space
- $O(n^3)$ time to generate (Gaussian Elim.)
- $O(n^2)$ time to enumerate one solution
- Complexity of **ApproxMC** (XOR hash) [Chakraborty et al, '16]:

  $O(m \cdot n^3 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

Row Echelon Hash Family

- $O(n^2)$ space
- **$O(n^2)$** time (Sampling the matrix)
- $O(n)$ time to enumerate one solution
- Complexity of **ApproxMC** with **RE hash**:

  $O(m \cdot n^2 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

where m = #cubes n = #vars

Hash Family Comparison

XOR Hash Family

- $O(n^2)$ space
- $O(n^3)$ time to generate (Gaussian Elim.)
- $O(n^2)$ time to enumerate one solution
- Complexity of **ApproxMC** (XOR hash) [Chakraborty et al, '16]:

  $O(m \cdot n^3 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

Row Echelon Hash Family

- $O(n^2)$ space
- **$O(n^2)$** time (Sampling the matrix)
- $O(n)$ time to enumerate one solution
- Complexity of **ApproxMC** with **RE hash**:

  $O(m \cdot n^2 \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

Karp et al. Algorithm: $O(m \cdot n \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

Symbolic Hashing and Stochastic Cell-Counting

Room for Improvement

- Time to sample a matrix from RE family: $L \times (n - L)$
  - If L ≈ n then sample time = O(n)

Room for Improvement

- Time to sample a matrix from RE family: $L \times (n - L)$
  - If L ≈ n then sample time = O(n)

- L ≈ n when $|S_{\varphi}| \approx |U|$

Room for Improvement

- Time to sample a matrix from RE family: $L \times (n - L)$
  - If L ≈ n then sample time = O(n)

- L ≈ n when $|S_\varphi| \approx |U|$

- But $|S_\varphi|$ can be much smaller than $|U|$
  - Exponentially smaller in worst case

Symbolic Hashing: Outline

1. Transform U to U' [Karp et al., '83]
   - $|S_\varphi|$ is polynomially smaller than $|U'|$ in worst case

2. Hash over the transformed universe U' (our contribution)

Symbolic Hashing: Space Transform

1. Transform U to U' [Karp et al., '83]

- $U = \{0,1\}^n$
  - Set of all assignments σ
  - Includes satisfying and unsatisfying assignments to φ

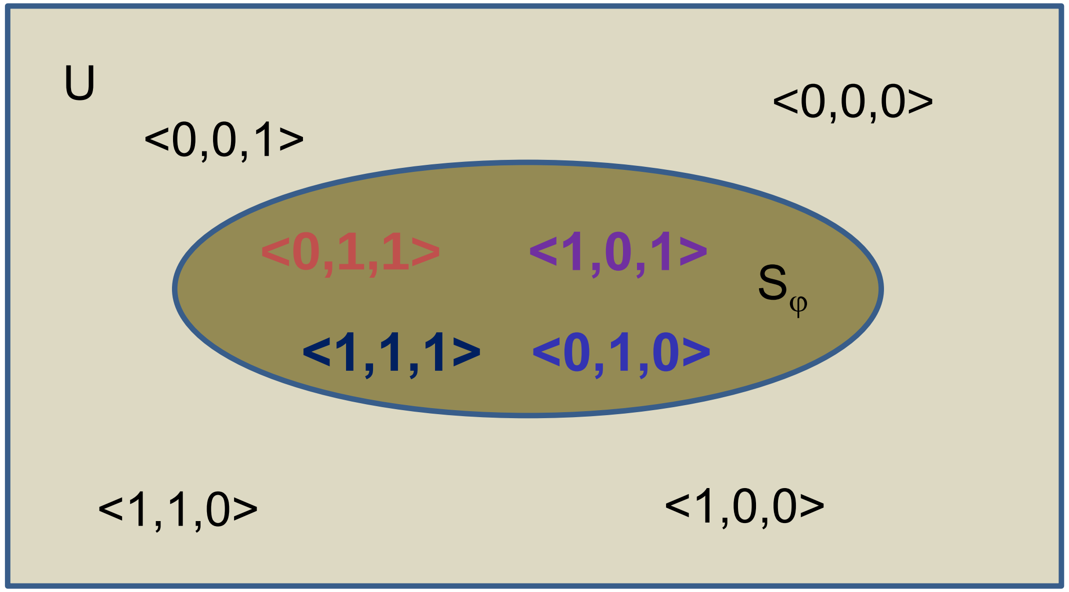
Symbolic Hashing: Space Transform

1. Transform U to U' [Karp et al., '83]

- $U' = \{ (\sigma , C_i) \mid \sigma \models C_i \}$
  - Multiset of satisfying assignments (Recall: $\sigma \models C_i \Rightarrow \sigma \models \varphi$)
  - Each σ occurs at most m = #cubes times
  - $|U'| \leqslant m.|S_\varphi|$

Symbolic Hashing: Space Transform Example

$$\varphi = (\neg X_1 \land X_2) \lor (X_2 \land X_3) \lor (X_1 \land X_3)$$

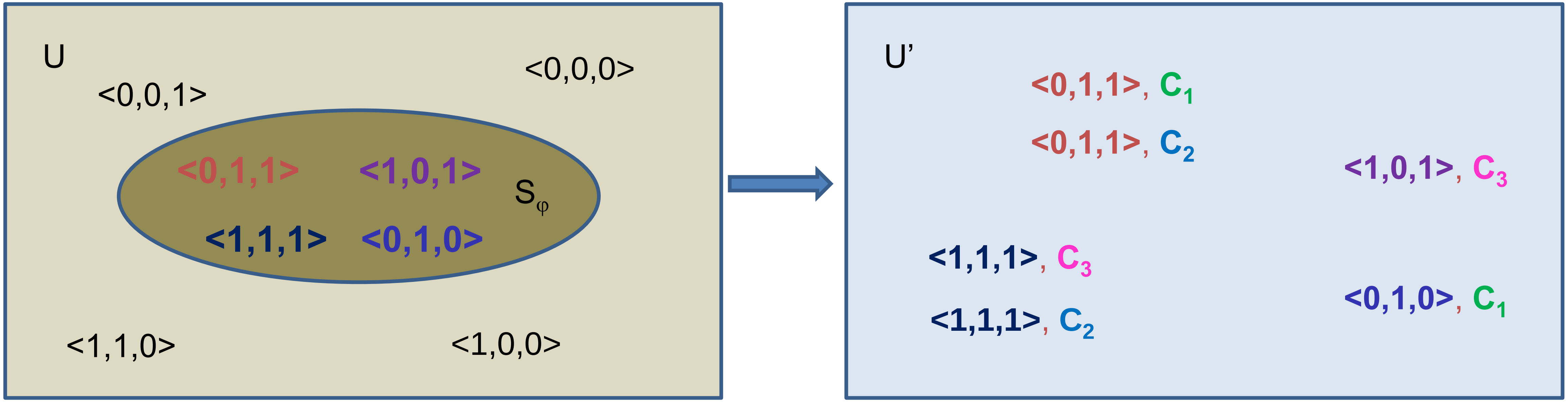


$|U| = 8$
$|S_\varphi| = 4$

$|U| \leqslant 2.|S_\varphi|$

Symbolic Hashing: Space Transform Example

$$\varphi = (\neg X_1 \wedge X_2) \vee (X_2 \wedge X_3) \vee (X_1 \wedge X_3)$$



$|U| = 8$
$|S_\varphi| = 4$

$|U| \leqslant 2.|S_\varphi|$

$|U'| = 6$
$|S_\varphi| = 4$

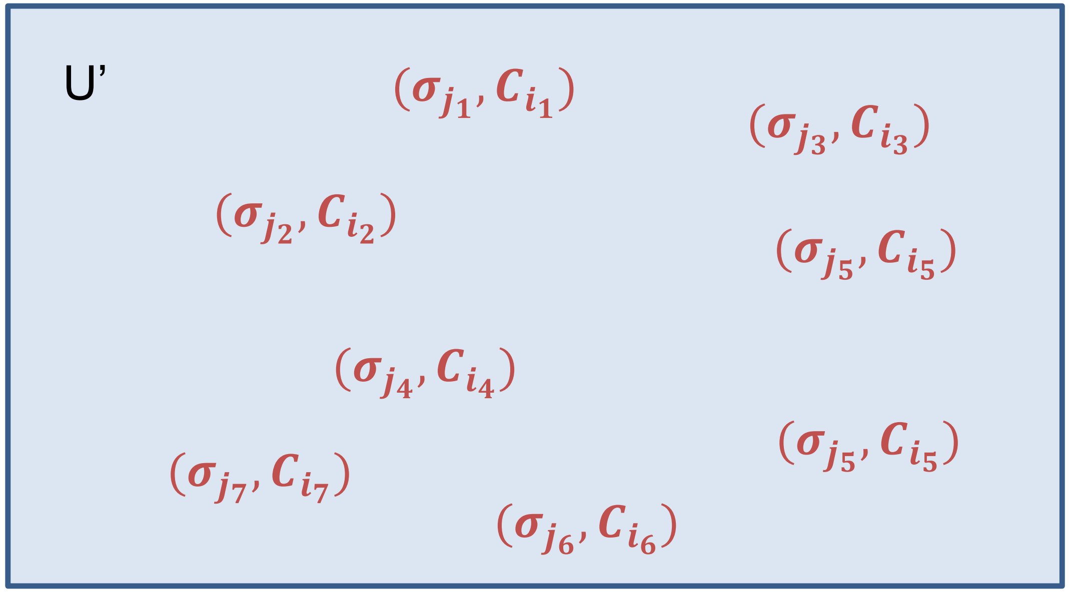
$|U'| \leqslant 1.5\ |S_\varphi|$

Symbolic Hashing: Hashing over U'

2. Partition $U' = \{ (\sigma , C_i) \mid \sigma \vDash C_i \}$
   - Use hash constraints with $n + \log(m)$ variables
     - $X_1 \ldots X_n$
     - $Y_1 \ldots Y_{\log(m)}$ - auxiliary variables
   - "Bit-Blasting"
     - First n bits select σ
     - Last log(m) bits select C_i
       - Binary representation of the index i

where m = #cubes   n = #vars

Symbolic Hashing: Counting over U’



Symbolic Hashing: Counting over U’

1) Add constraints ..

U’

(σ_{j_1}, C_{i_1})

(σ_{j_3}, C_{i_3})

(σ_{j_2}, C_{i_2})

(σ_{j_5}, C_{i_5})

(σ_{j_4}, C_{i_4})

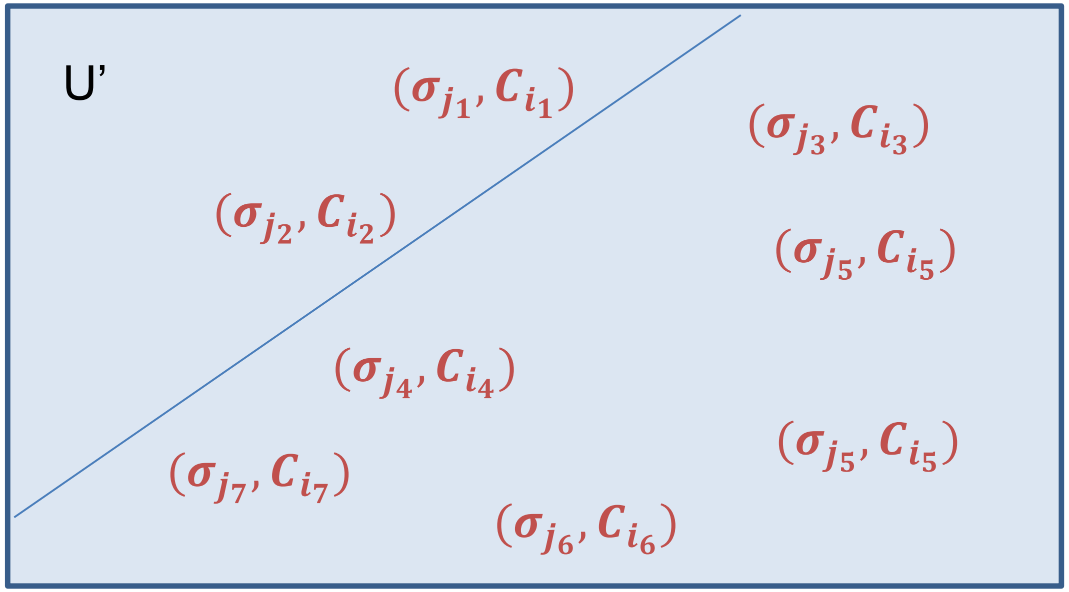
(σ_{j_5}, C_{i_5})

(σ_{j_7}, C_{i_7})

(σ_{j_6}, C_{i_6})

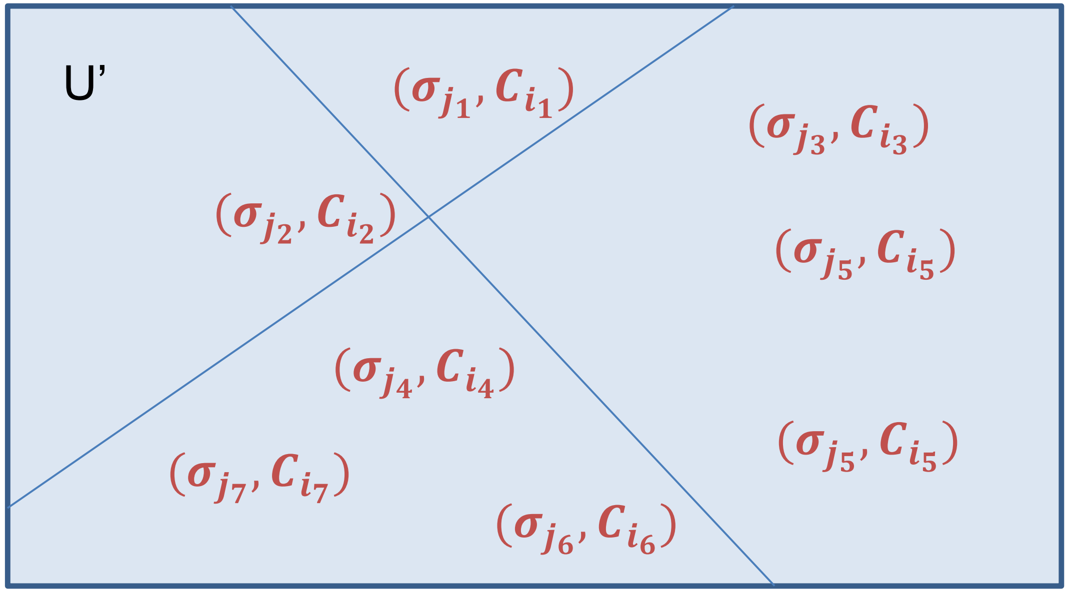
Symbolic Hashing: Counting over U’

1) Add constraints : Q_1



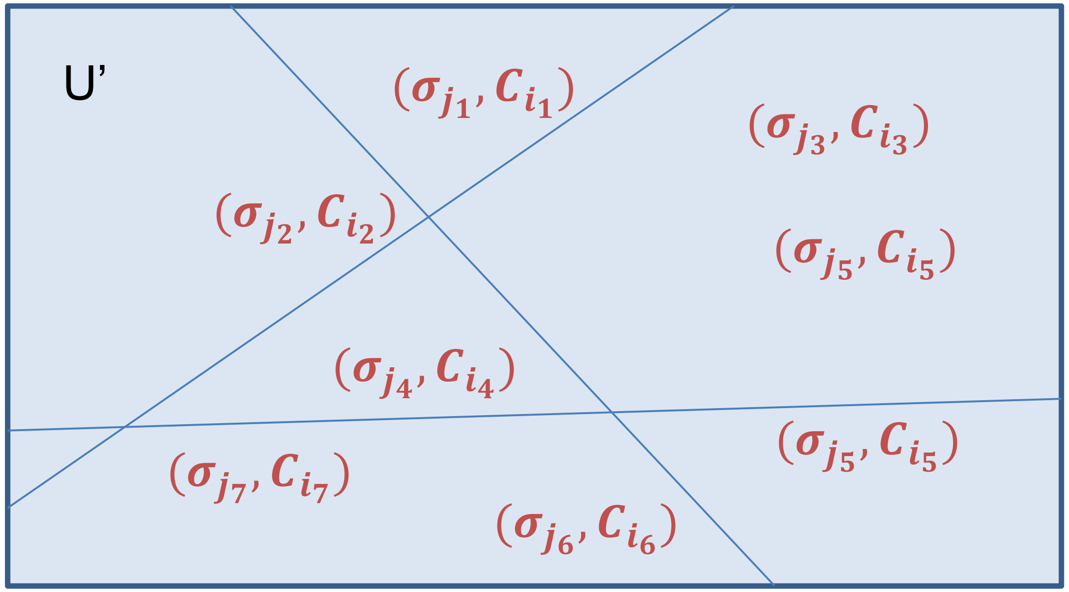
Symbolic Hashing: Counting over U'

1) Add constraints : Q_1 , Q_2

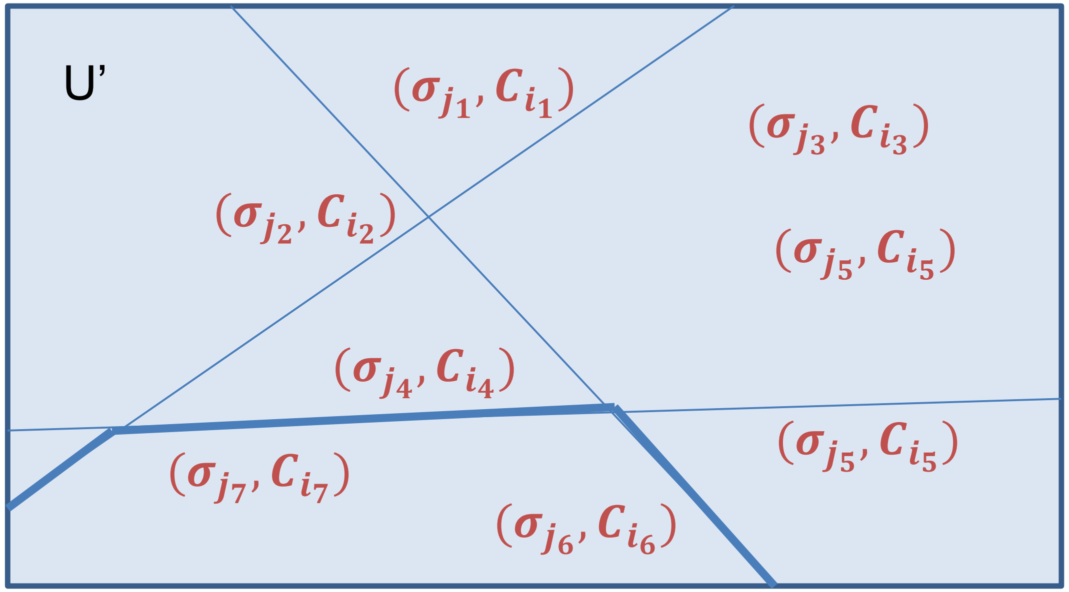


Symbolic Hashing: Counting over U'

1) Add constraints : Q_1 , Q_2 , … , Q_L



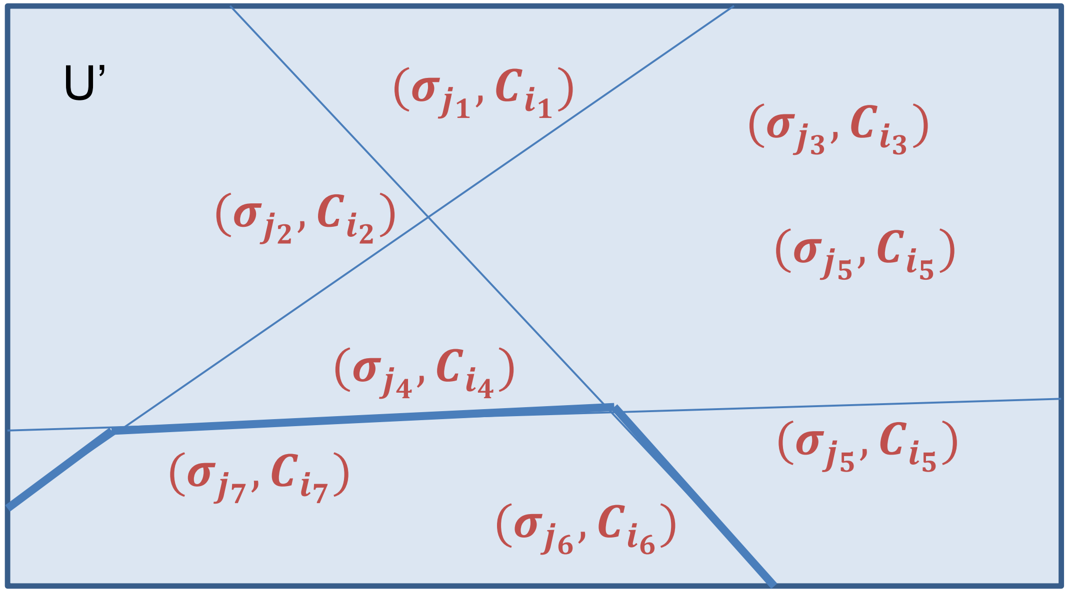
Symbolic Hashing: Counting over U'



1) Add constraints : Q_1 , Q_2 , … , Q_L

2) Pick a cell at random

Symbolic Hashing: Counting over U'



1) Add constraints : Q_1 , Q_2 , … , Q_L

2) Pick a cell at random

3) Estimate $|S_\varphi| \approx |S_{\varphi'}|$ x (# cells)
   - Details in paper

Symbolic Hashing: Significance

- **Theorem:** The estimate $|S_{\varphi'}| \times 2^L$ in U' provides the required tolerance ϵ and confidence δ
- [Chakraborty et al. '13, '16] imposed tight coupling between input formula and hash function
  - Removed this restriction
  - U' is never explicitly constructed
- Adapted ideas from Monte Carlo to Hashing

Stochastic Cell Counting

- Idea: Use Monte Carlo within cell!
  - Sample assignments uniformly from cell
  - No need to calculate $|S_{\varphi'}|$ exactly
  - Probabilistic estimate $Y \approx |S_{\varphi'}|$ adapting [Karp et al., '89]

- **Theorem:** The estimate $Y \times 2^L$ in U' provides the required tolerance ϵ and confidence δ

Complexity

- **DNF-ApproxMC**
  - Row Echelon Hash Functions
  - Symbolic Hashing
  - Stochastic Cell-Counting

- **Theorem:** DNF-ApproxMC runs in time $\tilde{O}(m \cdot n \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$

Summary

- **Problem:** Approximate DNF-Counting
- **Previous work**: Poor time complexity
- **Our contributions**: Improvements to the hashing framework
  - Row Echelon Hash Family, Symbolic Hashing, Stochastic Cell Counting
- **Result**: New complexity $\tilde{O}(m \cdot n \cdot log(\frac{1}{\delta}) \cdot \frac{1}{\varepsilon^2})$
- **Significance:** *General technique of hashing is as powerful as specialized technique of Monte Carlo for DNF-Counting!*

Future Work

- Extend techniques to Weighted DNF-Counting

- Utilize techniques to improve CNF-Counting
  - Techniques do not depend on encoding of formula
  - Sparsity of Row Echelon Hash functions

Main Results

- **Theorem 1:** Row-Echelon Hash Family is 2-Universal
- **Theorem 2:** The estimate obtained from DNF-ApproxMC provides the required tolerance ϵ and confidence δ
- **Theorem 3:** DNF-ApproxMC runs in time $\widetilde{\boldsymbol{O}}(\boldsymbol{m} \cdot \boldsymbol{n} \cdot \boldsymbol{log}(\frac{\mathbf{1}}{\boldsymbol{\delta}}) \cdot \frac{\mathbf{1}}{\boldsymbol{\varepsilon}^{\mathbf{2}}})$

Thank you

- Questions?