Model Counting meets Distinct Elements

Kuldeep S. Meel

School of Computing

National University of Singapore

Joint work with Arnab Bhattacharyya, A. Pavan, and N.V. Vinodchandran

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula φ over $X_1, X_2, \cdots X_n$
- $\mathrm{Sol}(\varphi) = \{$ satisfying assignments (aka models) of φ $\}$

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula φ over $X_1, X_2, \cdots X_n$
- $\mathrm{Sol}(\varphi) = \{$ satisfying assignments (aka models) of φ $\}$

- Model Counting: Determine $|\mathrm{Sol}(\varphi)|$

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula φ over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(\varphi) = \{$ satisfying assignments (aka models) of φ $\}$
- Model Counting: Determine $|\mathsf{Sol}(\varphi)|$
- Example $\varphi := (X_1 \vee X_2)$

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula φ over $X_1, X_2, \cdots X_n$
- $\mathrm{Sol}(\varphi) = \{$ satisfying assignments (aka models) of φ $\}$

- Model Counting: Determine $|\mathrm{Sol}(\varphi)|$

- Example $\varphi := (X_1 \vee X_2)$
- $\mathrm{Sol}(\varphi) = \{(0, 1), (1, 0), (1, 1)\}$

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula φ over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(\varphi) = \{$ satisfying assignments (aka models) of φ $\}$

- Model Counting: Determine $|\mathsf{Sol}(\varphi)|$

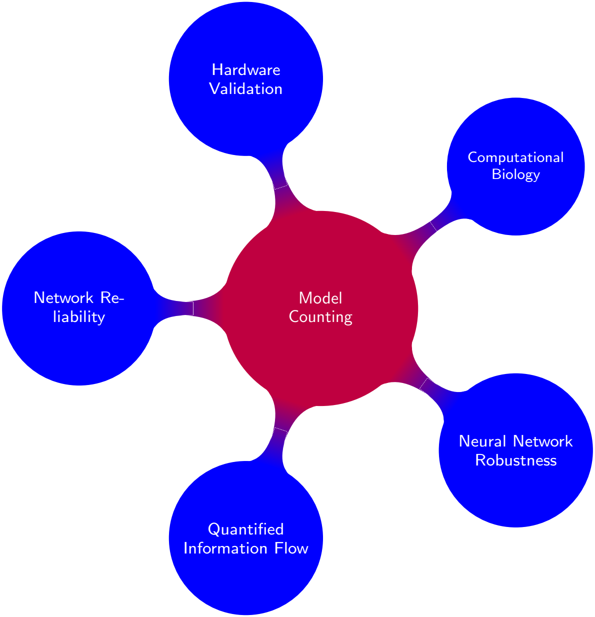
- Example $\varphi := (X_1 \vee X_2)$
- $\mathsf{Sol}(\varphi) = \{(0,1), (1,0), (1,1)\}$
- $|\mathsf{Sol}(\varphi)| = 3$

Model Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula φ over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(\varphi) = \{$ satisfying assignments (aka models) of φ $\}$
- Model Counting: Determine $|\mathsf{Sol}(\varphi)|$
- Example $\varphi := (X_1 \vee X_2)$
- $\mathsf{Sol}(\varphi) = \{(0,1), (1,0), (1,1)\}$
- $|\mathsf{Sol}(\varphi)| = 3$

Problem Compute (ε, δ) approximation of $|\mathsf{Sol}(\varphi)|$
Concern Number of NP Queries

Applications across Computer Science



Distinct Elements

- Given a stream $\mathbf{a} = a_1, a_2, \ldots a_m$ where $a_i \in \{0,1\}^n$
- $\text{DE}(\mathbf{a}) = |\cup_i a_i|$
  - Also known as F_0 estimation

Distinct Elements

- Given a stream $\mathbf{a} = a_1, a_2, \ldots a_m$ where $a_i \in \{0,1\}^n$
- $\mathrm{DE}(\mathbf{a}) = |\cup_i a_i|$
  - Also known as F_0 estimation

- Example $\mathbf{a} = 1, 2, 1, 1, 2, 1, 3, 5, 1, 2, 1, 3$
- $\mathrm{F}_0(\mathbf{a}) = |\cup_i a_i| = |\{1, 2, 3, 5\}| = 4$

Distinct Elements

- Given a stream $\mathbf{a} = a_1, a_2, \ldots a_m$ where $a_i \in \{0,1\}^n$
- $\mathrm{DE}(\mathbf{a}) = |\cup_i a_i|$
  - Also known as F_0 estimation
- Example $\mathbf{a} = 1, 2, 1, 1, 2, 1, 3, 5, 1, 2, 1, 3$
- $\mathrm{F}_0(\mathbf{a}) = |\cup_i a_i| = |\{1, 2, 3, 5\}| = 4$
- Fundamental problem in databases with a long history of work

Distinct Elements

- Given a stream $\mathbf{a} = a_1, a_2, \ldots a_m$ where $a_i \in \{0,1\}^n$
- $\mathrm{DE}(\mathbf{a}) = |\cup_i a_i|$
  - Also known as F_0 estimation

- Example $\mathbf{a} = 1, 2, 1, 1, 2, 1, 3, 5, 1, 2, 1, 3$
- $\mathrm{F}_0(\mathbf{a}) = |\cup_i a_i| = |\{1, 2, 3, 5\}| = 4$

- Fundamental problem in databases with a long history of work

  Problem Compute (ε, δ) approximation of F_0

  Concern Space Complexity

Hashing-Based Techniques

Model Counting (S83,GSS06,GHSS07,CMV13b,EGSS13b,CMV14,CDR15,CMV16,ZCSE16,AD16 KM18,ATD18,SM19,ABM20,SGM20)

Distinct Elements (FM85,AMS99,GT01,BKS02,BJKST02, CM03,CLKB04,PT07, TW12,SP09)

2-wise independent Hashing

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

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

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

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: $\frac{n}{2}$

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

$$X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} = 0 \qquad (Q_1)$$

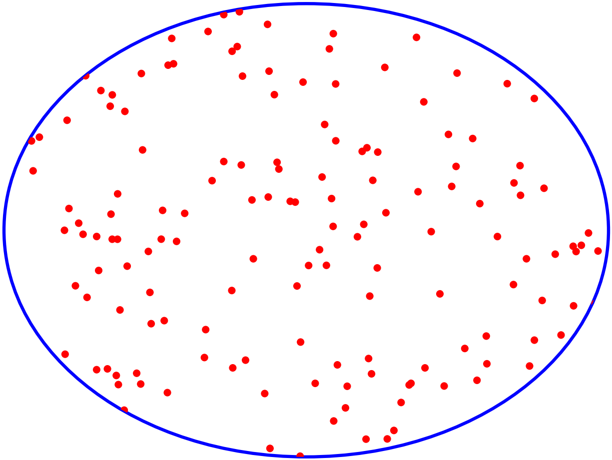
$$X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} = 1 \qquad (Q_2)$$

$$\cdots \qquad (\cdots)$$

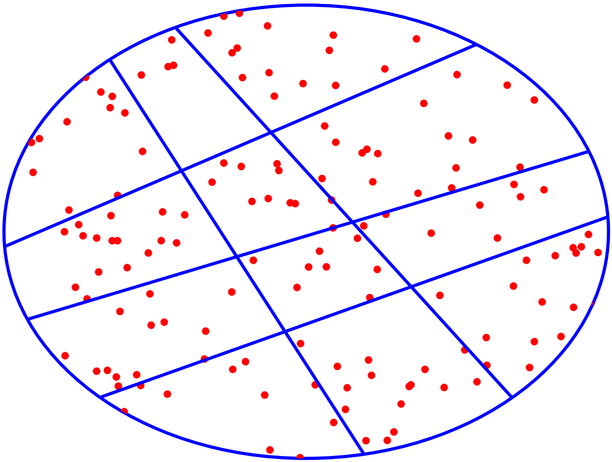
$$X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} = 1 \qquad (Q_m)$$

- Therefore, $h(X) = \alpha$ can be represented as $AX = b$

As Simple as Counting Dots

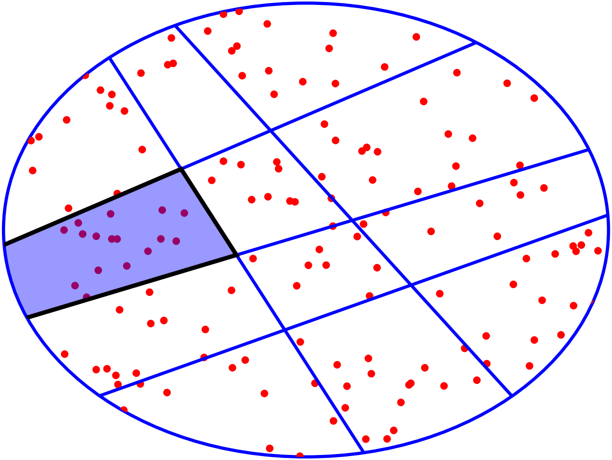


As Simple as Counting Dots



As Simple as Counting Dots

Pick a random cell



Estimate = Number of models in a cell × Number of cells

Challenges

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

Challenges

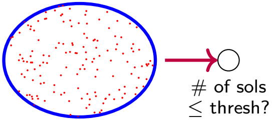
Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

Challenge 2 How many cells?

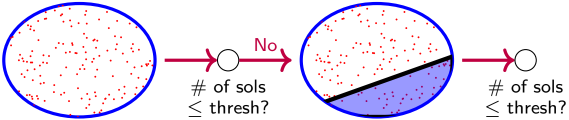
Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?
2-wise independent hash functions

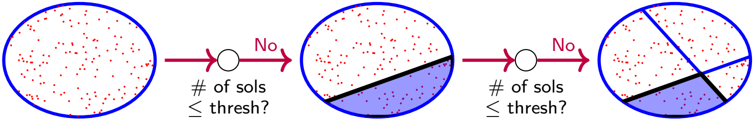
ApproxMC



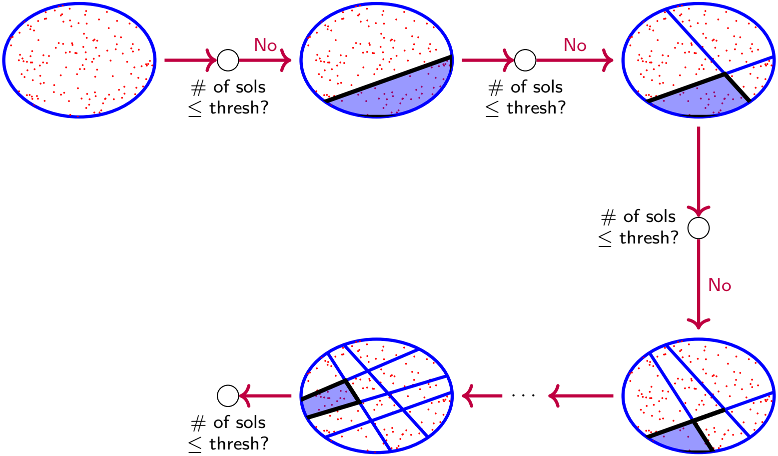
ApproxMC



ApproxMC

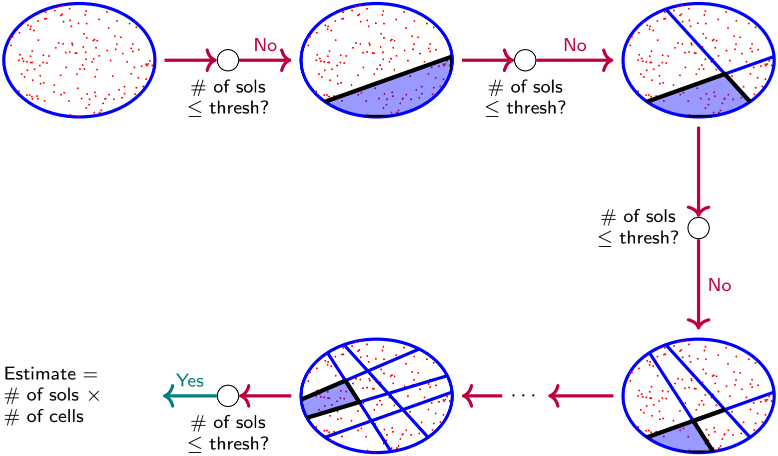


ApproxMC



No
# of sols
≤ thresh?
No
# of sols
≤ thresh?
# of sols
≤ thresh?
No
...
# of sols
≤ thresh?

ApproxMC


No
# of sols
≤ thresh?
No
# of sols
≤ thresh?
# of sols
≤ thresh?
No
Estimate =
# of sols ×
# of cells
Yes
# of sols
≤ thresh?
· · ·

Distinct Elements

Choose $h : \{0,1\}^n \mapsto \{0,1\}^n$
minhash $\leftarrow 2^n$;
for $a_i \in$ **a do**
    if $h(a_i) <$ minhash **then**
        minhash $= h(a_i)$
    end if
end for
return $\frac{2^n}{\text{minhash}}$

Is there more than meets the eyes?

- From Distinct Elements to Counting
- From Counting to Distinct Elements

Hashing-based Distinct Elements

$h \leftarrow$ ChooseHashFunctions
$\mathcal{S} \leftarrow \{\}$
for $a_i \in$ **a do**
    ProcessUpdate(S, h, a_i)
end for
Est $\leftarrow$ ComputeEst($\mathcal{S}$)
Return Est

Hashing-based Distinct Elements

```
h ← ChooseHashFunctions
S ← {}
for a_i ∈ a do
    ProcessUpdate(S, h, a_i)
end for
Est ← ComputeEst(S)
Return Est
```

Different Algorithms based on ProcessUpdate

- Minimum: Keep track of minimum $h(a_i)$
- Bucketing

Hashing-based Distinct Elements

$h \leftarrow$ ChooseHashFunctions
$\mathcal{S} \leftarrow \{\}$
for $a_i \in$ **a do**
    ProcessUpdate(S, h, a_i)
end for
Est $\leftarrow$ ComputeEst$(\mathcal{S})$
Return Est

Different Algorithms based on ProcessUpdate

- Minimum: Keep track of minimum $h(a_i)$
- Bucketing
- ...

From Distinct Elements to Counting: A Two Step Recipe

$\mathbf{a}_u$: set of all distinct elements of the stream $\mathbf{a}$.

Key Idea The formula φ can viewed as symbolic representation of some set $\mathbf{a}_u$ such that $\mathsf{Sol}(\varphi) = \mathbf{a}_u$.

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, h, and the set $\mathbf{a}_u$ at the end of stream.

Step 2 Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds. And now, we can estimate $|\mathsf{Sol}(\varphi)|$ from $\mathcal{S}$.

Min-based Estimation

Choose $h : \{0,1\}^n \mapsto \{0,1\}^n$
minhash $\leftarrow 2^n$;
for $a_i \in$ **a do**
    if minhash $< h(a_i)$ **then**
        minhash $= h(a_i)$
    end if
end for
return $\frac{2^n}{\text{minhash}}$

Application I: Min-based Counting Algorithm

Step1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, h, and the set $\mathbf{a}_u$ at the end of stream.

Application I: Min-based Counting Algorithm

Step1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, h, and the set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} := \min_{y \in a_u} h(y)$

Application I: Min-based Counting Algorithm

Step1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, h, and the set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} := \min_{y \in a_u} h(y)$
$\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ $\mathcal{S} := \min_{y \in \mathsf{Sol}(\varphi)} h(y)$

Application I: Min-based Counting Algorithm

Step1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, h, and the set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} := \min_{y \in a_u} h(y)$
$\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ $\mathcal{S} := \min_{y \in \mathsf{Sol}(\varphi)} h(y)$

Step2 Given a formula φ and set of hash functions H, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds. And now, we can estimate $|\mathsf{Sol}(\varphi)|$ from $\mathcal{S}$.

- Use polynomially many calls to NP Oracle to determine $\mathcal{S}$

Bucketing-based Streaming Algorithm

```
Choose $h : \{0,1\}^n \mapsto \{0,1\}^n$
$\ell \leftarrow 0; \mathcal{B} \leftarrow \emptyset$
for $a_i \in$ a do
    if $h(a_i) \mod 2^\ell = 0^\ell$ then
        $\mathcal{B}$.Append($a_i$)
        if $|\mathcal{B}| \geq$ thresh then
            $\ell$++
            Filter($\mathcal{B}, h, \ell$)
        end if
    end if
end for
return $|\mathcal{B}| \times 2^\ell$
```

Bucketing-based Streaming Algorithm

Choose $h : \{0,1\}^n \mapsto \{0,1\}^n$
$\ell \leftarrow 0; \mathcal{B} \leftarrow \emptyset$
for $a_i \in \mathbf{a}$ **do**
    if $h(a_i) \mod 2^\ell = 0^\ell$ **then**
        $\mathcal{B}$.Append(a_i)   Elements that satisfy XOR
        if $|\mathcal{B}| \geq$ thresh **then**
            ℓ++
            Filter($\mathcal{B}, h, \ell$)   Add another XOR
        end if
    end if
end for
return $|\mathcal{B}| \times 2^\ell$

Application II: Bucketing-based Counting Algorithm

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, hash function h and set $\mathbf{a}_u$ at the end of stream.

Application II: Bucketing-based Counting Algorithm

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, hash function h and set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathbf{a}_u \cap h^{-1}(0^\ell)$ and $|\{\mathbf{a}_u \cap h^{-1}(0^{\ell-1})\}| >$ thresh and $|\mathcal{B}| \leq$ thresh.

Application II: Bucketing-based Counting Algorithm

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, hash function h and set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathbf{a}_u \cap h^{-1}(0^\ell)$ and $|\{\mathbf{a}_u \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$.

$\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathsf{Sol}(\varphi) \cap h^{-1}(0^\ell)$ and $|\{\mathsf{Sol}(\varphi) \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$

Application II: Bucketing-based Counting Algorithm

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, hash function h and set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathbf{a}_u \cap h^{-1}(0^\ell)$ and $|\{\mathbf{a}_u \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$.

$\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathsf{Sol}(\varphi) \cap h^{-1}(0^\ell)$ and $|\{\mathsf{Sol}(\varphi) \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$

Step 2 Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds. And now, we can estimate $|\mathsf{Sol}(\varphi)|$ from $\mathcal{S}$.

Application II: Bucketing-based Counting Algorithm

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, hash function h and set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathbf{a}_u \cap h^{-1}(0^\ell)$ and $|\{\mathbf{a}_u \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$.

$\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathsf{Sol}(\varphi) \cap h^{-1}(0^\ell)$ and $|\{\mathsf{Sol}(\varphi) \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$

Step 2 Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds. And now, we can estimate $|\mathsf{Sol}(\varphi)|$ from $\mathcal{S}$.

- Use polynomially many calls to NP Oracle to determine $\mathcal{S}$

Application II: Bucketing-based Counting Algorithm

Step 1 Capture the relationship $\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$ between the sketch $\mathcal{S}$, hash function h and set $\mathbf{a}_u$ at the end of stream.

$\mathcal{P}(\mathcal{S}, h, \mathbf{a}_u)$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathbf{a}_u \cap h^{-1}(0^\ell)$ and $|\{\mathbf{a}_u \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$.

$\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$: $\mathcal{S} = (\ell, \mathcal{B})$ such that $\mathcal{B} = \mathsf{Sol}(\varphi) \cap h^{-1}(0^\ell)$ and $|\{\mathsf{Sol}(\varphi) \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|\mathcal{B}| \leq \text{thresh}$

Step 2 Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds. And now, we can estimate $|\mathsf{Sol}(\varphi)|$ from $\mathcal{S}$.

- Use polynomially many calls to NP Oracle to determine $\mathcal{S}$

This is ApproxMC!

From Distinct Elements to Counting: Implications

Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds.

Theorem (FPRAS)

If construction of sketch $\mathcal{S}$ is in PTIME for a class of formulas, then there is FPRAS for the corresponding class. E.g.: DNF, Union of XORs

From Distinct Elements to Counting: Implications

Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds.

Theorem (FPRAS)

If construction of sketch $\mathcal{S}$ is in PTIME for a class of formulas, then there is FPRAS for the corresponding class. E.g.: DNF, Union of XORs

Theorem (Space and Query)

$p(n)$ space algorithms in streaming imply $(p(n))^2$ NP query complexity algorithms for model counting

From Distinct Elements to Counting: Implications

Given a formula φ and hash function h, design an algorithm to construct sketch $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, h, \mathsf{Sol}(\varphi))$ holds.

Theorem (FPRAS)

If construction of sketch $\mathcal{S}$ is in PTIME for a class of formulas, then there is FPRAS for the corresponding class. E.g.: DNF, Union of XORs

Theorem (Space and Query)

p(n) space algorithms in streaming imply $(p(n))^2$ NP query complexity algorithms for model counting

Theorem (Lower Bounds)

Lower bounds for Distributed Streaming translate to lower bounds for Distributed DNF counting

Is there more to it than meets the eyes?

- From Distinct Elements to Counting
- From Counting to Distinct Elements

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)
- A stream can be viewed as a DNF
  - $a = a_1, a_2, a_3, \ldots a_m$

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)
- A stream can be viewed as a DNF
  - $a = a_1, a_2, a_3, \ldots a_m$
  - $|\cup_i a_i| = |\mathrm{Sol}(a_1 \vee a_2 \vee a_3 \vee a_m)|$
  - a_i is represented by conjunction of n literals $X_1, X_2, \ldots X_n$.

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)
- A stream can be viewed as a DNF
  - $a = a_1, a_2, a_3, \ldots a_m$
  - $|\cup_i a_i| = |\mathrm{Sol}(a_1 \vee a_2 \vee a_3 \vee a_m)|$
  - a_i is represented by conjunction of n literals $X_1, X_2, \ldots X_n$.
- So hashing-based FPRAS for DNF $\implies$ F_0 estimation

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)
- A stream can be viewed as a DNF
  - $a = a_1, a_2, a_3, \ldots a_m$
  - $|\cup_i a_i| = |\mathsf{Sol}(a_1 \vee a_2 \vee a_3 \vee a_m)|$
  - a_i is represented by conjunction of n literals $X_1, X_2, \ldots X_n$.
- So hashing-based FPRAS for DNF $\implies$ F_0 estimation
- A general scheme for structured sets

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)
- A stream can be viewed as a DNF
  - $a = a_1, a_2, a_3, \ldots a_m$
  - $|\cup_i a_i| = |\mathrm{Sol}(a_1 \vee a_2 \vee a_3 \vee a_m)|$
  - a_i is represented by conjunction of n literals $X_1, X_2, \ldots X_n$.
- So hashing-based FPRAS for DNF $\implies$ F_0 estimation
- A general scheme for structured sets
- Encompasses models such as ranges, affine spaces

From Counting to Distinct Elements

- ApproxMC is FPRAS for DNF formulas (CMV16,MSV17,MSV18)
- A stream can be viewed as a DNF
  - $a = a_1, a_2, a_3, \ldots a_m$
  - $|\cup_i a_i| = |\mathrm{Sol}(a_1 \vee a_2 \vee a_3 \vee a_m)|$
  - a_i is represented by conjunction of n literals $X_1, X_2, \ldots X_n$.
- So hashing-based FPRAS for DNF $\implies$ F_0 estimation
- A general scheme for structured sets
- Encompasses models such as ranges, affine spaces
- Application: Distinct Elements over Range
  - Every item $[a_i, b_i]$ can be represented using a DNF formula.
  - So just apply FPRAS for DNF

Conclusion

Summary

- From Distinct Elements to Counting
- From Counting to Distinct Elements

Conclusion

Summary

- From Distinct Elements to Counting
- From Counting to Distinct Elements

Future Directions

- Practical scalability of newly devised counting techniques
- Lifting Sparse Hashing techniques to streaming
- What is the analogue for higher moments (F_k)