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 $\varphi$ over $X_1, X_2, \cdots X_n$
- $\text{Sol}(\varphi) = \{ \text{satisfying assignments (aka models) of } \varphi \}$
Model Counting

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

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

Example $\varphi := (X_1 \lor X_2)$

- $\text{Sol}(\varphi) = \{(0, 1), (1, 0), (1, 1)\}$
- $|\text{Sol}(\varphi)| = 3$
Model Counting

• Given
  • Boolean variables $X_1, X_2, \cdots X_n$
  • Formula $\varphi$ over $X_1, X_2, \cdots X_n$

• $\text{Sol}(\varphi) = \{\text{satisfying assignments (aka models) of } \varphi \}$

• **Model Counting**: Determine $|\text{Sol}(\varphi)|$

• Example $\varphi := (X_1 \lor X_2)$
Model Counting

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

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

- **Example** $\varphi := (X_1 \lor X_2)$
  - $\text{Sol}(\varphi) = \{(0, 1), (1, 0), (1, 1)\}$
Model Counting

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

• Model Counting: Determine $|\text{Sol}(\varphi)|$

• Example $\varphi := (X_1 \lor X_2)$
  • $\text{Sol}(\varphi) = \{(0, 1), (1, 0), (1, 1)\}$
  • $|\text{Sol}(\varphi)| = 3$
Model Counting

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

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

- **Example** $\varphi := (X_1 \lor X_2)$
  - $\text{Sol}(\varphi) = \{(0, 1), (1, 0), (1, 1)\}$
  - $|\text{Sol}(\varphi)| = 3$

**Problem** Compute $(\varepsilon, \delta)$ approximation of $|\text{Sol}(\varphi)|$

**Concern** Number of NP Queries
Applications across Computer Science

- Hardware Validation
- Computational Biology
- Network Reliability
- Neural Network Robustness
- Quantified Information Flow
Distinct Elements

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

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

- Example $a = 1, 2, 1, 1, 2, 1, 3, 5, 1, 2, 1, 3$
- $F_0(a) = | \bigcup_i a_i | = |\{1, 2, 3, 5\}| = 4$
Distinct Elements

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

- Example \( a = 1, 2, 1, 1, 2, 1, 3, 5, 1, 2, 1, 3 \)
- \( F_0(a) = | \bigcup_i a_i | = |\{1, 2, 3, 5\}| = 4 \)

- Fundamental problem in databases with a long history of work
Distinct Elements

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

- Example $a = 1, 2, 1, 1, 2, 1, 3, 5, 1, 2, 1, 3$
- $F_0(a) = |\bigcup_i a_i| = |\{1, 2, 3, 5\}| = 4$

- Fundamental problem in databases with a long history of work
  
  **Problem** Compute $(\varepsilon, \delta)$ 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 $\{0, 1\}^m$

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

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

\[
\Pr[h(y_1) = \alpha_1 \land h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)^2
\]
2-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: $X_1, X_2, \ldots, X_n$
- 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: $\frac{n}{2}$
- To choose $\alpha \in \{0, 1\}^m$, set every XOR equation to 0 or 1 randomly
  
  \[
  \begin{array}{l}
  X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} = 0 \\
  X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} = 1 \\
  \quad \cdots \\
  X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} = 1 \\
  \end{array}
  \]  
  \[ (Q_1) \quad (Q_2) \quad (\cdots) \quad (Q_m) \]
- Therefore, $h(X) = \alpha$ can be represented as $AX = b$
As Simple as Counting Dots

Pick a random cell

Estimate = Number of models in a cell × Number of cells
As Simple as Counting Dots

Pick a random cell

Estimate = Number of models in a cell \times Number of cells
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

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

Estimate =  
# of sols ×  
# of cells

# of sols ≤ thresh?

No

No

No

Yes

...
1: Choose $h : \{0, 1\}^n \mapsto \{0, 1\}^n$
2: $\text{minhash} \leftarrow 2^n$;
3: for $a_i \in a$ do
4:  if $h(a_i) < \text{minhash}$ then
5:   $\text{minhash} = h(a_i)$
6:  end if
7: end for
8: 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

1: $h \leftarrow \text{ChooseHashFunctions}$
2: $S \leftarrow \{\}$
3: for $a_i \in a$ do
4: \hspace{1em} \text{ProcessUpdate}(S, h, a_i)$
5: end for
6: Est $\leftarrow \text{ComputeEst}(S)$
7: Return Est
Hashing-based Distinct Elements

1. $h \leftarrow \text{ChooseHashFunctions}$
2. $S \leftarrow \{\}$
3. for $a_i \in a$ do
4. \hspace{1em} ProcessUpdate$(S, h, a_i)$
5. end for
6. Est $\leftarrow \text{ComputeEst}(S)$
7. Return Est

Different Algorithms based on ProcessUpdate

- Minimum: Keep track of minimum $h(a_i)$
- Bucketing
Hashing-based Distinct Elements

1: \( h \leftarrow \text{ChooseHashFunctions} \)
2: \( S \leftarrow \{\} \)
3: \textbf{for} \( a_i \in a \) \textbf{do}
4: \quad \text{ProcessUpdate}(S, h, a_i)
5: \textbf{end for}
6: \( \text{Est} \leftarrow \text{ComputeEst}(S) \)
7: \text{Return Est}

Different Algorithms based on ProcessUpdate

- Minimum: Keep track of minimum \( h(a_i) \)
- Bucketing
- ...
\(a_u\): set of all distinct elements of the stream \(a\).

**Key Idea** The formula \(\varphi\) can viewed as symbolic representation of some set \(a_u\) such that \(\text{Sol}(\varphi) = a_u\).

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

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

1: Choose $h : \{0, 1\}^n \rightarrow \{0, 1\}^n$
2: minhash ← $2^n$;
3: for $a_i \in a$ do
4: \hspace{1em} if minhash $< h(a_i)$ then
5: \hspace{2em} minhash = $h(a_i)$
6: \hspace{1em} end if
7: end for
8: return $\frac{2^n}{\text{minhash}}$
Application I: Min-based Counting Algorithm

**Step1** Capture the relationship $\mathcal{P}(S, h, a_u)$ between the sketch $S$, $h$, and the set $a_u$ at the end of stream.
Application I: Min-based Counting Algorithm

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

$$\mathcal{P}(S, h, a_u) : S := \min_{y \in a_u} h(y)$$
Application I: Min-based Counting Algorithm

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

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

$\mathcal{P}(S, h, \text{Sol}(\varphi)) : S := \min_{y \in \text{Sol}(\varphi)} h(y)$
Application I: Min-based Counting Algorithm

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

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

$\mathcal{P}(S, h, \text{Sol}(\varphi)) : S := \min_{y \in \text{Sol}(\varphi)} h(y)$

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

- Use polynomially many calls to NP Oracle to determine $S$
Bucketing-based Streaming Algorithm

1: Choose $h : \{0, 1\}^n \mapsto \{0, 1\}^n$
2: $\ell \leftarrow 0; B \leftarrow \emptyset$
3: for $a_i \in a$ do
4:   if $h(a_i) \mod 2^\ell = 0^\ell$ then
5:     $B$.Append($a_i$)
6:   if $|B| \geq \text{thresh}$ then
7:     $\ell++$
8:     Filter($B, h, \ell$)
9:   end if
10: end if
11: end for
12: return $|B| \times 2^\ell$
Bucketing-based Streaming Algorithm

1: Choose $h : \{0, 1\}^n \mapsto \{0, 1\}^n$
2: $\ell \leftarrow 0; \mathcal{B} \leftarrow \emptyset$
3: for $a_i \in a$ do
4: \hspace{1em} if $h(a_i) \mod 2^\ell = 0^\ell$ then
5: \hspace{2em} $\mathcal{B}.\text{Append}(a_i)$
6: \hspace{2em} if $|\mathcal{B}| \geq \text{thresh}$ then
7: \hspace{3em} $\ell++$
8: \hspace{3em} Filter($\mathcal{B}, h, \ell$)
9: \hspace{2em} end if
10: \hspace{1em} end if
11: end for
12: return $|\mathcal{B}| \times 2^\ell$

Elements that satisfy XOR

Add another XOR
Application II: Bucketing-based Counting Algorithm

Step 1  Capture the relationship $P(S, h, a_u)$ between the sketch $S$, hash function $h$ and set $a_u$ at the end of stream.
Application II: Bucketing-based Counting Algorithm

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

$$\mathcal{P}(S, h, a_u) : S = (\ell, B) \text{ such that } B = a_u \cap h^{-1}(0^\ell) \text{ and } \{|a_u \cap h^{-1}(0^{\ell-1})| > \text{thresh} \text{ and } |B| \leq \text{thresh}.$$
Step 1  Capture the relationship $\mathcal{P}(S, h, a_u)$ between the sketch $S$, hash function $h$ and set $a_u$ at the end of stream.

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

$\mathcal{P}(S, h, \text{Sol}(\varphi)) : S = (\ell, B)$ such that $B = \text{Sol}(\varphi) \cap h^{-1}(0^\ell)$ and $|\{\text{Sol}(\varphi) \cap h^{-1}(0^{\ell-1})\}| > \text{thresh}$ and $|B| \leq \text{thresh}$.
Application II: Bucketing-based Counting Algorithm

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

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

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

**Step 2** Given a formula $\varphi$ and hash function $h$, design an algorithm to construct sketch $S$ such that $\mathcal{P}(S, h, \text{Sol}(\varphi))$ holds. And now, we can estimate $|\text{Sol}(\varphi)|$ from $S$. 
Application II: Bucketing-based Counting Algorithm

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

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

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

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

- Use polynomially many calls to NP Oracle to determine $S$.
Application II: Bucketing-based Counting Algorithm

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

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

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

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

- Use polynomially many calls to NP Oracle to determine \( S \)

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

**Theorem (FPRAS)**

*If construction of sketch $S$ is in PTIME for a class of formulas, then there is FPRAS for the corresponding class. E.g.: DNF, Union of XORs*
Given a formula $\varphi$ and hash function $h$, design an algorithm to construct sketch $S$ such that $\mathcal{P}(S, h, \text{Sol}(\varphi))$ holds.

**Theorem (FPRAS)**

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

**Theorem (FPRAS)**

*If construction of sketch \( S \) 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)

A stream can be viewed as a DNF
\[ a_1, a_2, a_3, \ldots, a_m \]
\[ | \bigcup_i a_i | = | \text{Sol}(a_1 \lor a_2 \lor a_3 \lor \ldots \lor a_m) | \]

So hashing-based FPRAS for DNF = \( F_{\text{estimation}} \)

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
• 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
  • \( a = a_1, a_2, a_3, \ldots a_m \)
  • \( | \bigcup_i a_i | = | \text{Sol}(a_1 \lor a_2 \lor a_3 \lor a_m) | \)
  • \( a_i \) is represented by conjunction of \( n \) literals \( X_1, X_2, \ldots X_n \).
ApproxMC is FPRAS for DNF formulas \((CMV16,MSV17,MSV18)\)

A stream can be viewed as a DNF

- \(a = a_1, a_2, a_3, \ldots a_m\)
- \(| \bigcup_i a_i | = | \text{Sol}(a_1 \lor a_2 \lor a_3 \lor a_m) |\)
- \(a_i\) is represented by conjunction of \(n\) literals \(X_1, X_2, \ldots X_n\).

So hashing-based FPRAS for DNF \(\implies F_0\) estimation
• ApproxMC is FPRAS for DNF formulas \( \text{(CMV16,MSV17,MSV18)} \)

• A stream can be viewed as a DNF
  • \( a = a_1, a_2, a_3, \ldots a_m \)
  • \( | \bigcup_i a_i | = | \text{Sol}(a_1 \lor a_2 \lor a_3 \lor a_m) | \)
  • \( a_i \) 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
• ApproxMC is FPRAS for DNF formulas \((CMV16, MSV17, MSV18)\)

• A stream can be viewed as a DNF
  • \(a = a_1, a_2, a_3, \ldots a_m\)
  • \(|\bigcup_i a_i| = |\text{Sol}(a_1 \lor a_2 \lor a_3 \lor a_m)|\)
  • \(a_i\) 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
  • $a = a_1, a_2, a_3, \ldots a_m$
  • $| \cup_i a_i | = | \text{Sol}(a_1 \lor a_2 \lor a_3 \lor a_m) |$
  • $a_i$ is represented by conjunction of $n$ literals $X_1, X_2, \ldots X_n$.

• So hashing-based FPRAS for DNF $\Rightarrow$ $F_0$ 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 ($F_k$)