Distinct Elements in Streams: An Algorithm for the (Text) Book

Kuldeep S. Meel
University of Toronto

Joint work with Sourav Chakraborty ² and N. V. Vinodchandran ³

² Indian Statistical Institute, Kolkata
³ University of Nebraska, Lincoln

with Addendum from: Don Knuth

Corresponding publications: PODS-21, PODS-22, ESA-22
Problem Setting

**Input**  A data stream $D = < a_1, a_2, \ldots a_m >$ where $a_i \in [n]$

**Output**  Compute the number of Distinct elements in $D$. Formally, $F_0 = |\bigcup_i \{a_i\}|$
Problem Setting

**Input** A data stream \( D = < a_1, a_2, \ldots a_m > \) where \( a_i \in [n] \)

**Output** Compute the number of Distinct elements in \( D \). Formally, \( F_0 = |\cup_i \{a_i\}| \)

Example: \( D = < 1, 1, 2, 1, 4, 1, 2, 1 > \) \( F_0 = 3 \)
**Problem Setting**

**Input**  A data stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$ where $a_i \in [n]$

**Output**  Compute the number of Distinct elements in $\mathcal{D}$. Formally, $F_0 = |\bigcup_i \{a_i\}|$

Example: $\mathcal{D} = \langle 1, 1, 2, 1, 4, 1, 2, 1 \rangle$  \hspace{1cm} $F_0 = 3$

- Our focus: $(\varepsilon, \delta)$-approximation

\[ Pr[(1 - \varepsilon)F_0 \leq \text{Est} \leq (1 + \varepsilon)F_0] \geq 1 - \delta \]
Problem Setting

**Input**  A data stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$ where $a_i \in [n]$

**Output**  Compute the number of Distinct elements in $\mathcal{D}$. Formally, $F_0 = |\bigcup_i \{a_i\}|$

**Example:** $\mathcal{D} = \langle 1, 1, 2, 1, 4, 1, 2, 1 \rangle$  $F_0 = 3$

- Our focus: $(\varepsilon, \delta)$-approximation
  \[ Pr \left[ (1 - \varepsilon)F_0 \leq \text{Est} \leq (1 + \varepsilon)F_0 \right] \geq 1 - \delta \]

**Naive Solution**  Maintain a large hash table: worst-case space complexity of $O(n)$

**Objective**  Optimize space and update time complexity
  - **Update Time:** Time to process each element of the stream
Rich History of work

- Flajolet and Martin (1985)

**Crowning Jewel** Optimal (time and ) space complexity:  $O \left( \log n + \frac{1}{\varepsilon^2} \cdot \log 1/\delta \right)$
Rich History of work

- Flajolet and Martin (1985)

Crowning Jewel Optimal (time and ) space complexity: \( O \left( \log n + \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta} \right) \)

Limitations Practically efficient algorithms are beyond graduate classroom. Theoretically efficient algorithms can be taught in graduate classroom but don’t work in practice.
Rich History of work

- Flajolet and Martin (1985)

**Crowning Jewel** Optimal (time and) space complexity: \( O\left(\log n + \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right) \)

**Limitations** Practically efficient algorithms are beyond graduate classroom
Theoretically efficient algorithms can be taught in graduate classroom but don’t work in practice

**Theorem (Primary Contribution)**

A **simple algorithm with time and space complexity** of \( O\left(\frac{1}{\varepsilon^2} \cdot \log n \cdot (\log m + \log \frac{1}{\delta})\right) \).

**Remark:** The description and algorithm requires only basic data structures and knowledge of elementary probability theory (Chernoff Bound), and can be easily taught in an undergraduate course, and the algorithm is practically efficient.

The paper is just five pages (including abstract and bibliographical remarks)
Knuth (May 23): “Ever since I saw it, a few days ago, I’ve been unable to resist trying to explain the ideas to just about everybody I meet.”
Rich History of work

- Flajolet and Martin (1985)

Crowning Jewel Optimal (time and ) space complexity: \( O\left(\log n + \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right) \)

Limitations Practically efficient algorithms are beyond graduate classroom
Theoretically efficient algorithms can be taught in graduate classroom but don’t work in practice

Theorem (Primary Contribution)

\begin{quote}
A simple algorithm with time and space complexity of \( O\left(\frac{1}{\varepsilon^2} \cdot \log n \cdot (\log m + \log \frac{1}{\delta})\right) \).
\end{quote}

Remark: The description and algorithm requires only basic data structures and knowledge of elementary probability theory (Chernoff Bound), and can be easily taught in an undergraduate course, and the algorithm is practically efficient.

The paper is just five pages (including abstract and bibliographical remarks)
Knuth (May 23): “Ever since I saw it, a few days ago, I’ve been unable to resist trying to explain the ideas to just about everybody I meet.”

Core Idea If we pick every ball in a bin with probability \( p \) in our bucket and we end up \( k \) balls in the bucket, then \( \frac{k}{p} \) is a good estimate of the number of balls in the bin.
Idea 1

Sample every element of $S_i = \{a_1, a_2, \ldots, a_m\}$ identically and independently with prob. $p$

Algorithm NaiveSampler

Input Stream $D = \langle a_1, a_2, \ldots, a_m \rangle$, $p$

1: Initialize $B \leftarrow \emptyset$
2: for $i = 1$ to $m$ do
3: With probability $p$, $B \leftarrow B \cup \{a_i\}$

Example: $D = \langle 1, 1, 2, 1, 4, 1, 2, 1 \rangle$

Challenge

Elements that repeat more often are more likely to be sampled

Solution

Throw it Away is All You Need

Algorithm Sampler

Input Stream $D = \langle a_1, a_2, \ldots, a_m \rangle$, $p$

1: Initialize $B \leftarrow \emptyset$
2: for $i = 1$ to $m$ do
3: $B \leftarrow B \setminus \{a_i\}$
4: With probability $p$, $B \leftarrow B \cup \{a_i\}$

Observation

Whether an element $x \in B$ or not only depends on whether $x$ was picked with probability when it appeared last time
Key Ingredients - I

**Idea 1** Sample every element of $\bigcup_{i=1}^{m} \{a_i\}$ identically and independently with prob. $p$. 
Key Ingredients - I

Idea 1  Sample every element of $\bigcup_{i=1}^{m} \{a_i\}$ identically and independently with prob. $p$

Algorithm NaiveSampler

1. **Input** Stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$, $p$
2. **Initialize** $B \leftarrow \emptyset$
3. **for** $i = 1$ **to** $m$ **do**
   3.1 With probability $p$, $B \leftarrow B \cup \{a_i\}$

Example: $\mathcal{D} = \langle 1, 1, 2, 1, 4, 1, 2, 1 \rangle$
Key Ingredients - I

Idea 1  Sample every element of $\bigcup_{i=1}^{m}\{a_i\}$ identically and independently with prob. $p$

Algorithm NaiveSampler

```
Input Stream $D = (a_1, a_2, \ldots, a_m)$, p
1: Initialize $B \leftarrow \emptyset$
2: for $i = 1$ to $m$ do
3:   With probability $p$, $B \leftarrow B \cup \{a_i\}$
```

Example: $D = \langle 1, 1, 2, 1, 4, 1, 2, 1 \rangle$

Challenge  Elements that repeat more often are more likely to be sampled
Idea 1. Sample every element of $\bigcup_{i=1}^{m} \{a_i\}$ identically and independently with prob. $p$.

**Algorithm** NaiveSampler

Input Stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$, $p$

1. Initialize $B \leftarrow \emptyset$;
2. for $i = 1$ to $m$ do
3. With probability $p$, $B \leftarrow B \cup \{a_i\}$

Example: $\mathcal{D} = \langle 1, 1, 2, 1, 4, 1, 2, 1 \rangle$

**Challenge** Elements that repeat more often are more likely to be sampled

**Solution** Throw it Away is All You Need

**Algorithm** Sampler

Input Stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$, $p$

1. Initialize $B \leftarrow \emptyset$;
2. for $i = 1$ to $m$ do
3. $B \leftarrow B \setminus \{a_i\}$
4. With probability $p$, $B \leftarrow B \cup \{a_i\}$

**Observation** Whether an element $x \in B$ or not only depends on whether $x$ was picked with probability when it appeared last time.
Idea 1  Sample every element of $\bigcup_{i=1}^{m} \{a_i\}$ identically and independently with prob. $p$

Idea 2  Determine *just the right* value of $p$?

- Too large $p$, $|B|$ is too large
- Too small $p$, $\frac{|B|}{p}$ is not a good estimator
Idea 1  Sample every element of $\bigcup_{i=1}^{m} \{a_i\}$ identically and independently with prob. $p$

Idea 2  Determine just the right value of $p$?

- Too large $p$, $|B|$ is too large
- Too small $p$, $\frac{|B|}{p}$ is not a good estimator

Algorithm Adaptive Estimator

Input Stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$, $\varepsilon$, $\delta$

1: Initialize $B \leftarrow \emptyset$; thresh $\leftarrow \frac{12}{\varepsilon^2} \log\left(\frac{8m}{\delta}\right)$; $p \leftarrow 1$

2: for $i = 1$ to $m$ do

3: \hspace{1em} $B \leftarrow B \setminus \{a_i\}$

4: \hspace{1em} With probability $p$, $B \leftarrow B \cup \{a_i\}$

5: \hspace{1em} while $|B| = \text{thresh}$ do

6: \hspace{2em} Throw away each element of $B$ with probability $\frac{1}{2}$

7: \hspace{1em} $p \leftarrow \frac{p}{2}$

8: Output $\frac{|B|}{p}$
Algorithm Adaptive Estimator

1: Initialize $B \leftarrow \emptyset$; thresh $\leftarrow \frac{12}{\epsilon^2} \log \left( \frac{8m}{\delta} \right)$; $p \leftarrow 1$
2: for $i = 1$ to $m$ do
3:   $B \leftarrow B \setminus \{a_i\}$
4:   With probability $p$, $B \leftarrow B \cup \{a_i\}$
5:   while $|B| = \text{thresh}$ do
6:      Throw away each element of $B$ with probability $\frac{1}{2}$
7:      $p \leftarrow \frac{p}{2}$
8: Output $\frac{|B|}{p}$
The Power of Adaptive Estimator

**Algorithm** Adaptive Estimator

1. **Initialize** $B \leftarrow \emptyset$; $\text{thresh} \leftarrow \frac{12}{\varepsilon^2} \log\left(\frac{8m}{\delta}\right)$; $p \leftarrow 1$
2. **for** $i = 1$ **to** $m$ **do**
3. 
4. 
5. 
6. 
7. 
8. **Output** $\frac{|B|}{p}$

Bad$_i$: The value of $p$ after processing $i$ elements is less than $\frac{1}{\varepsilon^2 \cdot F_0}$. 
The Power of Adaptive Estimator

**Algorithm** Adaptive Estimator

1. **Initialize** $B \leftarrow \emptyset$; thresh $\leftarrow \frac{12}{\varepsilon^2} \log\left(\frac{8m}{d}\right)$; $p \leftarrow 1$
2. **for** $i = 1$ to $m$ do
3.   $B \leftarrow B \setminus \{a_i\}$
4.   With probability $p$, $B \leftarrow B \cup \{a_i\}$
5.   **while** $|B| = \text{thresh}$ do
6.     Throw away each element of $B$ with probability $\frac{1}{2}$
7.     $p \leftarrow \frac{p}{2}$
8. **Output** $\frac{|B|}{p}$

**Bad$_i$:** The value of $p$ after processing $i$ elements is less than $\frac{1}{\varepsilon^2 \cdot F_0}$.

**Claim 2** $\Pr[\text{Bad}_i] \leq \frac{\delta}{2 \cdot m}$
- For $p$ to fall below $\frac{1}{\varepsilon^2 \cdot F_0}$, it should be the case that if every element is sampled with $p = \frac{1}{\varepsilon^2 \cdot F_0}$, we would have $|B| \geq \text{thresh}$
- Apply Chernoff bound on sum of i.i.d. indicator variables
The Power of Adaptive Estimator

Algorithm Adaptive Estimator

1. Initialize $B \leftarrow \emptyset$; thresh $\leftarrow \frac{12}{\varepsilon^2} \log \left( \frac{8m}{\delta} \right)$; $p \leftarrow 1$
2. for $i = 1$ to $m$ do
3. $B \leftarrow B \setminus \{a_i\}$
4. With probability $p$, $B \leftarrow B \cup \{a_i\}$
5. while $|B| = \text{thresh}$ do
6. Throw away each element of $B$ with probability $\frac{1}{2}$
7. $p \leftarrow \frac{p}{2}$
8. Output $\frac{|B|}{p}$

Bad$_i$: The value of $p$ after processing $i$ elements is less than $\frac{1}{\varepsilon^2 \cdot F_0}$.

Claim 2 $\Pr[\text{Bad}_i] \leq \frac{\delta}{2m}$

- For $p$ to fall below $\frac{1}{\varepsilon^2 \cdot F_0}$, it should be the case that if every element is sampled with $p = \frac{1}{\varepsilon^2 \cdot F_0}$, we would have $|B| \geq \text{thresh}$
- Apply Chernoff bound on sum of i.i.d. indicator variables
The Power of Adaptive Estimator

Algorithm Adaptive Estimator

1: **Initialize** $B \leftarrow \emptyset$; $\text{thresh} \leftarrow \frac{12}{\varepsilon^2} \log \left( \frac{8m}{\delta} \right)$; $p \leftarrow 1$
2: **for** $i = 1$ to $m$ **do**
3: $B \leftarrow B \setminus \{a_i\}$
4: With probability $p$, $B \leftarrow B \cup \{a_i\}$
5: **while** $|B| = \text{thresh}$ **do**
6: Throw away each element of $B$ with probability $\frac{1}{2}$
7: $p \leftarrow \frac{p}{2}$
8: **Output** $\frac{|B|}{p}$

Bad$_i$: The value of $p$ after processing $i$ elements is less than $\frac{1}{\varepsilon^2.F_0}$.

**Claim 2** $\Pr[\text{Bad}_i] \leq \frac{\delta}{2.m}$

- For $p$ to fall below $\frac{1}{\varepsilon^2.F_0}$, it should be the case that if every element is sampled with $p = \frac{1}{\varepsilon^2.F_0}$, we would have $|B| \geq \text{thresh}$
- Apply Chernoff bound on sum of i.i.d. indicator variables

Error$_i$: $\frac{|B|}{p} \notin [(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]$ after processing $i$ elements.
The Power of Adaptive Estimator

**Algorithm** Adaptive Estimator

1. **Initialize** $B \leftarrow \emptyset$; thresh $\leftarrow \frac{12}{\varepsilon^2} \log\left(\frac{8m}{\delta}\right)$; $p \leftarrow 1$
2. **for** $i = 1$ to $m$ **do**
3.     $B \leftarrow B \setminus \{a_i\}$
4.     With probability $p$, $B \leftarrow B \cup \{a_i\}$
5.     **while** $|B| = \text{thresh}$ **do**
6.         Throw away each element of $B$ with probability $\frac{1}{2}$
7.     $p \leftarrow \frac{p}{2}$
8. **Output** $\frac{|B|}{p}$

**Bad**: The value of $p$ after processing $i$ elements is less than $\frac{1}{\varepsilon^2 \cdot F_0}$.  

**Claim 2** $\Pr[\text{Bad}] \leq \frac{\delta}{2 \cdot m}$
- For $p$ to fall below $\frac{1}{\varepsilon^2 \cdot F_0}$, it should be the case that if every element is sampled with $p = \frac{1}{\varepsilon^2 \cdot F_0}$, we would have $|B| \geq \text{thresh}$
- Apply Chernoff bound on sum of i.i.d. indicator variables

**Error**: $\frac{|B|}{p} \not\in [(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]$ after processing $i$ elements.

**Claim 3** $\Pr[\text{Error} \cap \text{Bad}] \leq \frac{\delta}{2m}$
- Apply Chernoff bound on sum of i.i.d. indicator variables
Algorithm Adaptive Estimator

1: Initialize $B \leftarrow \emptyset$; thresh $\leftarrow \frac{12}{\varepsilon^2} \log \left( \frac{8m}{\delta} \right)$; $p \leftarrow 1$
2: for $i = 1$ to $m$ do
3: $B \leftarrow B \setminus \{a_i\}$
4: With probability $p$, $B \leftarrow B \cup \{a_i\}$
5: while $|B| = \text{thresh}$ do
6: Throw away each element of $B$ with probability $\frac{1}{2}$
7: $p \leftarrow \frac{p}{2}$
8: Output $\frac{|B|}{p}$

Bad$_i$: The value of $p$ after processing $i$ elements is less than $\frac{1}{\varepsilon^2 F_0}$.

Claim 2 $\Pr[\text{Bad}_i] \leq \frac{\delta}{2m}$
- For $p$ to fall below $\frac{1}{\varepsilon^2 F_0}$, it should be the case that if every element is sampled with $p = \frac{1}{\varepsilon^2 F_0}$, we would have $|B| \geq \text{thresh}$
- Apply Chernoff bound on sum of i.i.d. indicator variables

Error$_i$: $\frac{|B|}{p} \notin [(1 - \varepsilon)F_0, (1 + \varepsilon)F_0]$ after processing $i$ elements.

Claim 3 $\Pr[\text{Error}_i \cap \text{Bad}] \leq \frac{\delta}{2m}$
- Apply Chernoff bound on sum of i.i.d. indicator variables

Lemma 1 $\Pr[\text{Error} = \bigcup_i \text{Error}_i] \leq \delta$

Correct Estimate after processing every element
Well, here we are

**Algorithm** $F_0$-estimator

**Input** Stream $\mathcal{D} = \langle a_1, a_2, \ldots, a_m \rangle$, $\varepsilon$, $\delta$

1. **Initialize** $p \leftarrow 1; \mathcal{B} \leftarrow \emptyset; \text{thresh} \leftarrow \frac{12}{\varepsilon^2} \log\left(\frac{8m}{\delta}\right)$
2. **for** $i = 1$ to $m$ **do**
3. 3. $\mathcal{B} \leftarrow \mathcal{B} \setminus \{a_i\}$
4. 4. With probability $p$, $\mathcal{B} \leftarrow \mathcal{B} \cup \{a_i\}$
5. 5. **while** $|\mathcal{B}| = \text{thresh}$ **do**
6. 6. Throw away each element of $\mathcal{B}$ with probability $\frac{1}{2}$
7. 7. $p \leftarrow \frac{p}{2}$
8. **Output** $\frac{|\mathcal{B}|}{p}$
The Power of Simplicity: Beyond the (Text) Book

- Naturally extends to setting where every element $a_i$ is replaced by $S_i \subseteq [n]$ and we are interested in computing $| \bigcup S_i |$

- Delphic Family of Sets
  - Representation Size: $O(\log n)$
  - Actions supported in $O(\log n)$ space and time:
    - **Cardinality**: Know the size of $S_i$
    - **Sample**: Sample uniformly at random elements from $S_i$
    - **Membership**: For an element $x \in [n]$, check if $x \in S_i$

- Importance of Delphic Sets in Practice
  - Estimation of the number of solutions of a DNF Formulas
  - Klee’s Measure Problem: Volume of $d$-dimensional rectangles
  - Test Coverage Estimation Problem
Delphic Sets In Practice: DNF Formulas

- Consider set of Boolean variables $Y = \{y_1, y_2, \ldots, y_k\}$
- $[n] = 2^Y$; $k = \log n$
- Every set $S_i$ is \textbf{implicitly} represented by a term $T_i$, which is conjunction of variables (or their negations); e.g., $\neg y_1 \land y_2 \land y_3$
- The corresponding $S_i$ is set of solutions of $T_i$
- Is it \textbf{Delphic}?
• Consider set of Boolean variables $Y = \{y_1, y_2, \ldots y_k\}$
• $[n] = 2^Y$; $k = \log n$
• Every set $S_i$ is implicitly represented by a term $T_i$, which is conjunction of variables (or their negations); e.g., $\neg y_1 \land y_2 \land y_3$
• The corresponding $S_i$ is set of solutions of $T_i$
• Is it Delphic?
  • Know the size of $S_i$: $O(k)$
  • Sample uniformly at random elements from $S_i$: $O(k)$
  • For an element $x \in [n]$, check if $x \in S_i$: $O(k)$
Delphic Sets In Practice: Klee’s Measure Problem

- Estimate the union of axis-parallel rectangles in $\mathbb{R}^d$; (Discrete version: so count the number of integer points)
- $n = \Delta^d$
- Every $S_i = [a_{i,1}, b_{i,1}] \times [a_{i,2}, b_{i,2}] \times \cdots \times [a_{i,d}, b_{i,d}]$ where $a_{i,j} \leq \Delta; b_{i,j} \leq \Delta$
- Is it Delphic?
Delphic Sets In Practice: Klee’s Measure Problem

- Estimate the union of axis-parallel rectangles in $\mathbb{R}^d$; (Discrete version: so count the number of integer points)
- $n = \Delta^d$
- Every $S_i = [a_{i,1}, b_{i,1}] \times [a_{i,2}, b_{i,2}] \ldots [a_{i,d}, b_{i,d}]$ where $a_{i,j} \leq \Delta$; $b_{i,j} \leq \Delta$
- Is it Delphic?
  - Know the size of $S_i$: $O(d \log |\Delta|) = O(\log n)$
  - Sample uniformly at random elements from $S_i$: $O(d \log |\Delta|) = O(\log n)$
  - For any element $x \in [n]$, check if $x \in S_i$: $O(d \log |\Delta|) = O(\log n)$
- **Open Problem**: Solve Klee's Measure Problem can be done with space and update-time complexity $\tilde{O}(poly(d, \log |\Delta|))$. 
Delphic Sets in Practice: Coverage Estimation

- Let \( Y = \{y_1, y_2, \ldots, y_k\} \) be set of features
- Every test vector assigns a value of 0 or 1 to every feature.
  - \((y_1 = 1, y_2 = 0, y_3 = 1, \ldots y_k = 1)\)
- Objectives:
  - (Achieve) There is at least one test where \( y_i \) is set to 1 and another where \( y_i \) is set to 0 (1-wise coverage)
  - (Achieve) For every \( i, j \), ensure there are four tests where \((y_{i_1}, y_{i_2})\) are set to \((0, 0), (1, 0), (0, 1), (1, 1)\) (2-wise coverage)
  - (Achieve) For every subsets of size \( t \), ensure there are \( 2^t \) tests where \((y_i_1, y_i_2, \ldots, y_i_k)\) are set to \((0, 0, \ldots, 0), (1, 0, \ldots, 0), (1, 1, \ldots, 1)\) (t-wise coverage)
Delphic Sets in Practice: Coverage Estimation

- Let $Y = \{y_1, y_2, \ldots y_k\}$ be set of features
- Every test vector assigns a value of 0 or 1 to every feature.
  - $(y_1 = 1, y_2 = 0, y_3 = 1, \ldots y_k = 1)$
- Objectives:
  - (Achieve) There is at least one test where $y_i$ is set to 1 and another where $y_i$ is set to 0 (1-wise coverage)
  - (Achieve) For every $i, j$, ensure there are four tests where $(y_{i_1}, y_{i_2})$ are set to $(0, 0), (1, 0), (0, 1), (1, 1)$ (2-wise coverage)
  - (Achieve) For every subsets of size $t$, ensure there are $2^t$ tests where $(y_{i_1}, y_{i_2}, \ldots y_{i_k})$ are set to $(0, 0, \ldots 0), (1, 0, \ldots 0), (1, 1, \ldots 1)$ (t-wise coverage)
- Problem: Given constraints on what test vectors are allowed, generate a test suite that maximizes $t$-wise coverage?
- Given set of tests, estimate the $t$-wise coverage.
  - A test vector specifies the set and it again satisfies the Delphic set properties
Prior Work: Streaming

- Could only handle when every $S_i$ is singleton
  - Strong reliance on hash functions
  - Previous attempts yielded update time complexity of $O(n)$ (Tirthpura and Woodruff 2012)
  - Time complexity arises due to the typical need for the emptiness check of $\{x : h(x) = 0, x \in S_i\}$. 
Prior Work: Streaming

- Could only handle when every $S_i$ is singleton
  - Strong reliance on hash functions
  - Previous attempts yielded update time complexity of $O(n)$ (Tirthpura and Woodruff 2012)
  - Time complexity arises due to the typical need for the emptiness check of $\{x : h(x) = 0, x \in S_i\}$.
Our Main Theorem

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>There is a very simple algorithm that takes in input a stream of Delphic sets</strong> $S_1, \ldots, S_m$, parameters $\varepsilon$ and $\delta$, and provides $(\varepsilon, \delta)$-estimate of $</td>
</tr>
</tbody>
</table>

- **Update-time complexity**: $\tilde{O}(\log^2(m/\delta) \cdot \varepsilon^{-2} \cdot \log n)$
- **Space complexity**: $O(\log(m/\delta) \cdot \varepsilon^{-2} \cdot \log n)$. |
Some implications of our result

- **Klee’s Measure Problem** Estimate the union of axis-parallel rectangles in $\mathbb{R}^d$. Our algorithm gives the first efficient algorithm with linear dependence on the dimension, $d$, - a long standing open problem. (PODS-21, PODS-22)

- **Model Counting for DNF** Count the number of DNF solutions. Our algorithm (nearly) matches the optimal bounds (in non-streaming setting!) The practical implementation (after engineering improvements) achieves nearly 100× speed up over prior state of the art f (IJCAI-23)

- **Coverage Estimation Problem** A critical importance of software testing is to estimate the amount of coverage that has been achieved with a certain set of “test vectors”. Our algorithm out-performs all the currently used techniques in practice. (ICSE-22)
Same Algorithm (nearly) works

**Algorithm** Delphic-Union

1. Initialize $\mathcal{B} \leftarrow \emptyset; p \leftarrow 1$
2. $\text{thresh} \leftarrow 3 \cdot \left( \frac{\log(2m/\delta)}{\epsilon^2} \right)$
3. for $i = 1$ to $m$ do
   4. for all $s \in \mathcal{B}$ do
      5. if $s \in S_i$ then remove $s$ from $\mathcal{B}$
   6. Pick each element of $S_i$ with probability $p$ add them to $\mathcal{B}$.
5. while $|\mathcal{B}| \geq \text{thresh}$ do
   6. Update $p = p/2$
   7. Throw away each element of $\mathcal{B}$ with probability $1/2$
10. Output $\frac{|\mathcal{B}|}{p}$
Same Algorithm (nearly) works

**Algorithm** Delphic-Union

1. Initialize $\mathcal{B} \leftarrow \emptyset; p \leftarrow 1$
2. $\text{thresh} \leftarrow 3 \cdot \left(\frac{\log(2m/\delta)}{\varepsilon^2}\right)$
3. for $i = 1$ to $m$ do
4. for all $s \in \mathcal{B}$ do
5. if $s \in S_i$ then remove $s$ from $\mathcal{B}$
6. Pick each element of $S_i$ with probability $p$ add them to $\mathcal{B}$.
7. while $|\mathcal{B}| \geq \text{thresh}$ do
8. Update $p = p/2$
9. Throw away each element of $\mathcal{B}$ with probability $1/2$
10. Output $\frac{|\mathcal{B}|}{p}$

**Challenge** Pick each element of $S_i$ with probability $p$ add them to $\mathcal{B}$. 
Algorithm Delphic-Union

1: Initialize $B \leftarrow \emptyset; p \leftarrow 1$
2: $\text{thresh} \leftarrow 3 \cdot \left( \frac{\log(2m/\delta)}{\epsilon^2} \right)$
3: for $i = 1$ to $m$ do
4:     for all $s \in B$ do
5:         if $s \in S_i$ then remove $s$ from $B$
6:     Pick each element of $S_i$ with probability $p$ add them to $B$.
7: while $|B| \geq \text{thresh}$ do
8:     Update $p = p/2$
9:     Throw away each element of $B$ with probability $1/2$
10: Output $\frac{|B|}{p}$

Challenge  
Pick each element of $S_i$ with probability $p$ add them to $B$.
- $N_i \leftarrow \text{Bin}(|S_i|, p)$
- Draw $N_i$ distinct elements from $S_i$ by drawing $N_i \log N_i \log(\frac{2m}{\delta})$ samples
Algorithm Delphic-Union

1: Initialize $B \leftarrow \emptyset; p \leftarrow 1$
2: thresh $\leftarrow 3 \cdot \left( \frac{\log(2m/\delta)}{\varepsilon^2} \right)$
3: for $i = 1$ to $m$ do
4:   for all $s \in B$ do
5:     if $s \in S_i$ then remove $s$ from $B$
6:   Pick each element of $S_i$ with probability $p$ add them to $B$.
7: while $|B| \geq$ thresh do
8:   Update $p = p/2$
9:   Throw away each element of $B$ with probability $1/2$
10: Output $\frac{|B|}{p}$

Challenge  Pick each element of $S_i$ with probability $p$ add them to $B$.

- $N_i \leftarrow \text{Bin}(|S_i|, p)$
- Draw $N_i$ distinct elements from $S_i$ by drawing $N_i \log N_i \log(\frac{2m}{\delta})$ samples

One Last thing: What if $N_i$ is too large? (Update time complexity)

- Well, just update $p$ to $p/2$ and resample $N_i \leftarrow \text{Bin}(N_i, 1/2)$ until $N_i < \text{thresh}$
### Algorithm Final Algorithm

1: Initialize $B \leftarrow \emptyset$; $p \leftarrow 1$; thresh $\leftarrow 3 \cdot \left(\frac{\log(2m/\delta)}{\varepsilon^2}\right)$

2: for $i = 1$ to $m$ do

3:     for all $s \in B$ do

4:         if $s \in S_i$ then remove $s$ from $B$

5:     $N_i \leftarrow \text{Bin}(|S_i|, p)$

6: while $|B| + N_i \geq \text{thresh}$ do

7:     $N_i \leftarrow \text{Bin}(N_i, 1/2)$ and $p \leftarrow p/2$

8:     Throw away each element of $B$ with probability $1/2$

9: Pick $N_i$ distinct elements of $S_i$ randomly and add them to $B$.

10: Output $\frac{|B|}{p}$
Here we are

**Algorithm** Final Algorithm

1: Initialize $\mathcal{B} \leftarrow \emptyset$; $p \leftarrow 1$; $\text{thresh} \leftarrow 3 \cdot \left(\frac{\log(2m/\delta)}{\epsilon^2}\right)$

2: for $i = 1$ to $m$ do

3: for all $s \in \mathcal{B}$ do

4: if $s \in S_i$ then remove $s$ from $\mathcal{B}$

5: $N_i \leftarrow \text{Bin}(|S_i|, p)$

6: while $|\mathcal{B}| + N_i \geq \text{thresh}$ do

7: $N_i \leftarrow \text{Bin}(N_i, 1/2)$ and $p \leftarrow p/2$

8: Throw away each element of $\mathcal{B}$ with probability $1/2$

9: Pick $N_i$ distinct elements of $S_i$ randomly and add them to $\mathcal{B}$.

10: Output $\frac{|\mathcal{B}|}{p}$

**Conclusion** A simple algorithm that generalizes and is practically efficient

**Further Work** Algorithm for Delphic sets without dependence on stream size ($m$)
Algorithm Final Algorithm

1: Initialize $\mathcal{B} \leftarrow \emptyset$; $p \leftarrow 1$; thresh $\leftarrow 3 \cdot \left(\frac{\log(2m/\delta)}{\varepsilon^2}\right)$

2: for $i = 1$ to $m$ do
3:     for all $s \in \mathcal{B}$ do
4:         if $s \in S_i$ then remove $s$ from $\mathcal{B}$
5:     $N_i \leftarrow \text{Bin}(|S_i|, p)$
6:     while $|\mathcal{B}| + N_i \geq \text{thresh}$ do
7:         $N_i \leftarrow \text{Bin}(N_i, 1/2)$ and $p \leftarrow p/2$
8:         Throw away each element of $\mathcal{B}$ with probability $1/2$
9:     Pick $N_i$ distinct elements of $S_i$ randomly and add them to $\mathcal{B}$.
10:    Output $\frac{|\mathcal{B}|}{p}$

Conclusion A simple algorithm that generalizes and is practically efficient
Further Work Algorithm for Delphic sets without dependence on stream size ($m$)
Open Problem Optimal algorithm for Delphic sets

These slides are available at https://www.cs.toronto.edu/~meel/talks.html
Knuth's Note: https://cs.stanford.edu/~knuth/papers/cvm-note.pdf