Motivation: Stream Processing

- In stream processing:
  - Data cannot be stored; one-pass
  - Analysis needs to be online – no waiting for answers
  - Time per update is limited
Motivation: Stream Processing

• Many of these trivial questions become extremely difficult for streams
  • How much traffic from/to a certain IP address?
  • How many distinct flows?
  • What are the heavy hitters?
Stream Mining

- Abstraction:
  - Stream is a continuous sequence of *items*.

- Problems:
  - Heavy hitters
    - Several items.
  - How many distinct items do I have in my stream?
    - Several items (6).
  - Frequent items in the stream
    - 3 or more.
Stream Mining

- It won’t always be possible to give an exact answer
  - Therefore relaxations

- Popular: $\varepsilon, \delta$ - approximation:
  - In 1-$\delta$ of the cases we are at most $\varepsilon$ off.

- We will show three examples of stream mining algorithms:
  - Min-wise sampling
  - Number of Distinct Items (min-hash)
  - Frequent items
Outline

- Some Basic Techniques
  - I. Heavy hitters
  - II. Frequent items
- Sketching
  - III. Distinct count sketches
  - IV. Count-Min Sketch
- Semi-streaming:
  - V. Neighborhood function
  - VI. Counting local triangles
- Conclusion
I. Heavy Hitters

• “Given a stream, identify all items that occur more than 10% of the time”
I. Heavy Hitters

• “Given a stream, identify all items that occur more than 10% of the time”

Solution storing 9 colors and counters:

• Summary={}
• For each item that arrives
  – If (item, count) is in Summary:
    update count to count + 1
  – Else if |S|<10:
    add (item, 1) to S
  – Else:
    decrease the count of all pairs in S
    remove all pairs with count = 0
I. Heavy Hitters

- “Given a stream, identify all items that occur more than 10% of the time”

Solution storing 9 colors and counters:

- Guarantee: if an item appeared more than 10% of time, there will be an entry (●, count) in the summary
- Disadvantage: there may be false positives
- Obviously extendible to other thresholds
  - Frequency threshold $1/k \rightarrow k-1$ memory places
Outline

- **Some Basic Techniques**
  - I. Heavy hitters
  - II. Frequent items
- **Sketching**
  - III. Distinct count sketches
  - IV. Count-Min Sketch
- **Semi-streaming:**
  - V. Neighborhood function
  - VI. Counting local triangles
- **Conclusion**
II. Identify Frequent Items

• Counting every item is impossible
  • E.g., all pairs of people that phone to each other

• We do not know on beforehand which combinations will be frequent

• Example:

30 items;●:8,●:6,●:5
All others are 3
If frequency is 20%:● and● need to be outputted
II. Identify Frequent Items

- The following algorithm finds a *superset* of the s-frequent items:
  - Initialization: none of the items has a counter
  - Item • enters at time t:
    - If • has a counter: counter(•) ++
    - Else:
      - counter(•) = 1
      - start(•) = t
    - For all other counters • do:
      - If counter(•) / ( t – start(•) + 1 ) < s:
        - Delete counter(•), start(•)
  - When the frequent items are needed: return all items that have a counter
II. Identify Frequent Items

- **Example: (20%)**
  
<table>
<thead>
<tr>
<th>start</th>
<th># (freq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (100%)</td>
</tr>
</tbody>
</table>
II. Identify Frequent Items

• **Example:** (20%)
  
<table>
<thead>
<tr>
<th>start</th>
<th># (freq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (50%)</td>
</tr>
<tr>
<td>2</td>
<td>1 (100%)</td>
</tr>
</tbody>
</table>
II. Identify Frequent Items

- **Example: (20%)**
  - 1
  - 2
  - 3
  - 4

<table>
<thead>
<tr>
<th>start</th>
<th># (freq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (20%)</td>
</tr>
<tr>
<td>2</td>
<td>1 (25%)</td>
</tr>
<tr>
<td>3</td>
<td>2 (66%)</td>
</tr>
<tr>
<td>4</td>
<td>1 (50%)</td>
</tr>
</tbody>
</table>
II. Identify Frequent Items

- **Example: (20%)**

<table>
<thead>
<tr>
<th>start</th>
<th># (freq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (17%)</td>
</tr>
<tr>
<td>2</td>
<td>1 (20%)</td>
</tr>
<tr>
<td>3</td>
<td>2 (50%)</td>
</tr>
<tr>
<td>4</td>
<td>1 (33%)</td>
</tr>
<tr>
<td>5</td>
<td>1 (100%)</td>
</tr>
</tbody>
</table>
II. Identify Frequent Items

• **Example: (20%)**

<table>
<thead>
<tr>
<th>start</th>
<th># (freq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2 (25%)</td>
</tr>
<tr>
<td>3</td>
<td>2 (29%)</td>
</tr>
<tr>
<td>6</td>
<td>1 (25%)</td>
</tr>
<tr>
<td>8</td>
<td>2 (100%)</td>
</tr>
</tbody>
</table>
## II. Identify Frequent Items

**Example: (20%)**

<table>
<thead>
<tr>
<th>start</th>
<th># (freq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 (25%)</td>
</tr>
<tr>
<td>17</td>
<td>4 (29%)</td>
</tr>
<tr>
<td>27</td>
<td>1 (25%)</td>
</tr>
<tr>
<td>8</td>
<td>6 (26%)</td>
</tr>
<tr>
<td>19</td>
<td>3 (25%)</td>
</tr>
</tbody>
</table>
II. Identify Frequent Items

- Why does it work?
  - If \( \bullet \) is not recorded, \( \bullet \) is not frequent in the stream

- Imagine marking when \( \bullet \) was recorded:
  - If \( \bullet \) occurs, recording starts
  - Only stopped if \( \bullet \) becomes infrequent since start recording

- Whole stream can be partitioned into parts in which \( \bullet \) is not frequent \( \Rightarrow \) \( \bullet \) is not frequent in the whole stream

Algorithm is called “lossy counting”
II. Lossy Counting – Space Requirements

• Let $N$ be the length of the stream
• $s$ minimal frequency threshold. Let $k = \frac{1}{s}$

• Item $a$ is in the summary if:
  • $a$ appears once among last $k$ items
  • $a$ appears twice among last $2k$ items
  • ...
  • $a$ appears $x$ times among last $xk$ items
  • ...
  • $a$ appears $sN$ times among last $N$ items
II. Lossy Counting – Space Requirements

- Divide stream in blocks of size $k = \frac{1}{s}$

- Constellation with maximum number of candidates:
  - k candidates; “consume” 4 elements
  - k candidates; “consume” 3 elements
  - k candidates; “consume” 2 elements
  - k candidates; “consume” 1 element

- k/4 different each appears 4 times
- k/3 different each appears 3 times
- k/2 different each appears 2 times
- k different each appears 1 time
II. Lossy Counting – Space Requirements

• Hence total space requirement:
  \[ \sum_{i=1\ldots N/k} k/i \approx k \log(N/k) \]

• Recall: \( k = 1/s \)

• Worst case space requirement: \( 1/s \log(Ns) \)
II. Lossy Counting – Guarantee

- Suppose that we want to know the frequency up to a factor $\varepsilon$
  - Same algorithm, yet use $\varepsilon$ as minimum support threshold
  - Report all items with count $\geq (s- \varepsilon) N$

- Guaranteed: true frequency in the interval $[\text{count}/N, \text{count}/N+\varepsilon]$
II. Lossy Counting - Summary

• Worst case space consumption: \( \frac{1}{\varepsilon} \log(N\varepsilon) \)

• Guarantee: with 100% certainty, the relative error for all s-frequent itemsets is \( \varepsilon \)

• Performs very well in practice
  • Optimization: check if item is frequent only every \( \frac{1}{\varepsilon} \) steps
Outline

• **Some Basic Techniques**
  • I. Heavy hitters
  • II. Frequent items

• **Sketching**
  • III. Distinct count sketches
  • IV. Count-Min Sketch

• **Semi-streaming:**
  • V. Neighborhood function
  • VI. Counting local triangles

• **Conclusion**
III. How Many Different Items do I have?

- Number of distinct items is too big to keep all in memory

- Observation:
  If $h(.)$ is a hash function: every $x_i \rightarrow [0,1]$
  Maintain $\min\{ h(x_1), h(x_2), ..., h(x_n) \}$
  $E[ \min \{ h(x_1), h(x_2), ..., h(x_n) \} ] = 1/(1+D)$
  with $D = | \{ x_1, x_2, ..., x_n \} |$

- Average over many (independent) $h$ to decrease variance
- Called: min-hash algorithm
III. How Many Different Items do I have?

- **Example:**

  
  \[ \text{Min } h(x) = 0.13 \]
  
  \[ 1/(1+d) = 0.13 \Rightarrow d = 1/0.13 - 1 \approx 6.7 \]

- **Min** \( h(x) = 0.13 \)

- **Estimate D:** \( 1/(1+d) = 0.13 \Rightarrow d = 1/0.13 - 1 \approx 6.7 \)

- **Averaging over independent trials makes the result more accurate**
III. How Many Different Items do I have?

• Many variations on the same idea

• Multiple hash-functions $h_1 \ldots h_k$
  - $H_1 \rightarrow$ estimate 1  mean D  high variance
  - $H_2 \rightarrow$ estimate 2  mean D  high variance
  - ...
  - $H_k \rightarrow$ estimate k  mean D  high variance
  - Median \{estimate\_i\}  mean D  low variance

• HyperLogLog sketch: count 1,000,000,000 items with 2% error $\rightarrow$ 1.5kB
Stream still too fast?

- No problem; easily parallelizable
  - $\min (\min(A), \min(B)) = \min(A \cup B)$

Local computation

- $\min h_1, \ldots, \min h_k$

Global minimum

- $\min h_1, \ldots, \min h_k$
Outline

• Some Basic Techniques
  • I. Heavy hitters
  • II. Frequent items
• Sketching
  • III. Distinct count sketches
  • IV. Count-Min Sketch
• Semi-streaming:
  • V. Neighborhood function
  • VI. Counting local triangles
• Conclusion
IV. Sketching

- **Extension of the model**
  - **Items + numbers**
    - (a,5) → add 5 to a
    - (a,-3) → subtract 3 from the count of item a
  - **Query**
    - Sum for item a

- **New technique based upon a sketch**
  - Smart summary of the data
IV. Sketching

• There is not enough space to store sums for all items

• Instead we will store a \((d \times n)\) – matrix \(S\)
  • We have \(d\) hash functions \(h_1, \ldots, h_d\)
  • The counts of item \(i_t\) are stored in cells \(S[1,h_1(i_t)], \ldots, S[d,h_d(i_t)]\)
IV. Sketching

- Notice that there will be collisions:

- For the non-negative case:
  - all cells $S[1,h_1(i_t)], \ldots , S[d,h_d(i_t)]$ will be overestimations of the count of $i_t$
  - Return $\min(S[1,h_1(i_t)], \ldots , S[d,h_d(i_t)])$
### Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th>h1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Stream:**
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **Stream:**
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Stream:** .  .
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Stream:**
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>h2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>h4</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Stream:** 🟠🟢🟡🔴
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Stream:** 🍊 🍊 🍊 🍊 🍊
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

```
<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>h2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>h3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>h4</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
```

- **Stream:** 🔴 🔴 🔴 🔴 🔴 🔴
**Example: Count-min Sketch**

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th>h1</th>
<th>3</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>h3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>h4</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Stream:** 🟠🟢🟠🟢🟢🟢🟢🟢

---

**Note:** The table represents the CM-Sketch with 3 columns and 4 rows, showing how the counts are distributed across the hash functions for each element in the stream.
**Example: Count-min Sketch**

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>h2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>h3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>h4</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- **Stream:**

---

**Diagram:**

- Stream icons: □ □ □ □ □ □ □ □ □
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

  h1 | 4 | 0 | 5  
  h2 | 3 | 2 | 4  
  h3 | 4 | 3 | 2  
  h4 | 6 | 1 | 2  

- **Stream:** [●●●●●●●●●●]
Example: Count-min Sketch

- CM-Sketch with 3 columns and 4 rows

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Stream: ⬤⬤⬤⬤⬤⬤⬤⬤
**Example: Count-min Sketch**

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>h2</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>h3</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>h4</td>
<td>7</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

- **Stream:** 🟢🟠🟠🟠🟠🟠🟠🟠🟠🟠
Example: Count-min Sketch

**CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th>Column</th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Stream:** • • • • • • • • • • • • •
Example: Count-min Sketch

- **CM-Sketch with 3 columns and 4 rows**

<table>
<thead>
<tr>
<th></th>
<th>h1</th>
<th>h2</th>
<th>h3</th>
<th>h4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Stream:** 🟠🟢🔵🟠🟢🔵🟢🟢🔴🟢🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🟢🟢🔴🟢🌏🟢🔴🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢🟢 trò
Example: Count-min Sketch

• CM-Sketch with 3 columns and 4 rows

<table>
<thead>
<tr>
<th></th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>h1</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>h2</td>
<td>3</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>h3</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>h4</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

• Stream:

• Report frequencies:

<table>
<thead>
<tr>
<th>estimate</th>
<th>true count</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
IV. Sketching

• Usually for many more items than in the example
  • Number of items usually exceeds number of cells by orders of magnitude
• Especially effective if only few “heavy” items, many rare items
  • E.g., Zipfian distribution

• Tight guarantees on the estimation

\[ w = \left\lceil \frac{\epsilon}{\delta} \right\rceil \text{ and } d = \left\lceil \ln \frac{1}{\delta} \right\rceil; \ h_1, \ldots, h_d \text{ pairwise independent with probability } 1 - \delta, \ \hat{a_i} \leq a_i + \epsilon \|a\|_1 \]
Outline

• Some Basic Techniques
  • I. Heavy hitters
  • II. Frequent items

• Sketching
  • III. Distinct count sketches
  • IV. Count-Min Sketch

• Semi-streaming:
  • V. Neighborhood function
  • VI. Counting local triangles

• Conclusion
V. Neighborhood Function

- Count the number of pairs of nodes at distance 1, 2, 3, ...

- Important statistics; allows to compute average degree, diameter, effective diameter.
V. Neighborhood Function

- **Straightforward algorithm**
  
  Set $N_0(v) = \{v\}$

  For $i = 1$ to $r$:
  
  For all $v$ in $V$:

  $N_i(v) = N_{i-1}(v)$

  For $\{v, w\}$ in $E$:

  $N_i(v) \leftarrow N_i(v) \cup N_{i-1}(w)$

  $N_i(w) \leftarrow N_i(w) \cup N_{i-1}(v)$

  Return $\text{avg}(|N_1(v)|), \text{avg}(|N_2(v)| - |N_1(v)|), \ldots$

- **Time:** $O(r |V| |E|)$

- **Space:** $O(|V|^2)$
V. Neighborhood Function

- **Observation**: we can replace every set by a *summary*
  - Take union, cardinality, add an element

- **Size of set**: $V$ versus size of summary: $k \ll |V|$
  - $|V|$ versus $\log(\log(|V|))$
  - With the summary we can:

    - **Time**: $O( r \ k \ |E| )$
    - **Space**: $O( k \ |V| )$

- **Speedup** is enormous (1000s of times faster!)
Outline

- Some Basic Techniques
  - I. Heavy hitters
  - II. Frequent items

- Sketching
  - III. Distinct count sketches
  - IV. Count-Min Sketch

- Semi-streaming:
  - V. Neighborhood function
  - VI. Counting local triangles

- Conclusion
Example of an application of stream processing for attacking a truly big data problem

Given a graph, count, for every node, in how many triangles it appears

Becchetti et al. Efficient Semi-streaming algorithms for local triangle counting in massive graphs. In: KDD’08
• Example of an application of stream processing for attacking a truly big data problem

• Given a graph, count, for every node, in how many triangles it appears
VI. Streaming Graph Processing

• Example of an application of stream processing for attacking a truly big data problem

• Given a graph, count, for every node, in how many triangles it appears
VI. Streaming Graph Processing

- Example of an application of stream processing for attacking a truly big data problem

- Given a graph, count, for every node, in how many triangles it appears
VI. Streaming Graph Processing

- Example of an application of stream processing for attacking a truly big data problem

- Given a graph, count, for every node, in how many triangles it appears
  - Indicator for connectedness of the node into the community
VI. Storage Model

- **Graph stored as a stream of edges**
  
<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

- **Random access is expensive**
- **Access data using limited number of linear scans**
VI. Counting Triangles - Notation

- $S(u)$: neighbors of $u$
- $T(u)$: number of triangles in which $u$ is involved
- $d_u$: degree of $u$
- Local clustering coefficient:
  \[
  \frac{2 \times T(u)}{d_u(d_u-1)}
  \]

**WHY counting triangles?** $T(u)$ and local clustering coefficient are informative features for many problems.
VI. Counting Triangles

Figure from: Becchetti et al. Efficient Semi-streaming algorithms for local triangle counting in massive graphs. In: KDD’08

[Graph showing the distribution of triangles with an annotation: Straightforward, exact algorithm: $O(V^3)$]
VI. We Need Brains, Not Just More Power …

- N processors can speed up only a factor N at most
  - So, for N nodes, we need $N^2$ processors to make it linear

- Solution will be based upon:
  $$T(u) = \sum_{v \in S(u)} \frac{|S(u) \cap S(v)|}{2}$$
  and a smart way to do intersection *approximately*

- Building block: estimate for the “Jaccard coefficient”
VI. Brute Force- Example

1. Compute

\[ S(a) = \{b,c,d,e\} \]
\[ S(b) = \{a,c,d,e\} \]
\[ S(c) = \{a,b,d\} \]
\[ S(d) = \{a,b,c\} \]
\[ S(e) = \{a,b\} \]

2. Initialize all T(u) to 0

3. Iterate over all edges (u,v)

Add \(|S(u) \cap S(v)|\) to T(u) and T(v)

4. Divide all T(u) by 2

Too big to fit into memory

Random access to secondary storage

<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
VI. Building Block: Jaccard Coefficient

\[ J(A,B) = \frac{|A \cap B|}{|A \cup B|} \]

Indicates how similar the sets A and B are.

Example:

\[ J(\{a,b,c\},\{c,d\}) = \frac{1}{4} \]
\[ J(\{a,b,c\},\{b,c,d\}) = \frac{2}{4} \]

Used, e.g., to detect near duplicates (Altavista)

A set of n-grams in document 1
B set of n-grams in document 2
VI. Building Block: Jaccard Coefficient

Let $A$, $B$ be subsets of $U$

$h$ is a function mapping elements of $U$ to $\{1,2,\ldots,|U|\}$

Example: $d \rightarrow 1$, $c \rightarrow 2$, $a \rightarrow 3$, $b \rightarrow 4$

Let $\min_h(A) := \min_{a \in A} h(a)$

\[
\Pr[ \min_h(A) = \min_h(B) ] = \Pr[ \text{min of all elements in } A \cup B \text{ is in } A \cap B ] = \frac{|A \cap B|}{|A \cup B|} = J(A,B)
\]
VI. Building Block: Jaccard Coefficient

For random $h$, $\Pr[ \min_h(A) = \min_h(B) ] = J(A,B)$

“estimate” this probability by sampling many independent $h$

$\Rightarrow$ excellent estimate of $J(A,B)$

$|A \cap B| = J(A,B)$
$|A \cup B| = J(A,B) (|A|+|B|-|A \cap B|)$
$= (|A| + |B|) J(A,B) / (1+J(A,B))$
VI. Building Block: Jaccard Coefficient

- Independent functions $h_1, \ldots, h_m$
- “signature” of set $A$:
  $|A|$ and vector $(\min_{h_1}(A), \min_{h_2}(A), \ldots, \min_{h_m}(A))$

- Estimating $|A \cap B|$
  - $(a_1, \ldots, a_m)$ vector for $A$
  - $(b_1, \ldots, b_m)$ vector for $B$
  Let $e = \# \{ i \mid a_i=b_i \}$
  $e / m$ is an estimator for $J(A,B)$

$$|A \cap B| \approx (|A| + |B|) \frac{e}{(m + e)}$$
VI. Building Block: Jaccard Coefficient

Example: \( U = \{ a, b, c, d, e \} \)

\( A = \{ a, b \} \)
\( B = \{ b, c, d \} \)
\( C = \{ a, b, c, e \} \)

<table>
<thead>
<tr>
<th></th>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>( h_3 )</th>
<th>( h_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ J(A,B) = \frac{1}{4} ; \text{ estimate: } 0 \]
\[ J(A,C) = \frac{1}{2} ; \text{ estimate: } \frac{1}{2} \]
\[ J(B,C) = \frac{2}{5} ; \text{ estimate: } \frac{1}{4} \]

\[ \Rightarrow \quad 0 \]
\[ \Rightarrow \quad 6 \times \frac{2}{6} = 2 \]
\[ \Rightarrow \quad 7 \times \frac{1}{5} = \frac{7}{5} \]
VI. The Algorithm

- **Memory requirements:**
  - Main memory: couple of bytes per vertex
  - External memory: One entry for every edge $e$

- **Based upon** $T(u) = \sum_{v \in S(u)} |S(u) \cap S(v)| / 2$
  - For every edge $(u,v)$ we maintain estimate of $|S(u) \cap S(v)|$ in external memory
    - Using $m$ functions $h_1, h_2, \ldots, h_m$
VI. Intelligent Intersection Algorithm - Example

1. Compute

\[
\begin{align*}
\text{Sig}(a) &= (a_1, \ldots, a_m) \\
\text{Sig}(b) &= (b_1, \ldots, b_m) \\
\text{Sig}(c) &= (c_1, \ldots, c_m) \\
\text{Sig}(d) &= (d_1, \ldots, d_m) \\
\text{Sig}(e) &= (e_1, \ldots, e_m)
\end{align*}
\]

Still quite expensive on memory

<table>
<thead>
<tr>
<th>src</th>
<th>dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

2. Initialize all $T(u)$ to 0

3. Iterate over all edges $(u,v)$

   Compute $e = \# \{ i \mid u_i = v_i \}$
   
   Estimate $|S(u) \cap S(v)|$ based upon $e$
   
   Add estimate of $|S(u) \cap S(v)|$ to $T(u)$ and $T(v)$

4. Divide all $T(u)$ by 2
VI. Intelligent Intersection Algorithm - Example

For $p = 1$ to $m$:

1. Compute
   
   $\text{Sig}(a) = h_p(S(a))$
   
   $\ldots$
   
   $\text{Sig}(e) = h_p(S(e))$

2. Iterate over all edges $(u,v)$
   
   If $p==1$: initialize $Z_{uv}$ to 0
   
   If $h_p(u) == h_p(v)$: add 1 to $Z_{uv}$

Iterate over all $Z_{uv}$:

   Estimate $|S(u) \cap S(v)|$ based upon $Z_{uv}$
   
   Add estimate of $|S(u) \cap S(v)|$ to $T(u)$ and $T(v)$

Divide all $T(u)$ by 2
VI. The Complete Algorithm

for p : 1 to m
    for every vertex v
        min(v) := ∞
    for every edge (v,w)
        min(v) := min(min(v), h_p(w))
        min(w) := min(min(w), h_p(v))
    for every edge (v,w)
        if p==1 then Z_{v,w} := 0
        if min(v) == min(w) then
            Z_{v,w} := Z_{v,w} + 1
    for every Z_{v,w} :
        T(v) := T(v) + estimate of |S(v) ∩ S(w)|
        T(w) := T(w) + estimate of |S(v) ∩ S(w)|
    for all vertices v:
        T(v) := T(v)/2
VI. The Complete Algorithm

for p : 1 to m

for every vertex v
  min(v) := ∞

for every edge (v, w)
  min(v) := min(min(v), h_p(w))
  min(w) := min(min(w), h_p(v))

for every edge (v, w)
  if p == 1 then
    Z_{v,w} := 0
  if min(v) == min(w) then
    Z_{v,w} := Z_{v,w} + 1

for every Z_{v, w}:
  T(v) := T(v) + estimate of |S(v) ∩ S(w)|
  T(w) := T(w) + estimate of |S(v) ∩ S(w)|

for all vertices v:
  T(v) := T(v)/2

min(u) for all vertices u: in memory
T(u) for all vertices: in memory
Z_{u,v} for all edges (u, v): on disk
VI. Counting Triangles

• Reduce complexity from $|V|^3$ to $O(m|E|)$

• Computing power is great, but only gives you an at most linear speed-up

• Willingness to sacrifice exactness leads to incredible performance gains

• Resulting approximation still excellent feature
Outline

- Some Basic Techniques
  - I. Heavy hitters
  - II. Frequent items
- Sketching
  - III. Distinct count sketches
  - IV. Count-Min Sketch
- Semi-streaming:
  - V. Neighborhood function
  - VI. Counting local triangles
- Conclusion
Conclusion

- **Stream mining:**
  - Severe computational restrictions
  - Yet, surprisingly many operations are still possible
    - Heavy hitters
    - Number of distinct items
    - Frequent items
    - “Cash register”

- Counting triangles and neighborhood function as applications