Mergeable Summaries and the DataSketches Library

Edo Liberty, Principal Scientist Head of Amazon AI Labs

Justin Thaler Assistant Professor Georgetown University.

Lee Rhodes Distinguished Architect Oath, Inc.

Alaxander Saydakov Senior Software Engineer Oath, Inc

Kevin Lang Principal Scientist Oath, Inc

Jon Malkin Senior Scientist Oath, Inc
Single machine data processing
Distributed storage
Distributed compute (map/reduce, MPI, ...)
Distributed model (indexes, databases, Spark...)
207 big-data infographics (a meta infographic)
The streaming model

The World

Compute

Sketch

Query

Query Algorithm

Result

Result
Mergeable Summaries

The World

Compute + Sketch
Compute + Sketch
Compute + Sketch
Compute + Sketch

Merge+ Sketch

Result

Query Algorithm

Query
Unique Counting with Map Reduce

This is a shuffle operation which is very compute and network heavy!
Unique Counting with Mergeable Summaries

No shuffle operation is needed!
Data Mining with Traditional Windowing

Every dataset is processed 3 times for a model consisting of 3 days
Data Mining with Mergeable Summaries

Every dataset is processed once for a model consisting of the entire history.
Dynamic Windowing with Mergeable Summaries
OLAP with Mergeable Summaries

“Median latency of IoT device call in Westeros January 2018”

“Median latency of IoT device call in The North Q1 2018”
Some Basic Problems are Impossible
Others Have Great Solutions

**Items**
(words, IP-addresses, events, clicks,...)
- Counting distinct elements
- Item frequencies
- Approximate Quantiles
- Moment and entropy estimation
- Approximate set operations
- Sampling

**Matrices**
(text corpora, recommendations, ...)
- Covariance estimation matrix
- Low rank approximation
- Sparsification

**Vectors**
(text documents, images, example features,...)
- Dimensionality reduction
- Clustering (k-means, k-median,...)
- Linear Regression
- Machine learning (some of it at least)
- Density Estimation / Anomaly detection

**Graphs**
(social networks, communications, ...)
- Connectivity
- Cut Sparsification
- Weighted Matching
Data Sketches – Open Source Project


```bash
>> brew tap DataSketches/sketches-cmd
>> brew install data-sketches
```
Production ready
New Research

A high-performance algorithm for identifying frequent items in data streams.

[DLRT16] Anirban Dasgupta, Kevin J. Lang, Lee Rhodes, and Justin Thaler.

[KLL16] Zohar S. Karnin, Kevin J. Lang, and Edo Liberty.


[LMTU16] Edo Liberty, Michael Mitzenmacher, Justin Thaler, and Jonathan Ullman.

In this talk

**Items**
(words, IP-addresses, events, clicks,...)
- Counting distinct elements
- Item frequencies
- Approximate Quantiles
- Moment and entropy estimation
- Approximate set operations
- Sampling

**Matrices**
(text corpora, recommendations, ...)
- Covariance estimation matrix
- Low rank approximation
- Sparsification

**Vectors**
(text documents, images, example features,...)
- Dimensionality reduction
- Clustering (k-means, k-median,...)
- Linear Regression
- Machine learning (some of it at least)
- Density Estimation / Anomaly detection

**Graphs**
(social networks, communications, ...)
- Connectivity
- Cut Sparsification
- Weighted Matching
Counting Distinct Elements

N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments
E. Cohen. All-distances sketches, revisited: HIP estimators for massive graphs analysis
E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches
G. Cormode. Sketch techniques for massive data
P. Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications
D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem
M. Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence
Problem Definition

Approximate the number of distinct items in the stream

- # of unique IPs are important statistics of networks
- # of different customers using web services
- # of unique keys in a database join table
- ...

...
General Hashing Idea

- Map all entries to the interval (0,1) using a hash function
- Keep only $k$ values smaller than some threshold $\theta$.
- From $(k, \theta)$ we can approximate the number of unique items.
Our results

- Generalize a family of algorithms (Adaptive sampling, KMV)
- New Variance bounds for all such algorithms
- New tradeoffs between accuracy, space, and update time (alpha alg’)
- Very careful implementation
Experimental Results

**Equal Space Comparison of \((\text{Total Processing Time}) / N)\)

- KMV: Blue line
- Adapt: Green line
- Alpha: Red line

**Equal Space Comparison of Standard Error**

- KMV: Blue line
- Adapt: Green line
- Alpha: Red line

*Graphs show time in nanoseconds and standard error over varying numbers of unique items in the stream.*
Quick Demo
Counting Distinct Elements
Assume you need to estimate the number of unique numbers in a file

```
>>head data.csv
0
1
0
3
0
2
3
7
3
2
```

In this file, row i tasks integer value from [0,i] uniformly at random.
Some stats: there are 10,000,000 such numbers in this ~76Mb file.

```bash
>>time wc -lc data.csv
 10000000 76046666 data.csv

real  0m0.101s
user  0m0.072s
sys   0m0.021s
```

Reading the file take ~1/10 seconds. We don’t foresee IO being an issue.
To count the number of distinct items you might try this:

```
>>sort data.csv | uniq | wc -l
```

However, it is faster to “uniqify” while sorting.

```
>>sort data.csv -u | wc -l
```

```
>>time sort data.csv -u | wc -l
5001233
real 2m37.071s
user 2m36.587s
sys 0m0.376s
```
Still, most of the time is spent on comparing strings....

```
>>sort data.csv -u -n -S 100% | wc -l
```

This is much better!

```
>>time sort data.csv -u -n | wc -l
5001233
real 0m11.809s
user 0m11.587s
sys 0m0.228s
```
This is the way to do this with the sketching library

```
>>cat data.csv | ds theta
```

```
>>time cat data.csv | ds theta
4835874.828244
4974249.044005
5116568.411181
real 0m1.539s
user 0m2.184s
sys 0m0.310s
```

Too fast for system monitor UI...

It uses ~ 32k of memory!
Weighted Item frequencies

Space-optimal heavy hitters with strong error bounds. J. R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss
An optimal algorithm for 1-heavy hitters in insertion streams and related problems A. Bhattacharyya, P. Dey, and D. P. Woodruff
Finding frequent items in data streams M. Charikar, K. Chen, and M. Farach-Colton
Methods for finding frequent items in data streams G. Cormode and M. Hadjieleftheriou
Approximate frequency counts over data streams G. S. Manku and R. Motwani.
Efficient computation of frequent and top-k elements in data streams A. Metwally, D. Agrawal, and A. El Abbadi.
Finding repeated elements J. Misra and D. Gries.

A High-Performance Algorithm for Identifying Frequent Items in Data Streams
Daniel Anderson Pryce Bevin, Kevin Lang, Edo Liberty, Lee Rhodes, Justin Thaler
Problem Definition

\[ w(\bullet) = w_0(x) \]

\[ |w'(x) - w(x)| \leq \varepsilon W \]

\[ W = \sum_i w_i \]
Our Contributions

• Improved streaming algorithm for weighted updates

• Improved merging procedure

• Improved Estimator

• Careful implementation
Comparable Error
Significantly Faster Updates
Weighted Item Frequencies Application
Threading Machine Generated Email

Ailon, Karnin, Maarek, Liberty, Threading Machine Generated Email, WSDM 2013
If \( b \) should be threaded with \( a \) then the lift should be large

\[
\text{lift}(a, b) = \frac{p(b|a)}{p(b)} \gg 1
\]

Alas, computing all pair conditional probability is impossible!

\[
\text{lift}(a, b) = \frac{n(b, a)}{n(b)n(a)} = \frac{n}{n(b)n(a)} = n(b, a)w(b)
\]

This is possible with weighted frequency sketching!
Threading Machine Generated Email

PayPal.com: “You submitted an order in the amount of $ usd to overstock.com.”

overstock.com: “Overstock.com password reset request.”

payless.com “Order confirmation”

C=632
w=1,221

C=652
w=1,300

C=652
w=1,300

C=193
w=12,098

payless.com “Your order is shipped”

C=1,742
w=6,446

overstock.com: “Your overstock.com order has shipped.”

C=769
w=1,490

C=753
w=1,395

C=153
w=704
Threading Machine Generated Email

- Order Confirmation (retail) 64% → Shipping Notification
- 19%

- Utility bill payment due 44% → Payment received

- Insurance payment due 35% → Service cancellation 53%
- 15%
Streaming quantiles

Manku, Rajagopalan, Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets.
Munro, Paterson. Selection and sorting with limited storage.
Greenwald, Khanna. Space-efficient online computation of quantile summaries.
Wang, Luo, Yi, Cormode. Quantiles over data streams: An experimental study.
Greenwald, Khanna. Quantiles and equidepth histograms over streams.
Agarwal, Cormode, Huang, Phillips, Wei, Yi. Mergeable summaries.
Felber, Ostrovsky. A randomized online quantile summary in $O((1/\epsilon) \log(1/\epsilon))$ words.
Lang, Karnin, Liberty, Optimal Quantile Approximation in Streams.
Ivking, Lang, Karnin, Liberty, Braverman, Streaming quantiles algorithms with small space and update time
Problem Definition

Sketch the stream to estimate \(|R' - R| < \varepsilon n\)
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Simple?</th>
<th>Mergeable</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform sampling</td>
<td>✓</td>
<td>✓</td>
<td>$1/\varepsilon^2$</td>
</tr>
<tr>
<td>Greenwald Khanna (GK)</td>
<td>✗</td>
<td>✗</td>
<td>$\log(n)/\varepsilon$</td>
</tr>
<tr>
<td>Felber-Ostrovsky</td>
<td>✗</td>
<td>✗</td>
<td>$\log(1/\varepsilon)/\varepsilon$</td>
</tr>
<tr>
<td>Manku-Rajagopalan-Lindsay (MRL)</td>
<td>✓</td>
<td>✓</td>
<td>$\log^2(n)/\varepsilon$</td>
</tr>
<tr>
<td>Agarwal, Cormode, Huang, Phillips, Wei, Yi</td>
<td>✓</td>
<td>✓</td>
<td>$\log^{3/2}(1/\varepsilon)/\varepsilon$</td>
</tr>
<tr>
<td>Karnin, Lang, Liberty</td>
<td>✓</td>
<td>✓</td>
<td>$\sqrt{\log(1/\varepsilon)/\varepsilon}$</td>
</tr>
<tr>
<td>Karnin, Lang, Liberty</td>
<td>✓</td>
<td>✗</td>
<td>$\log^2 \log(1/\varepsilon)/\varepsilon$</td>
</tr>
<tr>
<td>Karnin, Lang, Liberty</td>
<td>✗</td>
<td>✗</td>
<td>$\log \log(1/\varepsilon)/\varepsilon$</td>
</tr>
<tr>
<td>Still open...</td>
<td>✓</td>
<td>✓</td>
<td>$\log \log(1/\varepsilon)/\varepsilon$</td>
</tr>
</tbody>
</table>
The basic buffer idea

Buffer of size k

5 1 4 7 0 3
The basic buffer idea

Stores $k$ stream entries

```
3
0
7
4
1
5
```
The basic buffer idea

The buffer sorts $k$ stream entries

7
5
4
3
1
0
The basic buffer idea

Deletes every other item
The basic buffer idea

And outputs the rest with double the weight

5 3 0
The basic buffer idea

Let $R(x) = 2$ and $R'(x) = 2$ for $x = 0, 1, 3, 4, 5, 7$.

Let $R(x) = 5$ and $R'(x) = 6$ for $x = 0, 3, 5, 7$.

Let $R'(x) = 4$ for $x = 1, 4, 7$. 

Diagram:

- For $R(x) = 2$: 0, 1, 3, 4, 5, 7
- For $R'(x) = 2$: 0, 3, 4, 7
- For $R'(x) = 4$: 1, 4, 7
The basic buffer idea

Repeat $\frac{n}{k}$ time until the end of the stream

\[ |R'(x) - R(x)| < \frac{n}{k} \]
Manku-Rajagopalan-Lindsay (MRL) sketch

\[ \log_2(n) \] Buffers of size \( k \)

\[ |R'(x) - R(x)| \leq n \log_2(n)/k \]
Manku-Rajagopalan-Lindsay (MRL) sketch

If we set \( k = \frac{\log_2(n)}{\varepsilon} \)

We get \( |R'(x) - R(x)| \leq \varepsilon n \)

And we maintain only \( \frac{\log_2(n)}{\varepsilon} \) items from the stream!
Greenwald-Khanna (GK) sketch

Uses a completely different construction

It gets $|R'(x) - R(x)| \leq \varepsilon n$

And maintains only $O(\log(n)/\varepsilon)$ items from the stream!
Buffers of size $k \log(1/\varepsilon)$ start sampling after $O(1/\varepsilon^2)$ items. Reduces space usage to $\log^2(1/\varepsilon)/\varepsilon$ items from the stream.
Agarwal, Cormode, Huang, Phillips, Wei, Yi (2)

\[ E[R(x)] = \frac{R_0(x)}{R(x)} \]

\[ R'(x) = \begin{cases} 1 & \text{if } \frac{1}{R(x)} > \frac{1}{R_0(x)} \geq \frac{1}{R'(x)} \\ 2 & \text{if } \frac{1}{R'(x)} > \frac{1}{R_0(x)} \\ 0 & \text{if } \frac{1}{R_0(x)} > \frac{1}{R'(x)} \end{cases} \]

\[ R'(x) \] is a random variable now and \( E[R'(x)] = R(x) \)

Reduces space usage to \( \log^{3/2}(1/\varepsilon)/\varepsilon \) items from the stream.
Reduces space usage to \( \frac{\sqrt{\log(1/\varepsilon)}}{\varepsilon} \) items from the stream.
Reduces space usage to $\log \log(1/\varepsilon)/\varepsilon$ items from the stream.
Experimental Results

More experiments:  
https://datasketches.github.io/docs/Quantiles/KLLSketch.html  
https://datasketches.github.io/docs/Quantiles/KLLSketchVsTDigest.html
Even Newer Experimental Results

Ivking, Lang, Karnin, Liberty, Braverman, Streaming quantiles algorithms with small space and update time.
What else can we do in this model?

**Items**
(words, IP-addresses, events, clicks,...)

- Counting distinct elements
- Item frequencies
- Approximate Quantiles
- Moment and entropy estimation
- Approximate set operations
- Sampling

**Matrices**
(text corpora, recommendations, ...)

- Covariance estimation matrix
- Low rank approximation
- Sparsification

**Vectors**
(text documents, images, example features,...)

- Dimensionality reduction
- Clustering (k-means, k-median,...)
- Linear Regression
- Machine learning (some of it at least)
- Density Estimation / Anomaly detection

**Graphs**
(social networks, communications, ...)

- Connectivity
- Cut Sparsification
- Weighted Matching
Thank you!