Annotation in Data Streams

"with a little help from your friends"

Amit Chakrabarti *Dartmouth College*
Graham Cormode *AT&T Research Labs*
Andrew McGregor *University of Massachusetts, Amherst*

Data Stream Model

[Morris ’78] [Munro, Paterson ’78] [Flajolet, Martin ’85]
[Alon, Matias, Szegedy ’96] [Henzinger, Raghavan, Rajagopalan ’98]

Data Stream Model

[Morris '78] [Munro, Paterson '78] [Flajolet, Martin '85]
[Alon, Matias, Szegedy '96] [Henzinger, Raghavan, Rajagopalan '98]

- *Stream:* m elements from universe of size n

  e.g., $[x_1, x_2, \ldots, x_m] = 3,5,3,7,5,4,8,7,5,4,8,6,3,2,6,4,7, \ldots$

Data Stream Model

[Morris '78] [Munro, Paterson '78] [Flajolet, Martin '85]
[Alon, Matias, Szegedy '96] [Henzinger, Raghavan, Rajagopalan '98]

- *Stream:* m elements from universe of size n

  e.g., $[x_1, x_2, \ldots, x_m]$ = 3,5,3,7,5,4,8,7,5,4,8,6,3,2,6,4,7, ...

- *Goal:* Compute a function of stream, e.g., median, number of distinct elements, longest increasing sequence.

Data Stream Model

[Morris '78] [Munro, Paterson '78] [Flajolet, Martin '85]
[Alon, Matias, Szegedy '96] [Henzinger, Raghavan, Rajagopalan '98]

- *Stream:* m elements from universe of size n

  e.g., $[x_1, x_2, \ldots, x_m] = 3,5,3,7,5,4,8,7,5,4,8,6,3,2,6,4,7, \ldots$

- *Goal:* Compute a function of stream, e.g., median, number of distinct elements, longest increasing sequence.

- *The Catch:*

  i) Limited working memory, *i.e.*, sublinear(n,m)

  ii) Access data sequentially

  iii) Process each element quickly

Data Stream Model

[Morris '78] [Munro, Paterson '78] [Flajolet, Martin '85]
[Alon, Matias, Szegedy '96] [Henzinger, Raghavan, Rajagopalan '98]

- *Stream:* m elements from universe of size n

  e.g., $[x_1, x_2, \ldots, x_m] = 3,5,3,7,5,4,8,7,5,4,8,6,3,2,6,4,7, \ldots$

- *Goal:* Compute a function of stream, e.g., median, number of distinct elements, longest increasing sequence.

- *The Catch:*

  i) Limited working memory, *i.e.*, sublinear(n,m)

  ii) Access data sequentially

  iii) Process each element quickly

- Origins in '70s but has become popular in last ten years because of growing theory and very applicable.

Outsourcing Stream Processing

Outsourcing Stream Processing

- Many problems require linear space :(

Outsourcing Stream Processing

- Many problems require linear space :(
- *Off-load computation to more powerful "helper":*

  E.g., special hardware, multiple processing units, contractor who provides a commercial service.

Outsourcing Stream Processing

- Many problems require linear space :(
- *Off-load computation to more powerful "helper":*

  E.g., special hardware, multiple processing units, contractor who provides a commercial service.

- *Previous work in database community:*

  E.g., *stream punctuation* [Tucker et al. 05], *proof infused streams* [Li et al. 07], *stream outsourcing* [Yi et al. 08].

Outsourcing Stream Processing

- Many problems require linear space :(
- *Off-load computation to more powerful "helper":*

  E.g., special hardware, multiple processing units, contractor who provides a commercial service.

- *Previous work in database community:*

  E.g., *stream punctuation* [Tucker et al. 05], *proof infused streams* [Li et al. 07], *stream outsourcing* [Yi et al. 08].
- *Today's talk:* *What should model be and how powerful is it?*

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

12

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

3

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

18

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

103

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

37

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

56

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

43

Model Version 1: “Just trust the helper”

- ***Example Problem:*** Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

59

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

9

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

89

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

41

Model Version 1: “Just trust the helper”

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

51

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

56

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

26

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

26

* Numerical values not to actual scale.

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

43

* Numerical values not to actual scale.

Model Version 1: "Just trust the helper"

- *Example Problem:* Find median of m numbers from $[m]$. This requires $\Omega(m)$ space in the data stream model.

43

- Don't want to have to trust the third party.

* Numerical values not to actual scale.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

Model Version 2: “Helper is prescient”

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

43

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

12

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

3

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

18

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

103

- Helper first announces the answer: verification is easy.

Model Version 2: “Helper is prescient”

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

37

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

56

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

43

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

59

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

9

- Helper first announces the answer: verification is easy.

Model Version 2: “Helper is prescient”

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

89

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

41

- Helper first announces the answer: verification is easy.

Model Version 2: “Helper is prescient”

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

51

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

56

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

26

- Helper first announces the answer: verification is easy.

Model Version 2: "Helper is prescient"

- <u>*Example Problem:*</u> Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

26

- Helper first announces the answer: verification is easy.
- Can't expect the helper to know the future.

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

12

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

3

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

18

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

103

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

37

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

56

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

43

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

Model Version 3: “Helper is loquacious”

cf. “Best-Order Streaming” [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

9

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

89

Model Version 3: *“Helper is loquacious”*

cf. “Best-Order Streaming” [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

41

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- ***Example Problem:*** Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

51

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

56

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

26

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

26

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: “Helper is loquacious”

cf. “Best-Order Streaming” [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

3

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

9

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- <u>*Example Problem:*</u> Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

12

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: “Helper is loquacious”

cf. “Best-Order Streaming” [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

18

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

26

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

37

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

41

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- ***Example Problem:*** Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

43

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

51

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: “Helper is loquacious”

cf. “Best-Order Streaming” [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

56

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- ***Example Problem:*** Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

56

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

59

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

89

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: “Helper is loquacious”

cf. “Best-Order Streaming” [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

99

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

103

- Helper repeats stream in sorted order: Easy to find median

Model Version 3: "Helper is loquacious"

cf. "Best-Order Streaming" [Das Sarma, Lipton, Nanongkai 09]

- *Example Problem:* Find median of m numbers from [m]. This requires $\Omega(m)$ space in the data stream model.

103

- Helper repeats stream in sorted order: Easy to find median
- Fingerprint to check annotation B is rearranged stream A:

For prime $q \geq 3m$ and $r \in_R [q]$: check $FP_A(r)=FP_B(r)$ where

$$\mathrm{FP}_S(x) = \prod_{i \in S} (x - i) \bmod q$$

Final(ish) Model

Final(ish) Model

- ***Problem:*** Given stream S, want to compute f(S):

$$S=[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, \ldots, x_m]$$

Final(ish) Model

- ***Problem:*** Given stream S, want to compute f(S):

$$S=[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, \ldots, x_m]$$

- ***Helper:*** augments stream with "good" *h*-bit annotation:

$$(S,a)=[x_1, x_2, x_3, a_3, x_4, x_5, x_6, x_7, a_7, \ldots, x_m, a_m]$$

Final(ish) Model

- ***Problem:*** Given stream S, want to compute f(S):

$$S=[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, \ldots, x_m]$$

- ***Helper:*** augments stream with "good" *h*-bit annotation:

$$(S,a)=[x_1, x_2, x_3, a_3, x_4, x_5, x_6, x_7, a_7, \ldots, x_m, a_m]$$

annotation is a function
of previous elements

Final(ish) Model

- ***Problem:*** Given stream S, want to compute f(S):

$$S=[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, \ldots, x_m]$$

- ***Helper:*** augments stream with "good" *h*-bit annotation:

$$(S,a)=[x_1, x_2, x_3, a_3, x_4, x_5, x_6, x_7, a_7, \ldots, x_m, a_m]$$

annotation is a function of previous elements

- ***Verifier:*** using *v* bits of space and random string *r*, run verification algorithm to compute g(S,a,r) such that:

a) $\Pr_r[g(S,a,r)=f(S)] \geq 1-\delta$

b) $\Pr_r[g(S,a',r)=\perp] \geq 1-\delta$ for $a' \neq a$

Final(ish) Model

- *Problem:* Given stream S, want to compute f(S):

$$S=[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, \ldots , x_m]$$

- *Helper:* augments stream with "good" *h*-bit annotation:

$$(S,a)=[x_1, x_2, x_3, a_3, x_4, x_5, x_6, x_7, a_7, \ldots , x_m, a_m]$$

annotation is a function
of previous elements

- *Verifier:* using *v* bits of space and random string *r*, run verification algorithm to compute g(S,a,r) such that:

a) $\Pr_r[g(S,a,r)=f(S)] \geq 1\text{-}\delta$

b) $\Pr_r[g(S,a',r)=\perp] \geq 1\text{-}\delta$ for $a' \neq a$

- *Goal:* Minimize *h* and *v* for given error δ.

Better Median Protocol

- *Problem:* Find median of m numbers from $[m]$.
- *Thm:* $O(\sqrt{m} \log m)$ annotation bits and $O(\sqrt{m} \log m)$ verification memory is sufficient to find the median.

Upper Bound...

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

16th position

- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$
- Partition g into $v = m^{1/2}$ segments of length $h = m^{1/2}$

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$
- Partition g into $v = m^{1/2}$ segments of length $h = m^{1/2}$
- *Verifier:* *a)* Construct fingerprint of each segment

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

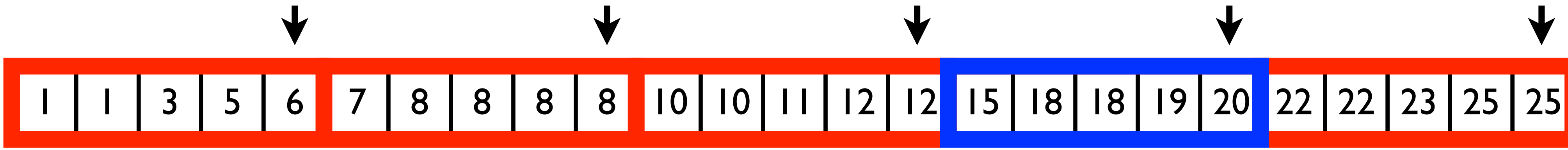
1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$
- Partition g into $v = m^{1/2}$ segments of length $h = m^{1/2}$
- <u>*Verifier:*</u> *a)* Construct fingerprint of each segment

  b) Compute last entry in each segment

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$



- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$
- Partition g into $v = m^{1/2}$ segments of length $h = m^{1/2}$
- ***Verifier:*** *a)* Construct fingerprint of each segment

  b) Compute last entry in each segment

  c) Identify "interesting" segment

Upper Bound...

- Define "cumulative frequency" vector: $g_i = |\{j : x_j \leq i\}|$

1	1	3	5	6	7	8	8	8	8	10	10	11	12	12	15	18	18	19	20	22	22	23	25	25

- Easy to see i is median iff $g_{i-1} < m/2$ and $g_i \geq m/2$
- Partition g into $v = m^{1/2}$ segments of length $h = m^{1/2}$
- <u>*Verifier:*</u> *a)* Construct fingerprint of each segment

  b) Compute last entry in each segment

  c) Identify "interesting" segment
- <u>*Helper:*</u> Presents entirety of interesting segment

Better Median Protocol

- *Problem:* Find median of m numbers from $[m]$.
- *Thm:* $O(\sqrt{m} \log m)$ annotation bits and $O(\sqrt{m} \log m)$ verification memory is sufficient to find the median.

Better Median Protocol

- *Problem:* Find median of m numbers from $[m]$.
- *Thm:* $O(\sqrt{m} \log m)$ annotation bits and $O(\sqrt{m} \log m)$ verification memory is sufficient to find the median.
- *Thm:* Any protocol for median requires

$$(\text{annotation "}h\text{"}) \times (\text{verification memory "}v\text{"}) = \Omega(m).$$

Lower Bound ...

Lower Bound ...

- Suppose algorithm "A" has parameters (h,v), error $\delta = 1/3$

Lower Bound ...

- Suppose algorithm "A" has parameters (h,v), error $\delta = 1/3$
- *Define "B":*

ALGORITHM "B"

1. Run $t = \Theta(h)$ copies of "A" in parallel
2. Use annotation a for all copies
3. Output majority answer if it exists, else $\perp$

- Algorithm "B" has parameters (h,hv), error $\delta = (1/3)2^{-h}$

If a valid, then expect $(2/3)t$ runs to return median

If a not valid, then expect $(2/3)t$ runs give $\perp$

Lower Bound ...

- Suppose algorithm "A" has parameters (h,v), error $\delta = 1/3$
- *Define "B":*

ALGORITHM "B"

1. Run $t = \Theta(h)$ copies of "A" in parallel
2. Use annotation *a* for all copies
3. Output majority answer if it exists, else $\perp$

- Algorithm "B" has parameters (h,hv), error $\delta = (1/3)2^{-h}$

  If *a* valid, then expect $(2/3)t$ runs to return median

  If *a* not valid, then expect $(2/3)t$ runs give $\perp$

- *Define "C":* Verifier ignores annotation, tries all 2^h possible annotations and ensures error at most 1/3 (by union bound).

Lower Bound ...

- Suppose algorithm "A" has parameters (h,v), error $\delta = 1/3$
- *Define "B":*

ALGORITHM "B"

1. Run $t = \Theta(h)$ copies of "A" in parallel
2. Use annotation a for all copies
3. Output majority answer if it exists, else $\perp$

- Algorithm "B" has parameters (h,hv), error $\delta = (1/3)2^{-h}$

    If a valid, then expect $(2/3)t$ runs to return median

    If a not valid, then expect $(2/3)t$ runs give $\perp$

- *Define "C":* Verifier ignores annotation, tries all 2^h possible annotations and ensures error at most 1/3 (by union bound).
- Algorithm "C" solves median of stream using $O(hv)$ space

Our Results

- *Median:* Optimal trade-off of annotation & verification.
- *Frequency Moments:* Optimal trade-off for exact version and lower bounds for approximate version.
- *Heavy Hitters:* Optimal trade-off for exact and protocol for approximate version via CM-sketch verification.
- *Graph Problems:* Trade-offs for counting triangles, matchings, and connectivity. Optimal in some regimes.

1. Frequency Moments
2. Counting Triangles
3. Beyond the Moraines...

1. Frequency Moments
2. Counting Triangles
3. Beyond the Moraines...

Frequency Moments

- ***Problem:*** Given m numbers from $[n]$, compute

$$F_k = \sum_{i \in [n]} f_i^k \quad \text{where } f_i = \text{freq. of } i$$

Frequency Moments

- ***Problem:*** Given m numbers from $[n]$, compute

$$F_k = \sum_{i \in [n]} f_i^k \quad \text{where } f_i = \text{freq. of } i$$

Frequency Moments

- ***Problem:*** Given m numbers from $[n]$, compute

$$F_k = \sum_{i \in [n]} f_i^k \quad \text{where } f_i = \text{freq. of } i$$

- ***Thm:*** $O(k^2\sqrt{n}\log m)$ annotation bits and $O(k\sqrt{n}\log m)$ verification memory is sufficient to compute F_k.

using "algebrization" ideas from [Aaronson, Widgerson 08]

Frequency Moments

- *Problem:* Given m numbers from $[n]$, compute

$$F_k = \sum_{i \in [n]} f_i^k \quad \text{where } f_i = \text{freq. of } i$$

- *Thm:* $O(k^2\sqrt{n} \log m)$ annotation bits and $O(k\sqrt{n} \log m)$ verification memory is sufficient to compute F_k.

using "algebrization" ideas from [Aaronson, Widgerson 08]

- *Thm:* Any protocol for F_k requires

(annotation) x (verification memory) = $\Omega(n)$

and any constant factor approx. requires

(annotation) x (verification memory) = $\Omega(n^{1-5/k})$.

using ideas from [Klauck 03], [Alon, Mattias, Szegedy 99]

Upper Bound (1/2)...

Upper Bound (1/2)...

- Transform universe [n] into [$\sqrt{}$n] x [$\sqrt{}$n] : $\sum_{i,j \in [\sqrt{n}]} f_{i,j}^k$

Upper Bound (1/2)...

- Transform universe [n] into [√n] x [√n] : $\sum_{i,j \in [\sqrt{n}]} f_{i,j}^k$

1	0	3	2	1	10	9	8	3

"frequency vector"

$\longrightarrow$

1	0	3
2	1	10
9	8	3

"frequency square"

Upper Bound (1/2)...

- Transform universe [n] into [$\sqrt{}$n] x [$\sqrt{}$n] : $\sum_{i,j \in [\sqrt{n}]} f_{i,j}^k$

1	0	3	2	1	10	9	8	3

"frequency vector"

$\longrightarrow$

1	0	3
2	1	10
9	8	3

"frequency square"

- Define $f(x,y) \in \mathbb{F}_q[x,y]$ where q large prime:

$$f(x,y) = \sum_{(i,j)\in S} p_{i,j}(x,y) \quad \text{where} \quad p_{i,j}(x,y) = \prod_{\ell=1:\ell\neq i}^{\sqrt{n}} \frac{x-\ell}{i-\ell} \cdot \prod_{\ell=1:\ell\neq j}^{\sqrt{n}} \frac{y-\ell}{j-\ell}$$

Upper Bound (1/2)...

- Transform universe [n] into [√n] x [√n] : $\sum_{i,j \in [\sqrt{n}]} f_{i,j}^k$

1	0	3	2	1	10	9	8	3

"frequency vector"

$\longrightarrow$

1	0	3
2	1	10
9	8	3

"frequency square"

- Define $f(x,y) \in \mathbb{F}_q[x,y]$ where q large prime:

$$f(x,y) = \sum_{(i,j) \in S} p_{i,j}(x,y) \quad \text{where} \quad p_{i,j}(x,y) = \prod_{\ell=1:\ell \neq i}^{\sqrt{n}} \frac{x-\ell}{i-\ell} \cdot \prod_{\ell=1:\ell \neq j}^{\sqrt{n}} \frac{y-\ell}{j-\ell}$$

$$f(i,j) = \text{ frequency of } (i,j) \text{ in stream}$$

Upper Bound (1/2)...

- Transform universe [n] into [√n] x [√n] : $\sum_{i,j \in [\sqrt{n}]} f_{i,j}^k$

1	0	3	2	1	10	9	8	3

"frequency vector"

$\longrightarrow$

1	0	3
2	1	10
9	8	3

"frequency square"

- Define $f(x,y) \in \mathbb{F}_q[x,y]$ where q large prime:

$$f(x,y) = \sum_{(i,j)\in S} p_{i,j}(x,y) \quad \text{where} \quad p_{i,j}(x,y) = \prod_{\ell=1:\ell\neq i}^{\sqrt{n}} \frac{x-\ell}{i-\ell} \cdot \prod_{\ell=1:\ell\neq j}^{\sqrt{n}} \frac{y-\ell}{j-\ell}$$

$$f(i,j) = \text{ frequency of } (i,j) \text{ in stream}$$

- <u>*Observe:*</u> *f* defined incrementally and:

$$\deg_x(f) = \deg_y(f) = \sqrt{n} - 1$$

Upper Bound (2/2)...

Upper Bound (2/2)...

- *Verifier:* pick $r \in_R [q]$; compute $f(r,y)$; and determine

$$C = \sum_{j \in [\sqrt{n}]} f(r,j)^k$$

Upper Bound (2/2)...

- *Verifier:* pick $r \in_R [q]$; compute $f(r,y)$; and determine

$$C = \sum_{j \in [\sqrt{n}]} f(r,j)^k$$

- *Helper:* annotates $O(\sqrt{n} \log q)$ bits with

$$s(x) = \sum_{j \in [\sqrt{n}]} f(x,j)^k$$

Upper Bound (2/2)...

- *Verifier:* pick $r \in_R [q]$; compute $f(r,y)$; and determine

$$C = \sum_{j \in [\sqrt{n}]} f(r,j)^k$$

- *Helper:* annotates $O(\sqrt{n} \log q)$ bits with

$$s(x) = \sum_{j \in [\sqrt{n}]} f(x,j)^k$$

- *Verifier:* checks $C = s(r)$ and outputs:

$$\sum_{i \in [\sqrt{n}]} s(i) = F_k$$

Lower Bound...

Lower Bound...

- <u>*Thm:*</u> Any protocol for F_k requires

  $$(\textit{annotation}) \times (\textit{verification memory}) = \Omega(n).$$

- <u>*Idea:*</u> Use MA lower bound for 2-party set-disjointness.

Lower Bound...

- *Thm:* Any protocol for F_k requires

  $$(annotation) \times (verification\ memory) = \Omega(n).$$

- *Idea:* Use MA lower bound for 2-party set-disjointness.

- *Thm:* Any constant factor approx. for F_k requires

  $$(annotation) \times (verification\ memory) = \Omega(n^{1-5/k}).$$

- *Idea:* *New* MA lower bound for *t*-party set-disjointness.

1. Frequency Moments
2. **Counting Triangles**
3. Beyond the Moraines...

Counting Triangles

Counting Triangles

- ***Problem:*** Given m edges on n nodes, find # triangles.

Counting Triangles

- ***Problem:*** Given m edges on n nodes, find # triangles.
- ***Thm:*** $O(n^{3\alpha} \log n)$ annotation bits and $O(n^{3-3\alpha} \log n)$ verification memory suffices for $0<\alpha<1$.

using idea from [Bar-Yossef, Kumar, Sivakumar 02]

Counting Triangles

- *Problem:* Given m edges on n nodes, find # triangles.
- *Thm:* $O(n^{3\alpha} \log n)$ annotation bits and $O(n^{3-3\alpha} \log n)$ verification memory suffices for $0<\alpha<1$.

using idea from [Bar-Yossef, Kumar, Sivakumar 02]

Construct induced stream S by replacing each (i,j) by (i,j,k) for each $k \neq i,j$. Compute $(F_3(S)-2F_2(S)+F_1(S))/12$.

Counting Triangles

- ***Problem:*** Given *m* edges on *n* nodes, find # triangles.
- ***Thm:*** $O(n^{3\alpha} \log n)$ annotation bits and $O(n^{3-3\alpha} \log n)$ verification memory suffices for $0<\alpha<1$.

  using idea from [Bar-Yossef, Kumar, Sivakumar 02]

  Construct induced stream S by replacing each (i,j) by (i,j,k) for each $k\neq i,j$. Compute $(F_3(S)-2F_2(S)+F_1(S))/12$.

- ***Thm:*** $O(n^2 \log n)$ annotation bits and $O(\log n)$ verification memory suffices.

Counting Triangles

- *Problem:* Given m edges on n nodes, find # triangles.
- *Thm:* $O(n^{3\alpha} \log n)$ annotation bits and $O(n^{3-3\alpha} \log n)$ verification memory suffices for $0<\alpha<1$.

  using idea from [Bar-Yossef, Kumar, Sivakumar 02]

  Construct induced stream S by replacing each (i,j) by (i,j,k) for each $k \neq i,j$. Compute $(F_3(S)-2F_2(S)+F_1(S))/12$.

- *Thm:* $O(n^2 \log n)$ annotation bits and $O(\log n)$ verification memory suffices.

  A is adjacency matrix. Prover give $(A[u,v], A^2[u,v])$ for each u,v. Verify A by finger-printing and verify A^2 by picking random s,r and checking $(sA)(Ar^T)= sA^2r^T$.

Counting Triangles

- *Problem:* Given *m* edges on *n* nodes, find # triangles.
- *Thm:* $O(n^{3\alpha} \log n)$ annotation bits and $O(n^{3-3\alpha} \log n)$ verification memory suffices for $0<\alpha<1$.

  using idea from [Bar-Yossef, Kumar, Sivakumar 02]

  Construct induced stream S by replacing each (i,j) by (i,j,k) for each $k \neq i,j$. Compute $(F_3(S)-2F_2(S)+F_1(S))/12$.

- *Thm:* $O(n^2 \log n)$ annotation bits and $O(\log n)$ verification memory suffices.

  A is adjacency matrix. Prover give $(A[u,v], A^2[u,v])$ for each u,v. Verify A by finger-printing and verify A^2 by picking random s,r and checking $(sA)(Ar^T) = sA^2r^T$.

- *Thm:* *Triangles requires (annotation) x (verification)=$\Omega(n^2)$.*

1. Frequency Moments
2. Counting Triangles
3. **Beyond the Moraines...**

Interactive Proofs & Streams

Forthcoming work from [Cormode, Yi '09]...

Interactive Proofs & Streams

Forthcoming work from [Cormode, Yi '09]...

- *Motivated by applications clouding computing:*

1. Consider messages back and forth between prover and verifier after the stream has been observed.
2. Measure in terms of memory used by verifier and subsequent communication.

Interactive Proofs & Streams

Forthcoming work from [Cormode, Yi '09]...

- *Motivated by applications clouding computing:*

1. Consider messages back and forth between prover and verifier after the stream has been observed.
2. Measure in terms of memory used by verifier and subsequent communication.

- *Results:*

1. Using log memory and log communication, many database problems can be solved exactly: *self join size, frequency moments, range queries, index* etc.

Sketch Verification

Sketch Verification

- *Sketch:* Let A be k by n measurement matrix and compute Af: for stream [1, 2, 3, 2, 4,...],

  $Af = a_1+a_2+a_3+a_2+a_4+\ldots$ where a_i is i[th] column of A

Sketch Verification

- *Sketch:* Let A be k by n measurement matrix and compute Af: for stream [1, 2, 3, 2, 4,...],

  $Af = a_1+a_2+a_3+a_2+a_4+\ldots$ where a_i is ith column of A

- *Annotation Protocol:* Helper gives Af in O(k log n) bits and verifier checks in O(log k) bits by composing fingerprint with A. Let B be k' by k and compute

  $BAf = Ba_1+Ba_2+Ba_3+Ba_2+Ba_4 + \ldots$

Sketch Verification

- ***Sketch:*** Let A be k by n measurement matrix and compute Af: for stream [1, 2, 3, 2, 4,...],

  $$Af = a_1 + a_2 + a_3 + a_2 + a_4 + \ldots$$ where a_i is i[th] column of A

- ***Annotation Protocol:*** Helper gives Af in O(k log n) bits and verifier checks in O(log k) bits by composing fingerprint with A. Let B be k' by k and compute

  $$BAf = Ba_1 + Ba_2 + Ba_3 + Ba_2 + Ba_4 + \ldots$$

- ***Tricky Part:*** Having helper guide verifier through the steps of extracting information from sketch.

Sketch Verification

- *Sketch:* Let A be k by n measurement matrix and compute Af: for stream [1, 2, 3, 2, 4,...],

  $Af = a_1+a_2+a_3+a_2+a_4+\ldots$ where a_i is i^{th} column of A

- *Annotation Protocol:* Helper gives Af in $O(k \log n)$ bits and verifier checks in $O(\log k)$ bits by composing fingerprint with A. Let B be k' by k and compute

  $BAf = Ba_1+Ba_2+Ba_3+Ba_2+Ba_4 + \ldots$

- *Tricky Part:* Having helper guide verifier through the steps of extracting information from sketch.

- *Mysterious Issue:* Most useful sketches are random but can't trust helper to pick random bits.

Summary

Model: Merlin-Arthur communication meets data streams and intractable stream problems become solvable!

Results: Annotation/verification trade-offs for classical stream problems. Some optimal and some open questions.

Thanks!