# Streaming Partitioning of Sequences and Trees

ICDT 2016

Christian Konrad

Reykjavik University

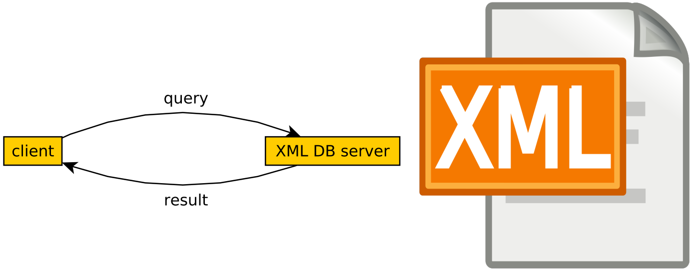
15.03.2016

# Outline

# Motivation

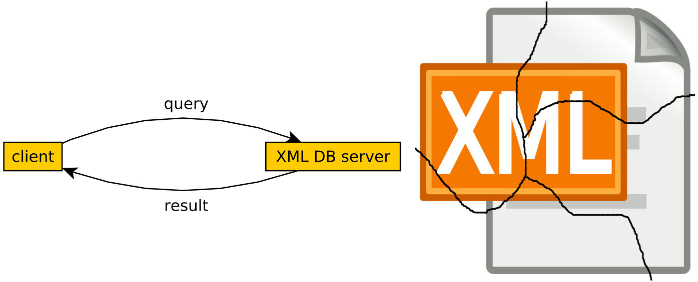
# XML Queries

**Querying massive XML Databases**



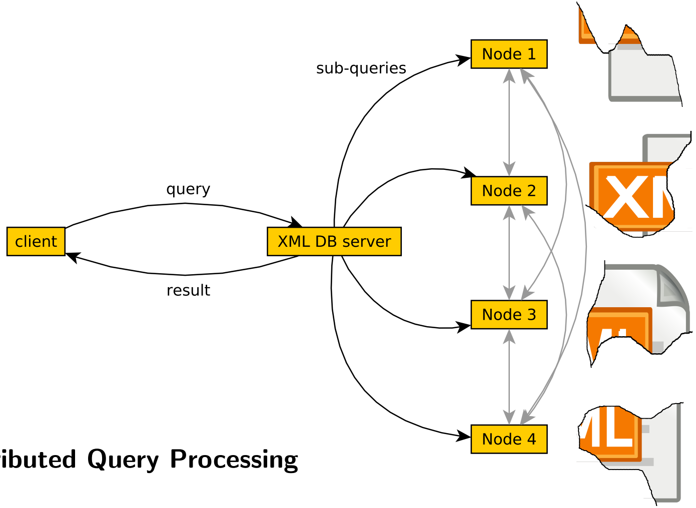
# XML Queries

**Querying massive XML Databases**



# XML Queries

**Querying massive XML Databases**




**Distributed Query Processing**

# XML Fragmentation

**How to fragment XML Documents?**

- Structured (taking XML schema into account)
- Ad-hoc
- Survey: **[Braganholo, Mattoso, SIGMOD 2014]**

Important: Fragments are of similar sizes for good load balancing

**Algorithmic Perspective**

Challanging if XML documents are massive

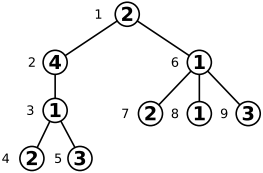
**Objective of this Work**

- Develop space efficient streaming algorithms for fragmenting XML documents
- Focus on load balancing aspect

# Problem Definitions

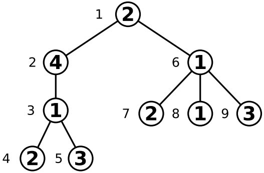
# Partitioning Trees

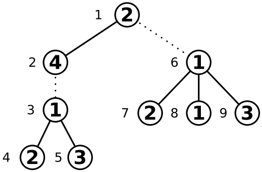
**Partitioning Trees:** Remove $p-1$ edges from a node-weighted tree s.t. maximum weight of the resulting subtrees is minimized



# Partitioning Trees

**Partitioning Trees:** Remove $p-1$ edges from a node-weighted tree s.t. maximum weight of the resulting subtrees is minimized




- $n$: number of nodes of input tree ($n = 9$)
- $p$: number of partitions to be created ($p = 3$)
- $B$: Bottleneck value, weight of heaviest subtree ($B = 7$)
- $B^*$: Bottleneck value of optimal partitioning ($B^* = 7$)

# Partitioning Integer Sequences

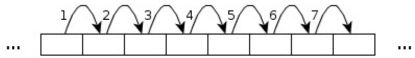
**Partitioning Integer Sequences:** Split sequence $X = X[1] \dots X[n]$ into $p$ blocks such that maximum weight of a block is minimized

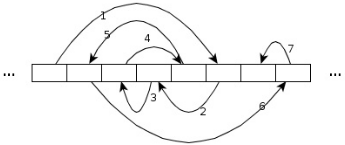
$$X = \underbrace{5 \quad 6 \quad 11 \quad 2 \quad 9}_{\sum=33} \quad | \quad \underbrace{14 \quad 3 \quad 8 \quad 1}_{\sum=26} \quad | \quad \underbrace{11 \quad 22}_{\sum=33}$$

- $n$: length of sequence ($n = 11$)
- $p$: number of partitions to be created ($p = 3$)
- $B$: Bottleneck value, weight of heaviest partition ($B = 33$)
- $B^*$: Bottleneck value of optimal partitioning ($B^* = 33?$)

# Streaming

**sequential access** **random access**




## Streaming

- **Objective:** compute some function $f(x_1, \ldots, x_n)$ given only sequential access

  How much RAM is required for the computation of $f$?

- **Motivation:** massive data sets (too large for storage in RAM)
- **Streaming Complexity**
  - Number of passes $p$, usually $\in \mathrm{O}(1)$ *this talk: $p = 1, 2$*
  - Memory space $s \in o(n)$
  - Update-time $t$, usually $\in \mathrm{O}(1)$ (or $\mathrm{O}(\log n)$)

**Partitioning Sequences in the Streaming Model:**

- Input Stream: sequence $X = X_1 X_2 \dots X_n$
- Output: positions of partition separators

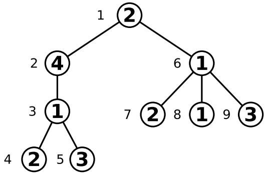
# Streaming Algorithms for Sequences and Trees

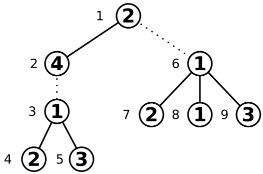
**Partitioning Sequences in the Streaming Model:**

- Input Stream: sequence $X = X_1 X_2 \dots X_n$
- Output: positions of partition separators

**Partitioning Trees in the Streaming Model:**

- Input Stream: depth-first-traversal of input tree



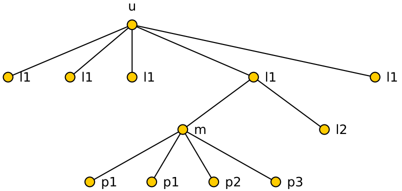




$2412\overline{2}3\overline{3}\overline{1}\overline{4}12\overline{2}1\overline{1}3\overline{3}\overline{1}\overline{2}$

- Output: IDs of root nodes of partitions $(1, 3, 6)$

# XML Document is a Depth-First-Traversal

```xml
<?xml version="1.0"?>
<university name="Reykjavik University">
   <lab name="DATALAB"></lab>
   <lab name="CADIA"></lab>
   <lab name="ICLT"></lab>
   <lab name="ICE-TCS">
      <members>
         <professor>Luca Aceto</professor>
         <professor>Magnus Halldorsson</professor>
         <postdoc>Ignacio Fabregas</postdoc>
         <phd-student>Christian Bean</phd-student>
      </members>
      <location>3rd floor, Mars</location>
   </lab>
   <lab name="ICE-ROSE"></lab>
</university>
```



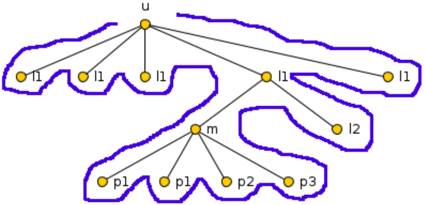

$$u l_1 \overline{l_1} l_1 \overline{l_1} l_1 \overline{l_1} l_1 m p_1 \overline{p_1} p_1 \overline{p_1} p_2 \overline{p_2} p_3 \overline{p_3}\,\overline{m} l_2 \overline{l_2}\,\overline{l_1} l_1 \overline{l_1}\,\overline{u}$$

**Depth-first traversal:**

- Opening tag $x$: down-step
- Closing tag $\overline{x}$: up-step

# XML Document is a Depth-First-Traversal

```xml
<?xml version="1.0"?>
<university name="Reykjavik University">
   <lab name="DATALAB"></lab>
   <lab name="CADIA"></lab>
   <lab name="ICLT"></lab>
   <lab name="ICE-TCS">
      <members>
         <professor>Luca Aceto</professor>
         <professor>Magnus Halldorsson</professor>
         <postdoc>Ignacio Fabregas</postdoc>
         <phd-student>Christian Bean</phd-student>
      </members>
      <location>3rd floor, Mars</location>
   </lab>
   <lab name="ICE-ROSE"></lab>
</university>
```




$$u l_1 \overline{l_1} l_1 \overline{l_1} l_1 \overline{l_1} l_1 m p_1 \overline{p_1} p_1 \overline{p_1} p_2 \overline{p_2} p_3 \overline{p_3} \overline{m} l_2 \overline{l_2} \overline{l_1} l_1 \overline{l_1} \overline{u}$$

**Depth-first traversal:**

- Opening tag $x$: down-step
- Closing tag $\overline{x}$: up-step

# Previous Work

# Previous Work: Partitioning Integer Sequences

**Dynamic Programming:**

| | | |
|---|---|---|
| Bokhari | 1988 | $O(n^3 p)$ |
| Anily & Federgruen | 1991 | $O(n^2 p)$ |
| Hansen & Liu | 1992 | $O(n^2 p)$ |

# Previous Work: Partitioning Integer Sequences

**Dynamic Programming:**

| | | |
|---|---|---|
| Bokhari | 1988 | $\mathrm{O}(n^3 p)$ |
| Anily & Federgruen | 1991 | $\mathrm{O}(n^2 p)$ |
| Hansen & Liu | 1992 | $\mathrm{O}(n^2 p)$ |

**Iterative Improvement:**

| | | |
|---|---|---|
| Manne & Sorevik | 1995 | $\mathrm{O}(np \log p)$ |
| Olstadt & Manne | 1995 | $\mathrm{O}(np)$ |

# Previous Work: Partitioning Integer Sequences

**Dynamic Programming:**

| | | |
|---|---|---|
| Bokhari | 1988 | $\mathrm{O}(n^3 p)$ |
| Anily & Federgruen | 1991 | $\mathrm{O}(n^2 p)$ |
| Hansen & Liu | 1992 | $\mathrm{O}(n^2 p)$ |

**Iterative Improvement:**

| | | |
|---|---|---|
| Manne & Sorevik | 1995 | $\mathrm{O}(np \log p)$ |
| Olstadt & Manne | 1995 | $\mathrm{O}(np)$ |

**Other Results:**

| | | |
|---|---|---|
| Nicol | 1991 | $\mathrm{O}(n + p^2 \log^2 n)$ |
| Charikar, Chekuri & Motwani | 1996 | $\mathrm{O}(n + p^2 \log^2 n)$ |
| Han, Narahari & Choi | 1992 | $\mathrm{O}(n + p^{1+\epsilon})$, for any $\epsilon > 0$ |

# Previous Work: Partitioning Integer Sequences

**Dynamic Programming:**

| | | |
|---|---|---|
| Bokhari | 1988 | $\mathrm{O}(n^3 p)$ |
| Anily & Federgruen | 1991 | $\mathrm{O}(n^2 p)$ |
| Hansen & Liu | 1992 | $\mathrm{O}(n^2 p)$ |

**Iterative Improvement:**

| | | |
|---|---|---|
| Manne & Sorevik | 1995 | $\mathrm{O}(np \log p)$ |
| Olstadt & Manne | 1995 | $\mathrm{O}(np)$ |

**Other Results:**

| | | |
|---|---|---|
| Nicol | 1991 | $\mathrm{O}(n + p^2 \log^2 n)$ |
| Charikar, Chekuri & Motwani | 1996 | $\mathrm{O}(n + p^2 \log^2 n)$ |
| Han, Narahari & Choi | 1992 | $\mathrm{O}(n + p^{1+\epsilon})$, for any $\epsilon > 0$ |

**Approach based on the Probe Algorithm:**

| | | |
|---|---|---|
| Frederickson | 1991 | $\mathrm{O}(n)$ |

# PROBE Algorithm

PROBE$(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE$(31)$

5 6 11 2 9 14 3 8 1 11 22

# PROBE Algorithm

PROBE$(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE$(31)$

5 6 11 2 9 14 3 8 1 11 22

5

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE(31)

5 6 11 2 9 14 3 8 1 11 22

11

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try $\text{PROBE}(31)$

5 6 11 2 9 14 3 8 1 11 22

22

# $\textsc{Probe}$ Algorithm

$\textsc{Probe}(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try $\textsc{Probe}(31)$

5 6 11 2 9 14 3 8 1 11 22

24

# $\textsc{Probe}$ Algorithm

$\textsc{Probe}(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try $\textsc{Probe}(31)$

$$\color{red}{5 \quad 6 \quad 11 \quad 2} \quad | \quad 9 \quad 14 \quad 3 \quad 8 \quad 1 \quad 11 \quad 22$$

$24 + 9 = 33 > 31$

# PROBE Algorithm

PROBE$(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE$(31)$

5 6 11 2 | 9 14 3 8 1 11 22

9

# PROBE Algorithm

PROBE$(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE$(31)$

$$5 \quad 6 \quad 11 \quad 2 \quad | \quad 9 \quad 14 \quad 3 \quad 8 \quad 1 \quad 11 \quad 22$$

23

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try $\text{PROBE}(31)$

$$5 \quad 6 \quad 11 \quad 2 \quad | \quad 9 \quad 14 \quad 3 \quad 8 \quad 1 \quad 11 \quad 22$$

26

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try $\textsc{Probe}(31)$

$$5 \quad 6 \quad 11 \quad 2 \quad | \quad 9 \quad 14 \quad 3 \quad | \quad 8 \quad 1 \quad 11 \quad 22$$

$26 + 8 = 34 > 31$

# PROBE Algorithm

PROBE$(B)$:

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE$(31)$

$$5 \quad 6 \quad 11 \quad 2 \quad | \quad 9 \quad 14 \quad 3 \quad | \quad 8 \quad 1 \quad 11 \quad 22$$

8

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try $\text{PROBE}(31)$

$$5 \quad 6 \quad 11 \quad 2 \quad | \quad 9 \quad 14 \quad 3 \quad | \quad 8 \quad 1 \quad 11 \quad 22$$

9

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE$(31)$

$$5 \quad 6 \quad 11 \quad 2 \quad | \quad 9 \quad 14 \quad 3 \quad | \quad 8 \quad 1 \quad 11 \quad 22$$

20

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE(31)

5 6 11 2 | 9 14 3 | 8 1 11 22

$20 + 22 = 42 > 31 \rightarrow$ **return** `false`.

Last partition larger than 31 $\rightarrow$ optimal bottleneck $B^* \geq 32$

# PROBE Algorithm

PROBE($B$):

- Traverse $X$ from left-to-right setting up maximal partitions so that partition weights do not exceed $B$
- Return `true` if successful, otherwise `false`

**Example:** $p = 3$, $\sum_i X_i = 92$, try PROBE(31)

5 6 11 2 | 9 14 3 | 8 1 11 22

$20 + 22 = 42 > 31 \rightarrow$ **return** `false`.

Trivial Bounds on $B^*$: $(m = \max X_i)$

$$1 \leq B^* \leq nm$$

Binary search: $\log mn$ calls to PROBE $\rightarrow \mathrm{O}(n \log(mn))$ algorithm

# Streaming Algorithms for Partitioning Integer Sequences

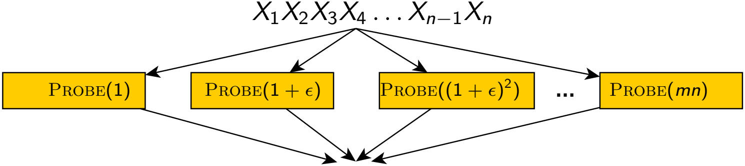
# Baseline Algorithm

**Observation:**

$\textsc{Probe}$ is a one-pass streaming alg. with $\mathrm{O}(p \log n + \log(mn))$ space

**One-pass Streaming Algorithm using Probe**

- Suppose $m, n$ are known in advance
- Then optimal bottleneck value $B^*$ is bounded: $1 \le B^* \le mn$
- Run $\textsc{Probe}(B)$ for $B = 1, (1+\epsilon), (1+\epsilon)^2, \ldots, mn$ in parallel




**Return:** Partitioning with smallest feasible value

$\rightarrow$ $(1+\epsilon)$-approximation using $\Theta(\log(mn)/\epsilon)$ copies of $\textsc{Probe}$

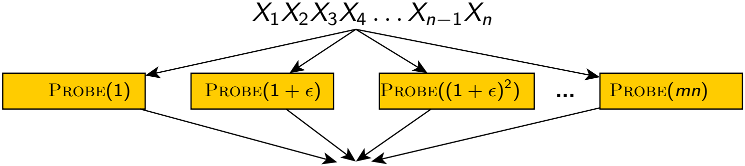
# Baseline Algorithm

**Observation:**

$\textsc{Probe}$ is a one-pass streaming alg. with $\mathrm{O}(p \log n + \log(mn))$ space

**One-pass Streaming Algorithm using Probe**

- Suppose $m, n$ are known in advance
- Then optimal bottleneck value $B^*$ is bounded: $1 \leq B^* \leq mn$
- Run $\textsc{Probe}(B)$ for $B = 1, (1+\epsilon), (1+\epsilon)^2, \ldots, mn$ in parallel




**Return:** Partitioning with smallest feasible value

$\rightarrow (1+\epsilon)$-approximation using $\cancel{\Theta(\log(mn)/\epsilon)}$ copies of $\textsc{Probe}$

$\Theta(\log p/\epsilon)$

# Our Results

**Algorithm:** One-pass $(1+\epsilon)$-approximation streaming algorithm with

1. $\mathrm{O}(\log(mn)p/\epsilon)$ space,
2. Optimal $\mathrm{O}(1)$ update-time.

**Lower Bounds:**

- $\Omega(n)$ is needed for exact algorithms
- $\Omega(\frac{1}{\epsilon}\log n)$ is needed for $(1+\epsilon)$-approximation

# New Technique: Coarsening

**Technique in Computer Science:**
Replace large (complicated) object by smaller (simpler) objects that capture important properties of inital object sufficiently well

E.g. Kernelization, Distance Oracles, Graph Sparsification, ...

# New Technique: Coarsening

**Technique in Computer Science:**
Replace large (complicated) object by smaller (simpler) objects that capture important properties of inital object sufficiently well

E.g. Kernelization, Distance Oracles, Graph Sparsification, ...

**Partitioning Sequences:** Coarse Version

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 | 1 |

$\downarrow$

| | | | | |
|---|---|---|---|---|
| | | | | |

# New Technique: Coarsening

**Technique in Computer Science:**
Replace large (complicated) object by smaller (simpler) objects that capture important properties of inital object sufficiently well

E.g. Kernelization, Distance Oracles, Graph Sparsification, . . .

**Partitioning Sequences:** Coarse Version

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 | 1 |

$\downarrow$

| | | | | |
|---|---|---|---|---|
| 9 | 15 | 10 | 15 | 11 |

1. Compute coarse version of smaller size

# New Technique: Coarsening

**Technique in Computer Science:**
Replace large (complicated) object by smaller (simpler) objects that capture important properties of inital object sufficiently well

E.g. Kernelization, Distance Oracles, Graph Sparsification, ...

**Partitioning Sequences:** Coarse Version

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 | 1 |

$\downarrow$

| | | | | |
|---|---|---|---|---|
| 9 | 15 | 10 | 15 | 11 |

1. Compute coarse version of smaller size
2. Partition coarse version exactly ($p = 2$)
3. Deduce partitioning of original version ($B = 34$, $B^* = 33$)

# Coarse Versions

**Definition:** $c$**-coarse Version**

X:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 | 1 |

↓

Y:

| | | | | |
|---|---|---|---|---|
| 9 | 15 | 10 | 15 | 11 |

| | | | | |
|---|---|---|---|---|
| (4, 5) | (8,7) | (2,9) | (7,8) | (5,6) |

(base, increment)

- Split elements of coarse version $Y$ into base and increment
- $c$-coarse version $\rightarrow$ maximal increment at most $c$ (here: 9-coarse)

**Lemma:** Let $B'$ be bottleneck value of opt. partitioning of $c$-coarse version $Y$. Then opt. partitioning of $X$ has bottleneck value $B^* + c \geq B'$.

- $\frac{S\epsilon}{p}$-coarse version suffices, ($S = \sum_i X_i$ total weight), since $B^* \geq S/p$
- Length of coarse version: $\mathrm{O}(p/\epsilon)$ independent of $n$!

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

$S$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

Mem:

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

| $S$: | 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

| Mem: | (4, 0) | (3, 0) | (2, 0) | (8, 0) | (7, 0) | (2, 0) | (1, 0) |
|---|---|---|---|---|---|---|---|

$\frac{S\epsilon}{p} = \frac{27 \cdot \frac{1}{2}}{2} = 6.75$

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

$S$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

Mem:

| | | | | | | |
|---|---|---|---|---|---|---|
| (4, 5) | | | (8, 0) | (7, 3) | | |

$\frac{S\epsilon}{p} = \frac{27 \cdot \frac{1}{2}}{2} = 6.75$

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

| $S$: | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

| Mem: | | | | | | | |
|---|---|---|---|---|---|---|---|
| | (4, 5) | (8, 0) | (7, 3) | | | | |

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

$S$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

Mem:

| | | | | | | |
|---|---|---|---|---|---|---|
| (4, 5) | (8, 0) | (7, 3) | (7, 0) | (7, 0) | (8, 0) | (5, 0) |

$\frac{S\epsilon}{p} = \frac{54 \cdot \frac{1}{2}}{2} = 13.5$

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

$S$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

Mem:

| | | | | | | |
|---|---|---|---|---|---|---|
| (4, 13) | | (7, 10) | | (7, 13) | | |

$\frac{S\epsilon}{p} = \frac{54 \cdot \frac{1}{2}}{2} = 13.5$

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

| $S$: | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

| Mem: | | | | | | | |
|---|---|---|---|---|---|---|---|
| | (4, 13) | (7, 10) | (7, 13) | | | | |

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

$S$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

Mem:

| | | | | | | |
|---|---|---|---|---|---|---|
| (4, 13) | (7, 10) | (7, 13) | (2, 0) | (3, 0) | | |

$\frac{S\epsilon}{p} = \frac{59 \cdot \frac{1}{2}}{2} = 14.75$

# Algorithm

**Example:** $p = 2, \epsilon = 1/2$ (i.e., compute a 1.5-approximation)

$S$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 8 | 7 | 2 | 1 | 7 | 7 | 8 | 5 | 2 | 3 |

**Algorithm:**

1. Fill memory with items from stream
2. Compress into $\frac{S\epsilon}{p}$-coarse version and repeat

Mem:

| | | | | | | |
|---|---|---|---|---|---|---|
| (4,13) | (7,10) | (7, 13) | (2, 3) | | | |

- Coarse version: 17 17 20 5
- Bottleneck value of resulting partitioning: $B = 34$
- Optimal bottleneck value: $B^* = 32$

# $\Omega(n)$ Lower Bound for Exact Algorithms

**Hard Communication Problem:** INDEX Problem

$$\textbf{Alice} \quad \xrightarrow{\text{M}} \quad \textbf{Bob} \quad \longrightarrow S[I]$$

$$S \in \{0,1\}^N \qquad\qquad I \in \{1,2,\dots,N\}$$

**Fact:** $|M| \in \Omega(n)$, for randomized protocols with bounded error

**Reduction:**

- Alice: $S = 0, 1, 0, 0, 1$ generates $X_1 = 13\,31\,13\,13\,31$
- Bob: $I = 4$ generates $X_2 = \underbrace{4 \dots 4}_{2I-N-1}\,2 = 4\,4\,2$
- Optimal split of $X_1 \circ X_2$ : $13\,31\,13\,13 \,|\, 31\,4\,4\,2$, no perfect split
- If $S[4] = 1$ then: $13\,31\,13\,3 \,|\, 1\,31\,4\,4\,2$, perfect split

# Algorithm: Summary

**Algorithm**

- Compute $(S\epsilon/p)$-coarse version of length $\mathrm{O}(p/\epsilon)$ in one pass
- **Post-processing:** Partition coarse version optimally and deduce $(1+\epsilon)$-partitioning of initial instance

**Properties of Algorithm**

- $\mathrm{O}(p \log(mn)/\epsilon)$ space
- Can be implemented with optimal $\mathrm{O}(1)$ update-time

**What is the correct space complexity?**

- $\Omega(n)$ for exact algorithms
- $\Omega(\log(n)/\epsilon)$ for $(1+\epsilon)$-approximations

# Streaming Algorithms for Partitioning Trees

# Coarse Version of Trees

**Structure Tree**

- Compute coarse structure tree consisting of $\mathrm{O}(p^2/\epsilon)$ nodes
- Pick subset of breakpoint nodes $U = \{u_1, u_2, \dots\}$ ordered w.r.t. a depth-first-traversal
- Let $L = \{\mathrm{lca}(u_i, u_{i+1}) \,:\, i\}$ be the set of lowest-common-ancestors of consecutive breakpoints
- Structure tree built on nodes $L \cup U$

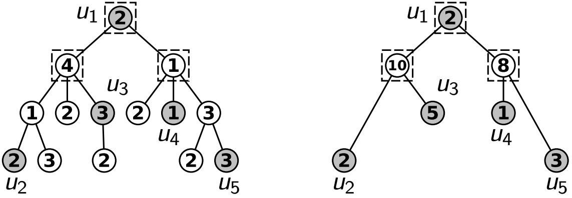
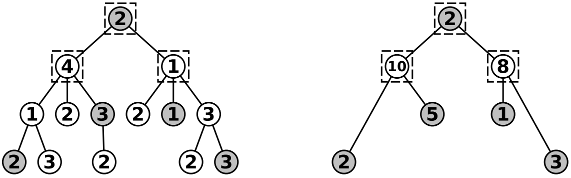



Figure: $U$: highlighted nodes. $L$: nodes within boxes.

## Breakpoints

- Compute coarse-version of sequence of down-steps $X'$ of depth-first-traversal $X$:



$$X' = 2\,4\,1\,2\,3\,2\,3\,2\,1\,2\,1\,3\,2\,3$$

- 5-coarse version of $X'$:

$$\begin{array}{c} 7\;7\;6\;7\;3 \\ \mathbf{2}\,4\,1\,|\,\mathbf{2}\,3\,2\,|\,\mathbf{3}\,2\,1\,2\,|\,\mathbf{1}\,3\,2\,|\,\mathbf{3} \end{array}$$

- Bold elements define $U$

$\rightarrow$ Reduction to Sequences

# Algorithm: Summary

**Algorithm**

- 2 passes required for computing structure tree
- **Post-processing:** Partition structure tree optimally and deduce $(1+\epsilon)$-approximate partitioning

**Properties of Algorithm**

- $\mathrm{O}(p^2 \log(mn)/\epsilon)$ space
- Two passes
- Can be implemented with optimal $\mathrm{O}(1)$ update-time

**Open Questions**

- Can space be reduced to $\mathrm{O}(p \log(mn)/\epsilon)$?
- One pass?

# Conclusion

**Conclusion**

- Modern applications provide new perspectives on old problems
- New insight: Coarsening

**Where to go from here?**

- XML documents: Partitioning respecting underlying structure
- Leightweight streaming algorithms for other partitioning problems?
- Prove space optimality

Thank You for Listening.
Questions?