Querying and Mining Data Streams: You Only Get One Look

A Tutorial

Minos Garofalakis[●] Johannes Gehrke[●]
Rajeev Rastogi[●]

●Bell Laboratories

●Cornell University

Outline

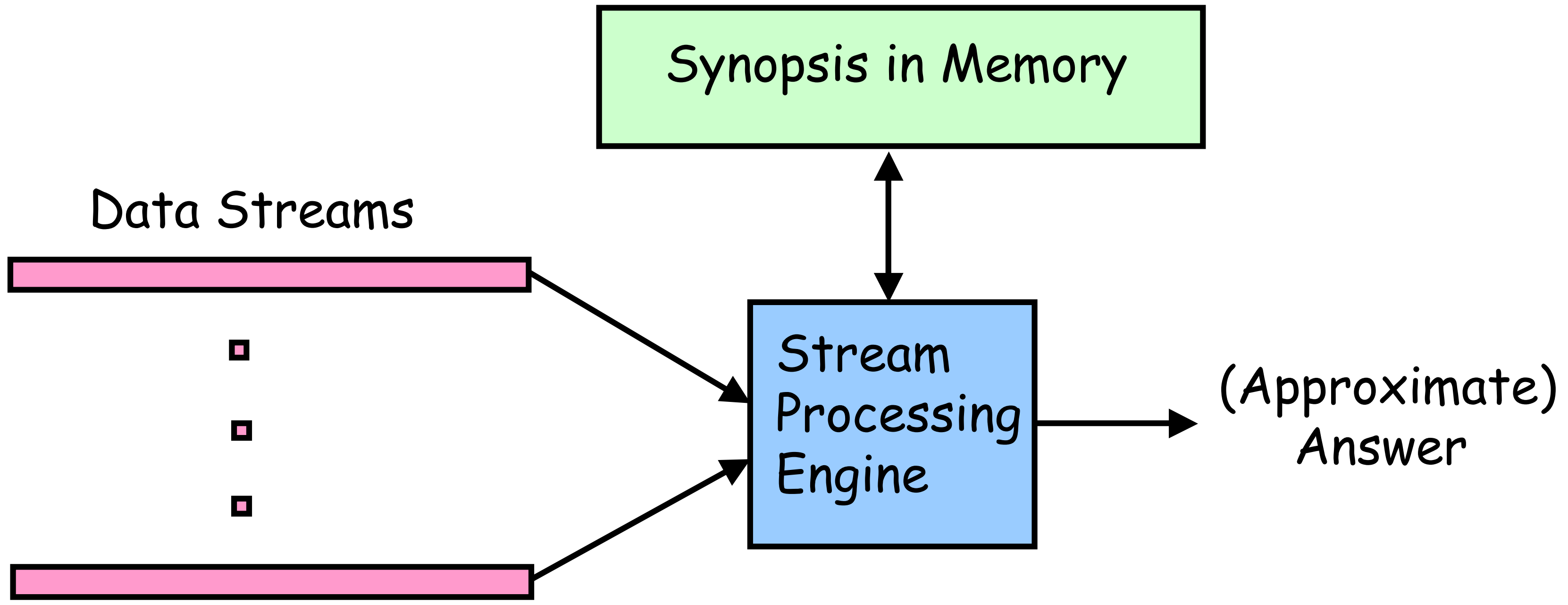
- Introduction & Motivation
  - Stream computation model, Applications
- Basic stream synopses computation
  - Samples, Equi-depth histograms, Wavelets
- Mining data streams
  - Decision trees, clustering, association rules
- Sketch-based computation techniques
  - Self-joins, Joins, Wavelets, V-optimal histograms
- Advanced techniques
  - Sliding windows, Distinct values, Hot lists
- Future directions & Conclusions

Processing Data Streams: Motivation

- A growing number of applications generate streams of data
  - Performance measurements in network monitoring and traffic management
  - Call detail records in telecommunications
  - Transactions in retail chains, ATM operations in banks
  - Log records generated by Web Servers
  - Sensor network data
- Application characteristics
  - Massive volumes of data (several terabytes)
  - Records arrive at a rapid rate
- Goal: Mine patterns, process queries and compute statistics on data streams in real-time

Data Streams: Computation Model

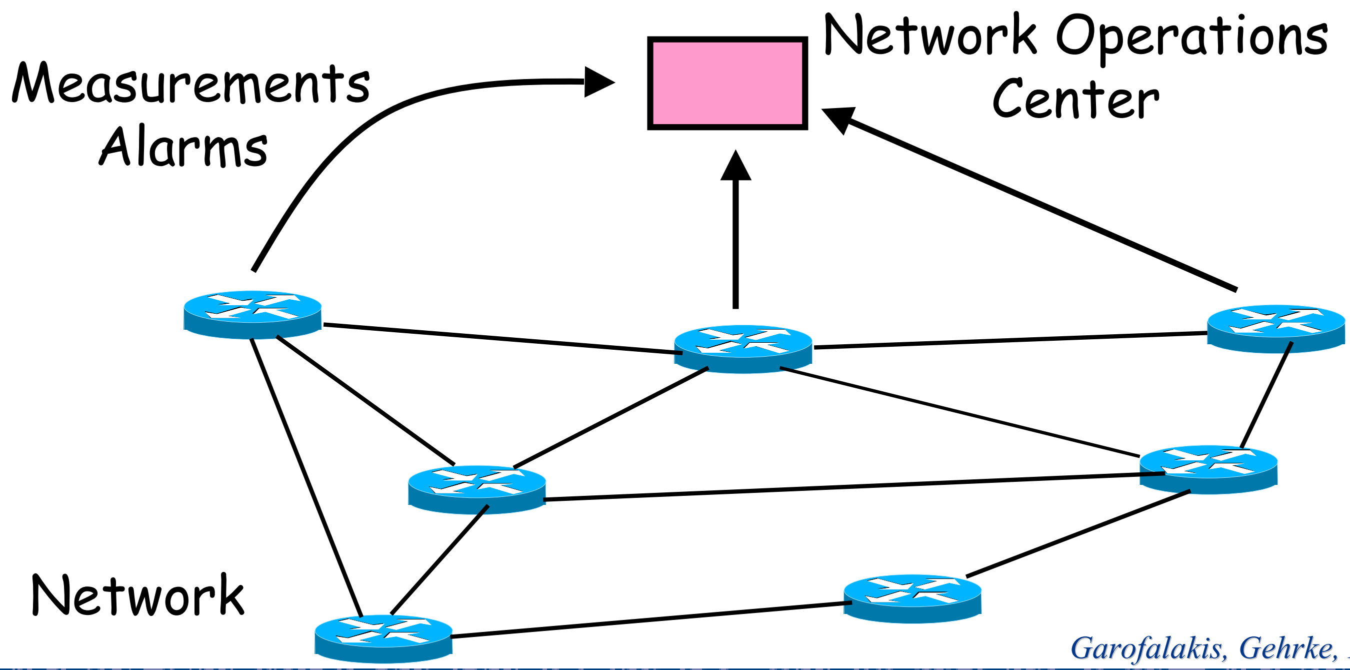
- A data stream is a (massive) sequence of elements: $e_1, \ldots, e_n$



- Stream processing requirements
  - Single pass: Each record is examined at most once
  - Bounded storage: Limited Memory (M) for storing synopsis
  - Real-time: Per record processing time (to maintain synopsis) must be low

Network Management Application

- Network Management involves monitoring and configuring network hardware and software to ensure smooth operation
  - Monitor link bandwidth usage, estimate traffic demands
  - Quickly detect faults, congestion and isolate root cause
  - Load balancing, improve utilization of network resources



IP Network Measurement Data

- IP session data (collected using Cisco NetFlow)

Source	Destination	Duration	Bytes	Protocol
10.1.0.2	16.2.3.7	12	20K	http
18.6.7.1	12.4.0.3	16	24K	http
13.9.4.3	11.6.8.2	15	20K	http
15.2.2.9	17.1.2.1	19	40K	http
12.4.3.8	14.8.7.4	26	58K	http
10.5.1.3	13.0.0.1	27	100K	ftp
11.1.0.6	10.3.4.5	32	300K	ftp
19.7.1.2	16.5.5.8	18	80K	ftp

- AT&T collects 100 GBs of NetFlow data each day!

Network Data Processing

- Traffic estimation
  - How many bytes were sent between a pair of IP addresses?
  - What fraction network IP addresses are active?
  - List the top 100 IP addresses in terms of traffic
- Traffic analysis
  - What is the average duration of an IP session?
  - What is the median of the number of bytes in each IP session?
- Fraud detection
  - List all sessions that transmitted more than 1000 bytes
  - Identify all sessions whose duration was more than twice the normal
- Security/Denial of Service
  - List all IP addresses that have witnessed a sudden spike in traffic
  - Identify IP addresses involved in more than 1000 sessions

Data Stream Processing Algorithms

- Generally, algorithms compute approximate answers
  - Difficult to compute answers accurately with limited memory
- Approximate answers - Deterministic bounds
  - Algorithms only compute an approximate answer, but bounds on error
- Approximate answers - Probabilistic bounds
  - Algorithms compute an approximate answer with high probability
    - With probability at least $1-\delta$, the computed answer is within a factor ε of the actual answer
- Single-pass algorithms for processing streams also applicable to (massive) terabyte databases!

Outline

- Introduction & Motivation
- Basic stream synopses computation
  - **Samples:** Answering queries using samples, Reservoir sampling
  - **Histograms:** Equi-depth histograms, On-line quantile computation
  - **Wavelets:** Haar-wavelet histogram construction & maintenance
- Mining data streams
- Sketch-based computation techniques
- Advanced techniques
- Future directions & Conclusions

Sampling: Basics

- Idea: A small random sample S of the data often well-represents all the data
  - For a fast approx answer, apply “modified” query to S
  - Example: select agg from R where R.e is odd

Data stream: 9 3 5 2 7 1 6 5 8 4 9 1 (n=12)

Sample S: 9 5 1 8

  - If agg is avg, return average of odd elements in S **answer: 5**
  - If agg is count, return average over all elements e in S of
    - n if e is odd
    - 0 if e is even

answer: 12*3/4 =9

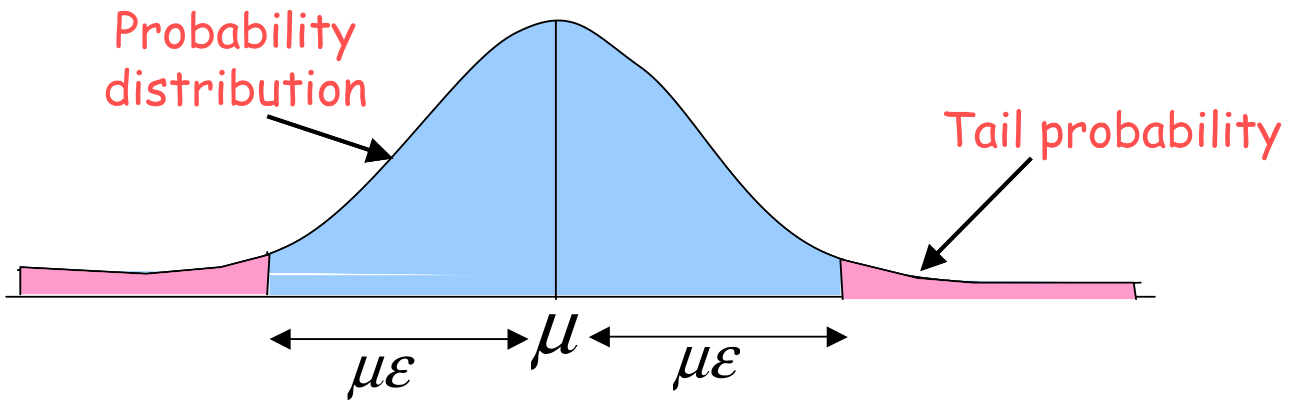
Unbiased: For expressions involving count, sum, avg: the estimator is unbiased, i.e., the expected value of the answer is the actual answer

Probabilistic Guarantees

- Example: Actual answer is within 5 ± 1 with prob ≥ 0.9
- Use Tail Inequalities to give probabilistic bounds on returned answer
  - Markov Inequality
  - Chebyshev's Inequality
  - Hoeffding's Inequality
  - Chernoff Bound

Tail Inequalities

- General bounds on *tail probability* of a random variable (that is, probability that a random variable deviates far from its expectation)



- Basic Inequalities: Let X be a random variable with expectation μ and variance Var[X]. Then for any $\varepsilon > 0$

Markov: $$\Pr(X \geq \varepsilon) \leq \frac{\mu}{\varepsilon}$$

Chebyshev: $$\Pr(|X-\mu| \geq \mu\varepsilon) \leq \frac{Var[X]}{\mu^2\varepsilon^2}$$

Tail Inequalities for Sums

- Possible to derive stronger bounds on tail probabilities for the sum of independent random variables
- Hoeffding's Inequality: Let X1, ..., Xm be independent random variables with 0<=Xi <= r. Let $\overline{X} = \frac{1}{m}\sum_i X_i$ and μ be the expectation of $\overline{X}$. Then, for any $\varepsilon > 0$,

$$\Pr(|\overline{X} - \mu| \geq \varepsilon) \leq 2\exp^{\frac{-2m\varepsilon^2}{r^2}}$$

- Application to avg queries:
  - m is size of subset of sample S satisfying predicate (3 in example)
  - r is range of element values in sample (8 in example)
- Application to count queries:
  - m is size of sample S (4 in example)
  - r is number of elements n in stream (12 in example)
- More details in [HHW97]

Tail Inequalities for Sums (Contd.)

- Possible to derive even stronger bounds on tail probabilities for the sum of independent *Bernoulli trials*
- Chernoff Bound: Let X1, ..., Xm be independent Bernoulli trials such that Pr[Xi=1] = p (Pr[Xi=0] = 1-p). Let $X = \sum_i X_i$ and $\mu = mp$ be the expectation of X. Then, for any $\varepsilon > 0$,

$$\Pr(|X - \mu| \geq \mu\varepsilon) \leq 2\exp^{\frac{-\mu\varepsilon^2}{2}}$$

- Application to count queries:
  - m is size of sample S (4 in example)
  - p is fraction of odd elements in stream (2/3 in example)
- Remark: Chernoff bound results in tighter bounds for count queries compared to Hoeffding's inequality

Computing Stream Sample

- Reservoir Sampling [Vit85]: Maintains a sample S of a fixed-size M
  - Add each new element to S with probability M/n, where n is the current number of stream elements
  - If add an element, evict a random element from S
  - Instead of flipping a coin for each element, determine the number of elements to skip before the next to be added to S
- Concise sampling [GM98]: Duplicates in sample S stored as <value, count> pairs (thus, potentially boosting actual sample size)
  - Add each new element to S with probability 1/T (simply increment count if element already in S)
  - If sample size exceeds M
    - Select new threshold T' > T
    - Evict each element (decrement count) from S with probability 1-T/T'
  - Add subsequent elements to S with probability 1/T'

Counting Samples [GM98]

- Effective for answering hot list queries (k most frequent values)
  - Sample S is a set of <value, count> pairs
  - For each new stream element
    - If element value in S, increment its count
    - Otherwise, add to S with probability 1/T
  - If size of sample S exceeds M, select new threshold T' > T
    - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented
      - First try, decrement count with probability 1- T/T'
      - Subsequent tries, decrement count with probability 1-1/T'
  - Subject each subsequent stream element to higher threshold T'

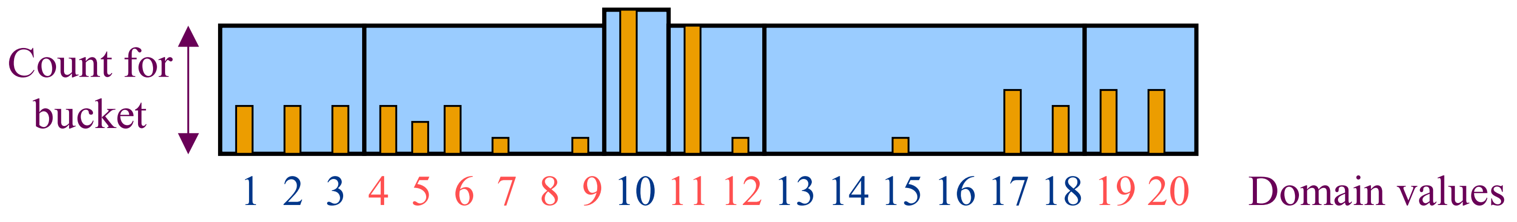
- Estimate of frequency for value in S: count in S + 0.418*T

Histograms

- Histograms approximate the frequency distribution of element values in a stream
- A histogram (typically) consists of
  - A partitioning of element domain values into buckets
  - A count C_B per bucket B (of the number of elements in B)
- Long history of use for selectivity estimation within a query optimizer [Koo80], [PSC84], etc.
- [PIH96] [Poo97] introduced a taxonomy, algorithms, etc.

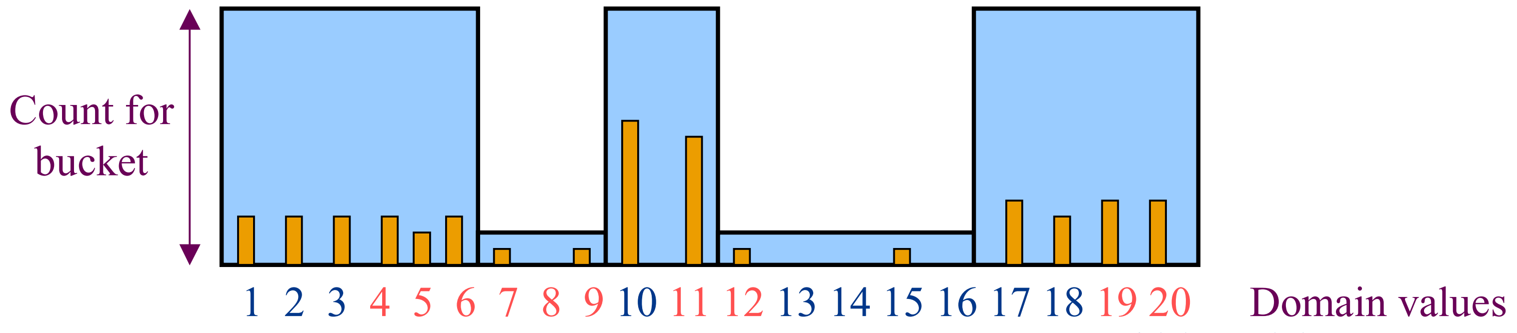
Types of Histograms

- Equi-Depth Histograms
  - Idea: Select buckets such that counts per bucket are equal



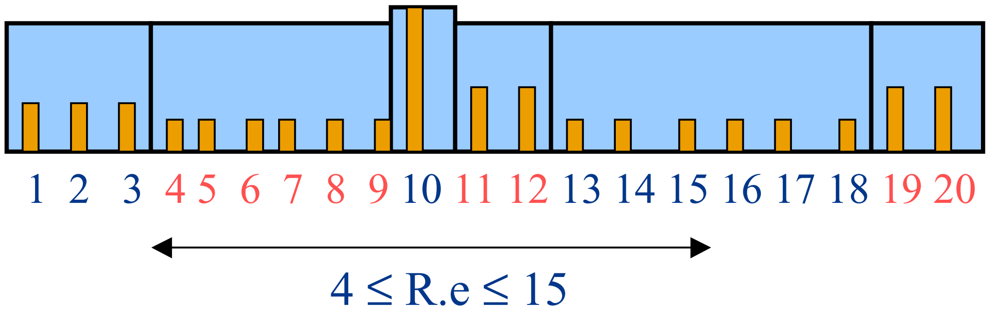
- V-Optimal Histograms [IP95] [JKM98]
  - Idea: Select buckets to minimize frequency variance within buckets

$$\text{minimize} \quad \sum_{B} \sum_{v \in B} \left(f_v - \frac{C_B}{V_B}\right)^2$$



Answering Queries using Histograms [IP99]

- (Implicitly) map the histogram back to an approximate relation, & apply the query to the approximate relation
- Example: select count(*) from R where 4 <= R.e <= 15



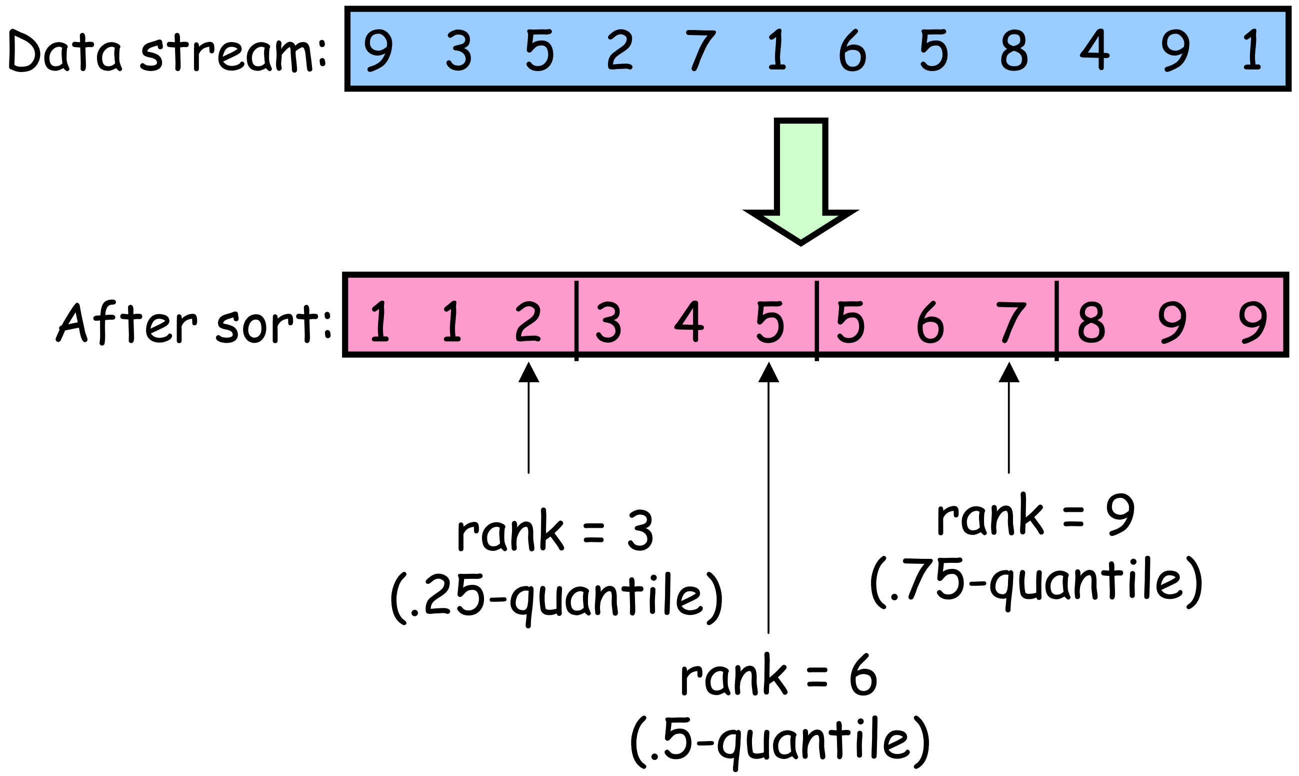
Count spread evenly among bucket values

answer: 3.5 * C_B

- For equi-depth histograms, maximum error: $\pm 2 * C_B$

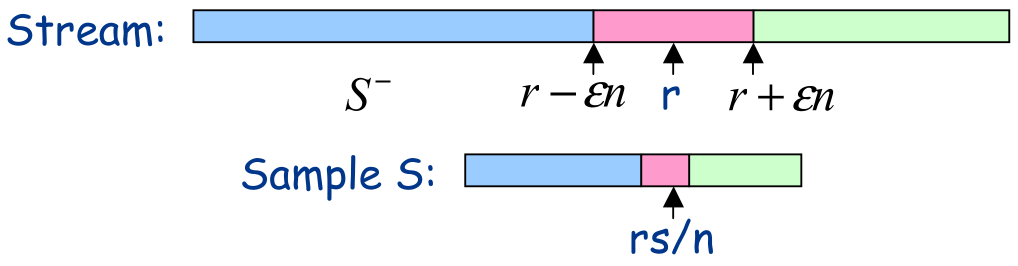
Equi-Depth Histogram Construction

- For histogram with b buckets, compute elements with rank n/b, 2n/b, ..., (b-1)n/b
- Example: (n=12, b=4)



Computing Approximate Quantiles Using Samples

- Problem: Compute element with rank r in stream
- Simple sampling-based algorithm
  - Sort sample S of stream and return element in position rs/n in sample (s is sample size)
  - With sample of size $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\delta}))$, possible to show that rank of returned element is in $[r-\varepsilon n, r+\varepsilon n]$ with probability at least $1-\delta$
    - Hoeffding's Inequality: probability that S contains greater than rs/n elements from S^- is no more than $\exp^{-2s\varepsilon^2}$



- [CMN98][GMP97] propose additional sampling-based methods

Algorithms for Computing Approximate Quantiles

- [MRL98],[MRL99],[GK01] propose sophisticated algorithms for computing stream element with rank in $[r-\varepsilon n, r+\varepsilon n]$
  - Space complexity proportional to $\frac{1}{\varepsilon}$ instead of $\frac{1}{\varepsilon^2}$

- [MRL98], [MRL99]
  - Probabilistic algorithm with space complexity $O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$
  - Combined with sampling, space complexity becomes $O(\frac{1}{\varepsilon}\log^2(\frac{1}{\varepsilon}\log(\frac{1}{\delta})))$

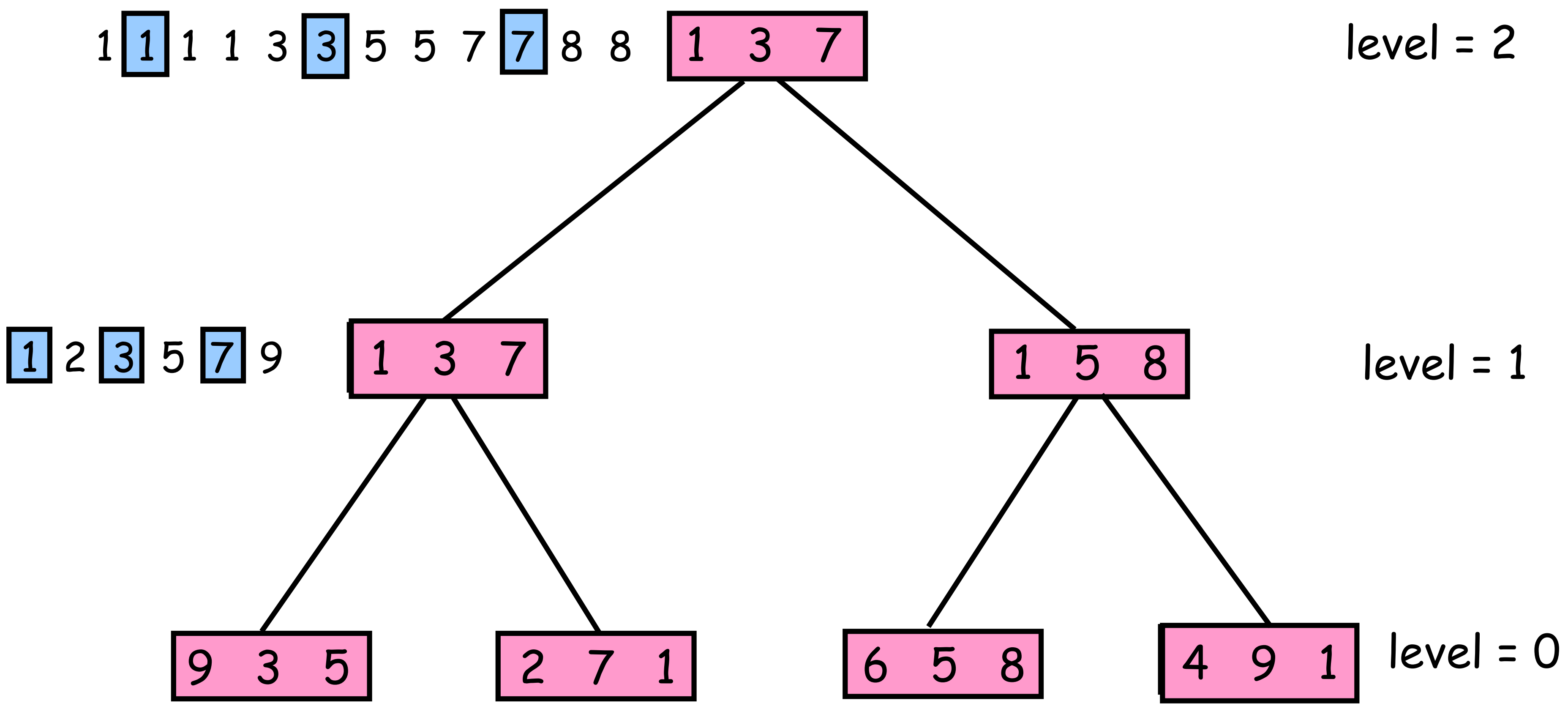
- [GK01]
  - *Deterministic algorithm* with space complexity $O(\frac{1}{\varepsilon}\log(\varepsilon n))$

Single-Pass Quantile Computation Algorithm [MRL 98]

- Split memory M into b buffers of size k (M = bk)
- For each successive set of k elements in stream
  - If free buffer B exists
    - insert k elements into B, set level of B to 0
  - Else
    - merge two buffers B and B' at same level l
    - output result of merge into B', set level of B' to l+1
    - insert k elements into B, set level of B to 0
- Output element in position r after making 2^l copies of each element in final buffer and sorting them
- Merge operation (input buffers B and B' at level l)
  - Make 2^l copies of each element in B and B'
  - Sort copies
  - Output elements in positions $j2^{l+1} + 2^l$ in sorted sequence, j=0, ..., k-1

Single-Pass Algorithm (Example)

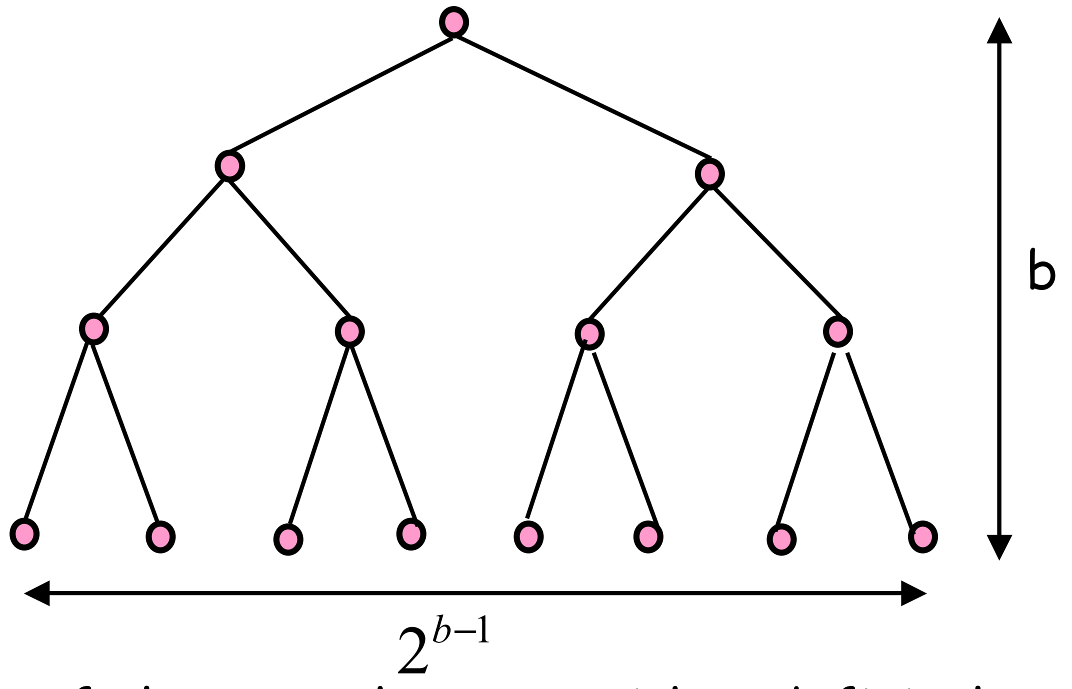
- M=9, b=3, k=3, r =10


1 $\boxed{1}$ 1 1 3 $\boxed{3}$ 5 5 7 $\boxed{7}$ 8 8
1 3 7
level = 2
$\boxed{1}$ 2 $\boxed{3}$ 5 $\boxed{7}$ 9
1 3 7
1 5 8
level = 1
9 3 5
2 7 1
6 5 8
4 9 1
level = 0

- Computed quantile (r=10)

1 1 1 1 3 3 3 3 7 $\boxed{7}$ 7 7

Analysis of Algorithm



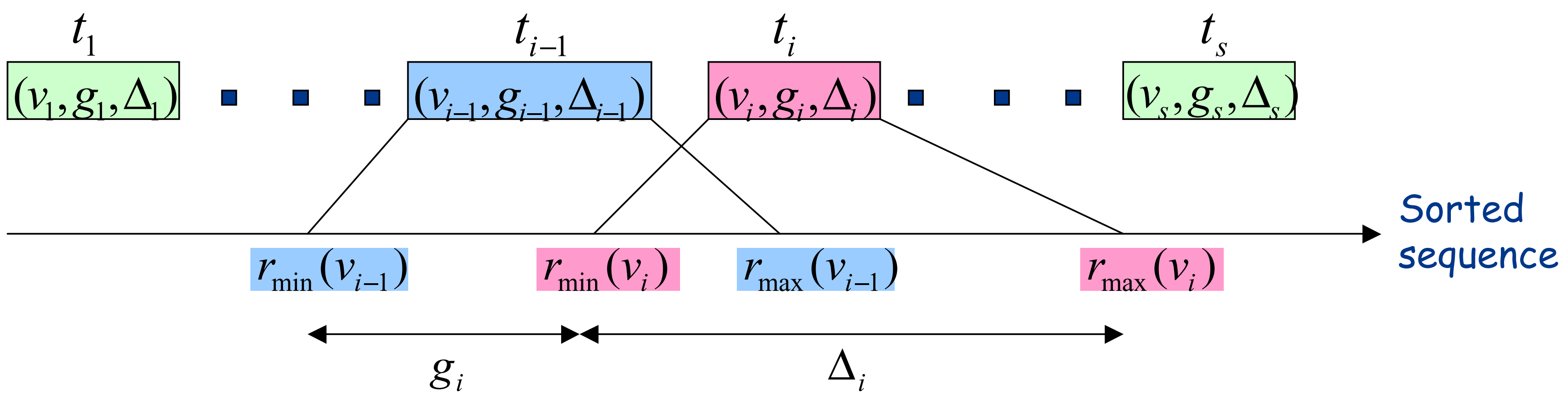
- Number of elements that are neither definitely small, nor definately large: $(b-2)2^{b-2}$
- Algorithm returns element with rank r', where

$$r-(b-2)2^{b-2} \leq r' \leq r+(b-2)2^{b-2}$$

- Choose smallest b such that $k2^{b-1} \geq n$ and bk = M

Computing Approximate Quantiles [GK01]

- Synopsis structure S: sequence of tuples $t_1, t_2, ..., t_s$



- $r_{\min}(v_i) / r_{\max}(v_i)$: min/max rank of v_i
- g_i: number of stream elements covered by t_i
- <u>Invariants:</u>

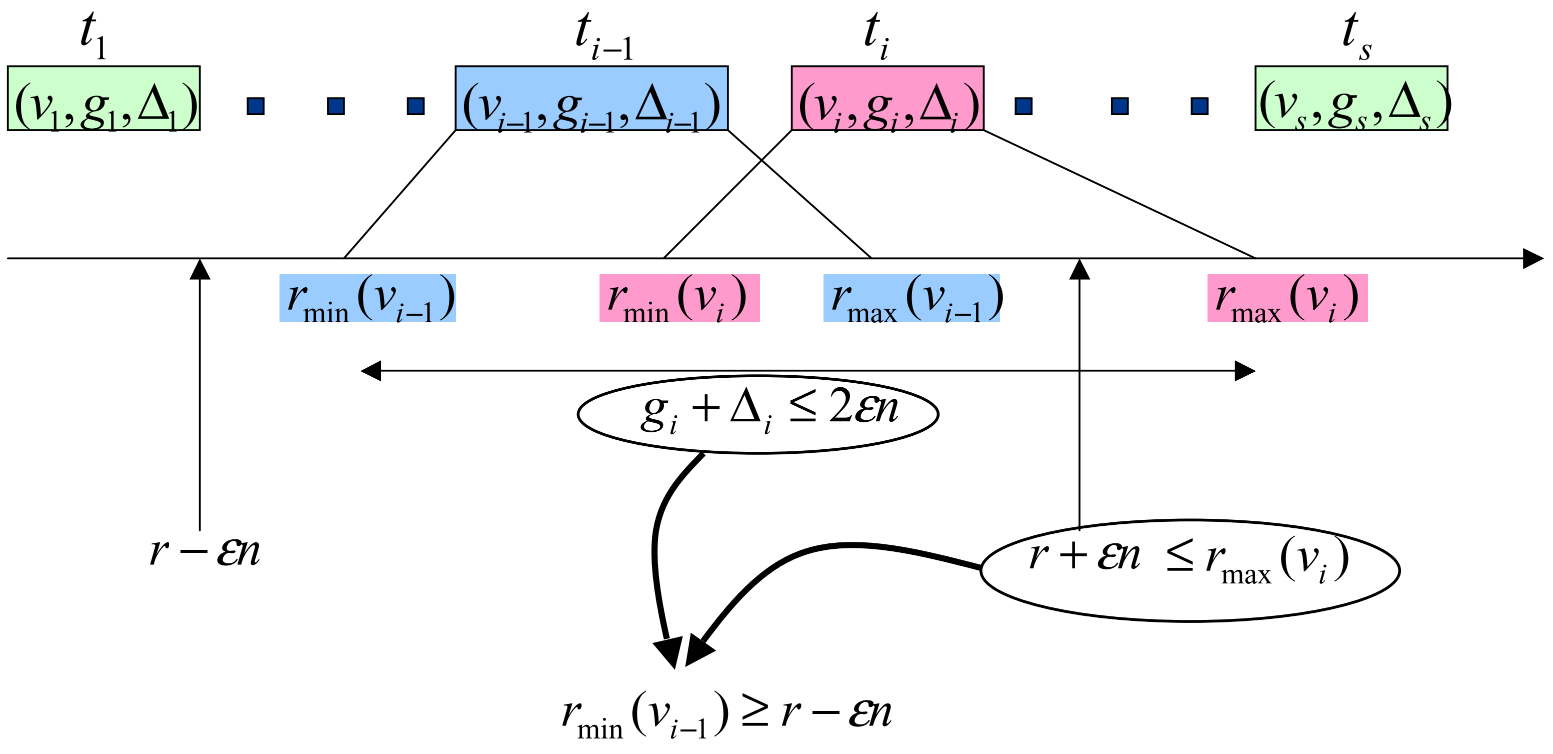
$$g_i + \Delta_i \le 2\varepsilon n$$

$$r_{\min}(v_i) = \sum_{j \le i} g_j, \qquad r_{\max}(v_i) = \sum_{j \le i} g_j + \Delta_i$$

Computing Quantile from Synopsis

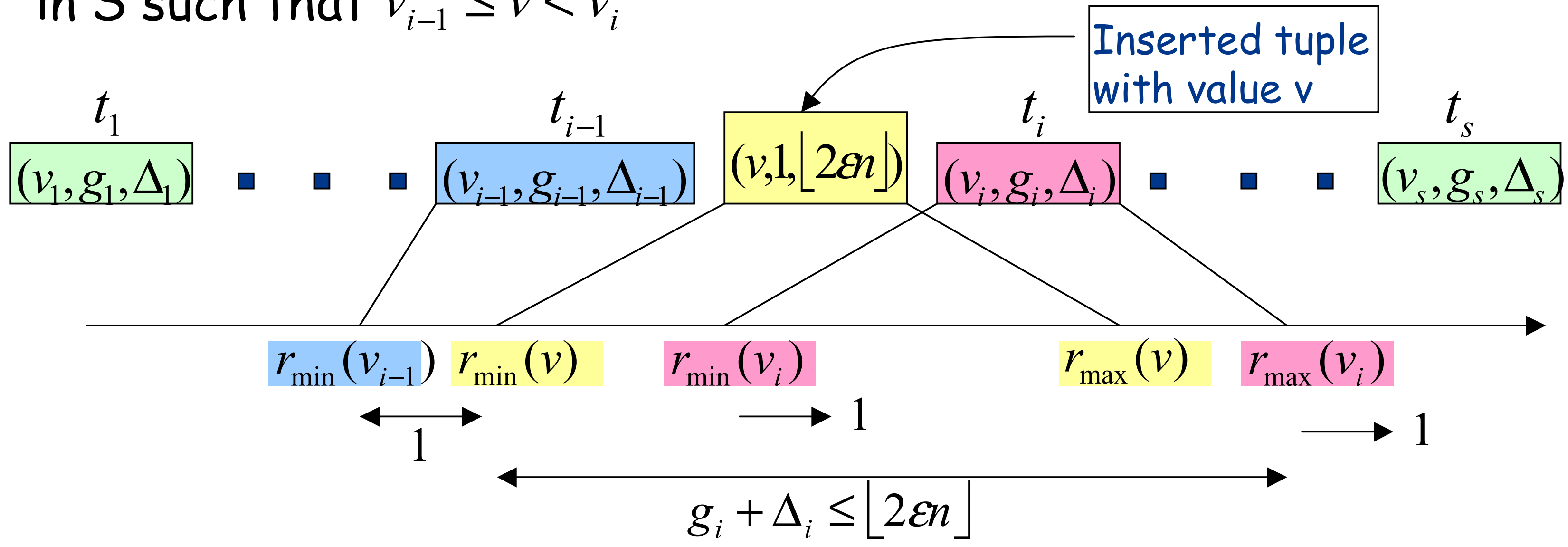
- Theorem: Let i be the max index such that $r_{\max}(v_{i-1}) \le r + \varepsilon n$. Then,

$$r - \varepsilon n \le \text{rank}(v_{i-1}) \le r + \varepsilon n$$



Inserting a Stream Element into the Synopsis

- Let v be the value of the $n+1^{th}$ stream element, and t_{i-1} and t_i be tuples in S such that $v_{i-1} \le v < v_i$



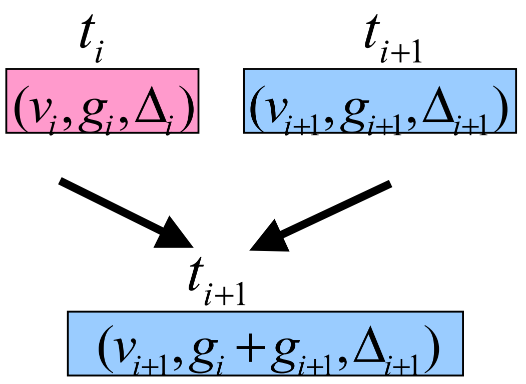
- Maintains invariants

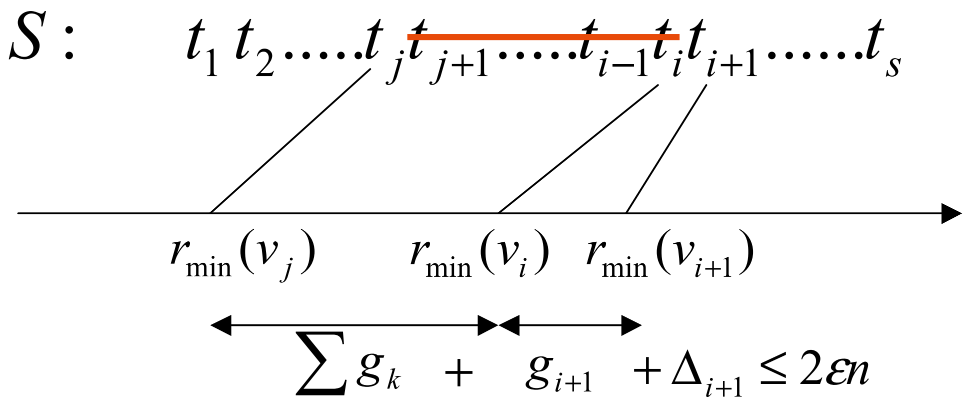
$$g_i = r_{\min}(v_i) - r_{\min}(v_{i-1}) \qquad \Delta_i = r_{\max}(v_i) - r_{\min}(v_i)$$

- $\frac{1}{2\varepsilon}$ elements per Δ_i value
  - Δ_i for a tuple is never modified, after it is inserted

Overview of Algorithm & Analysis

- Partition the Δ_i values into $\log(2\varepsilon n)$ "bands"
  - Remember: we need to maintain $g_i + \Delta_i \le 2\varepsilon n$ => tuples in higher bands have more *capacity* (= max. no. of observations that can be counted in g_i)
- Periodically (every $\frac{1}{2\varepsilon}$ observations) *compress* the quantile synopsis in a right-to-left pass
  - *Collapse ti into t(i+1)* if: (a) t(i+1) is at a higher Δ -band than ti, and (b) $g_i + g_{i+1} + \Delta_{i+1} < 2\varepsilon n$ ← **Maintain our error invariant**

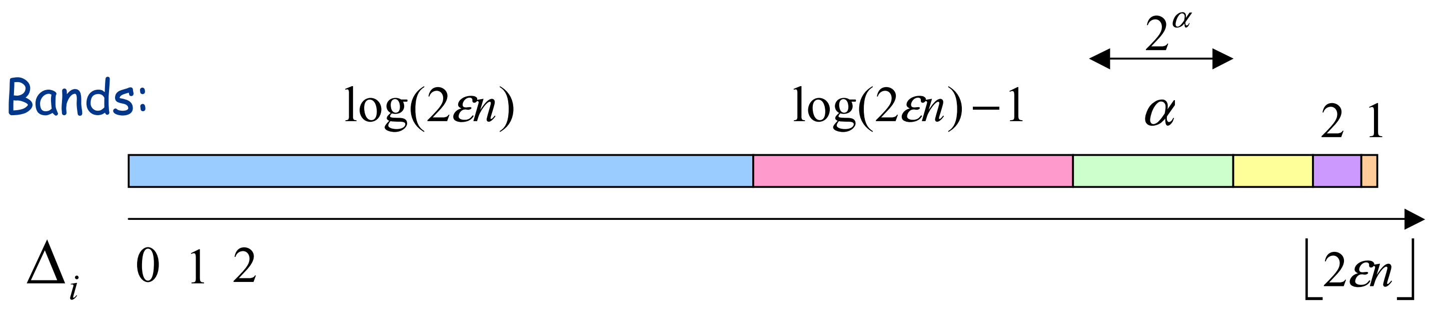

t_i
(v_i, g_i, Δ_i)
t_{i+1}
$(v_{i+1}, g_{i+1}, \Delta_{i+1})$
t_{i+1}
$(v_{i+1}, g_i + g_{i+1}, \Delta_{i+1})$


$S: \quad t_1\, t_2 \ldots\ldots t_j\, \cancel{t_{j+1} \ldots\ldots t_{i-1}\, t_i}\, t_{i+1} \ldots\ldots t_s$
$r_{\min}(v_j)$
$r_{\min}(v_i)$
$r_{\min}(v_{i+1})$
$\sum g_k \; + \; g_{i+1} \; + \Delta_{i+1} \le 2\varepsilon n$

- **Theorem:** Maximum number of "alive" tuples from each Δ -band is $\frac{11}{2\varepsilon}$
  - Overall space complexity: $\frac{11}{2\varepsilon}\log(2\varepsilon n)$

Bands

- Δ_i values split into $\log(2\varepsilon n)$ bands
- size of band $\alpha \leq 2^\alpha$ (adjusted as n increases)

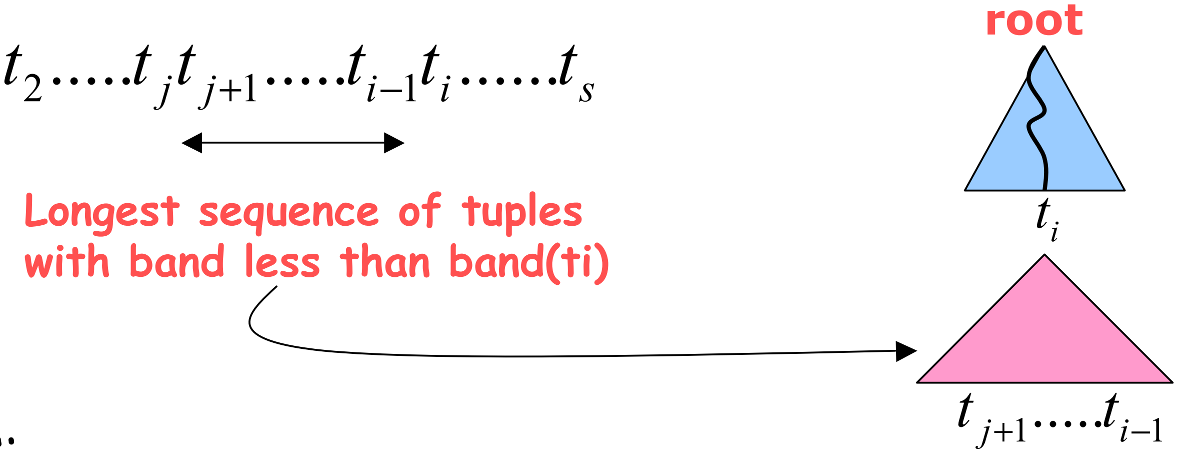


- Higher bands have higher capacities (due to smaller Δ_i values)
- Maximum value of Δ_i in band α: $(2\varepsilon n - 2^{\alpha-1})$
- Number of elements covered by tuples with bands in [0, ..., α]: $\dfrac{2^\alpha}{\varepsilon}$
  - $\dfrac{1}{2\varepsilon}$ elements per Δ_i value

Tree Representation of Synopsis

- Parent of tuple ti: closest tuple tj (j>i) with band(tj) > band(ti)

$S: \quad t_1 t_2 \ldots\ldots t_j t_{j+1} \ldots\ldots t_{i-1} t_i \ldots\ldots t_s$


Longest sequence of tuples
with band less than band(ti)
root
t_i
$t_{j+1} \ldots\ldots t_{i-1}$

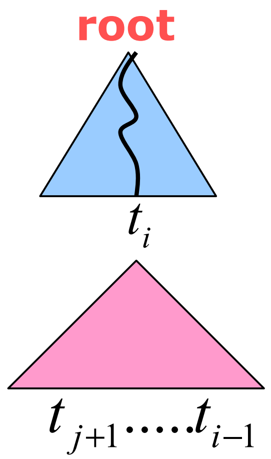
- Properties:
  - Descendants of ti have smaller band values than ti (larger Δ_i values)
  - Descendants of ti form a contiguous segment in S
  - Number of elements covered by ti (with band α) and descendants:

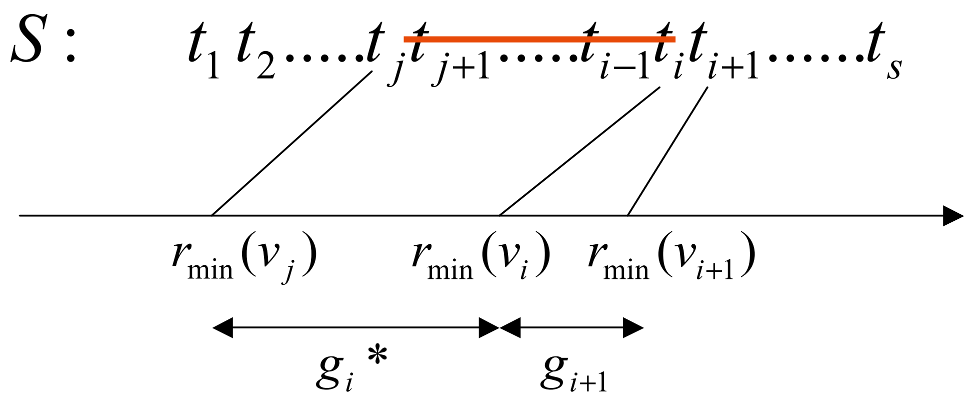
$$g_i^* \le 2^{\alpha} / \varepsilon$$

    - Note: gi* is sum of gi values of ti and its descendants
- Collapse each tuple with parent or sibling in tree

Compressing the Synopsis

- Every $\frac{1}{2\varepsilon}$ elements, compress synopsis
- For i from s-1 down to 1
  - if $(\text{band}(t_i) \le \text{band}(t_{i+1})$ and $g_i^* + g_{i+1} + \Delta_{i+1} < 2\varepsilon n)$
    - $g_{i+1} = g_i^* + g_{i+1}$
    - delete ti and all its descendants from S




$S:$
$t_1\, t_2 \ldots . t_j\, \cancel{t_{j+1} \ldots . t_{i-1}}\, t_i\, t_{i+1} \ldots . t_s$
$r_{\min}(v_j)$
$r_{\min}(v_i)$
$r_{\min}(v_{i+1})$
g_i^*
g_{i+1}

- Maintains invariants: $g_i + \Delta_i \le 2\varepsilon n, \qquad g_i = r_{\min}(v_i) - r_{\min}(v_{i-1})$

Analysis

- Lemma: Both insert and compress preserve the invariant $g_i + \Delta_i \leq 2\varepsilon n$
- Theorem: Let i be the max index in S such that $r_{\max}(v_{i-1}) \leq r + \varepsilon n$. Then,

$$r - \varepsilon n \leq \text{rank}(v_{i-1}) \leq r + \varepsilon n$$

- Lemma: Synopsis S contains at most $\frac{11}{2\varepsilon}$ tuples from each band α
  - For each tuple ti in S, $g_i * + g_{i+1} + \Delta_{i+1} \geq 2\varepsilon n$
  - Also, $g_i * \leq 2^\alpha / \varepsilon$ and $\Delta_i \leq (2\varepsilon n - 2^{\alpha-1})$
- Theorem: Total number of tuples in S is at most $\frac{11}{2\varepsilon}\log(2\varepsilon n)$
  - Number of bands: $\log(2\varepsilon n)$

One-Dimensional Haar Wavelets

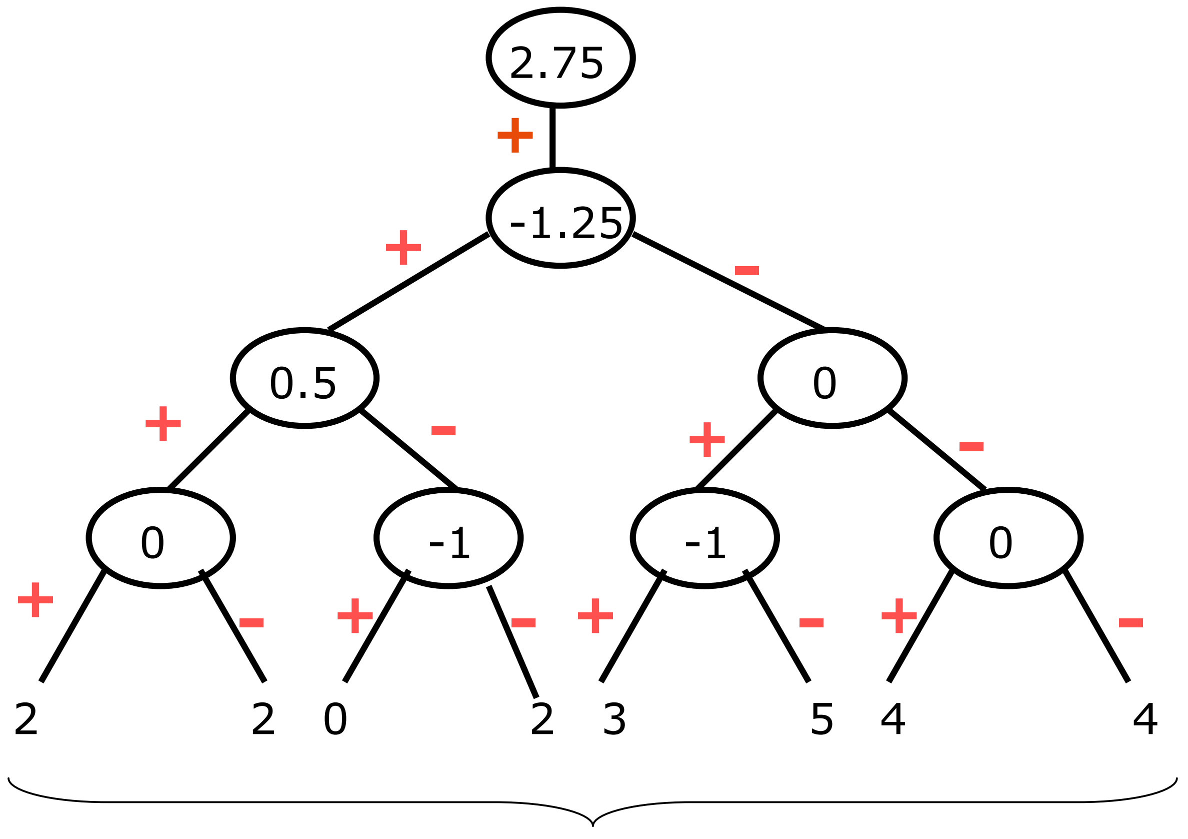
- Wavelets: Mathematical tool for hierarchical decomposition of functions/signals
- Haar wavelets: Simplest wavelet basis, easy to understand and implement
  - *Recursive pairwise averaging and differencing* at different resolutions

Resolution	Averages	Detail Coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	----
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[1.5, 4]	[0.5, 0]
0	[2.75]	[-1.25]

Haar wavelet decomposition: [2.75, -1.25, 0.5, 0, 0, -1, -1, 0]

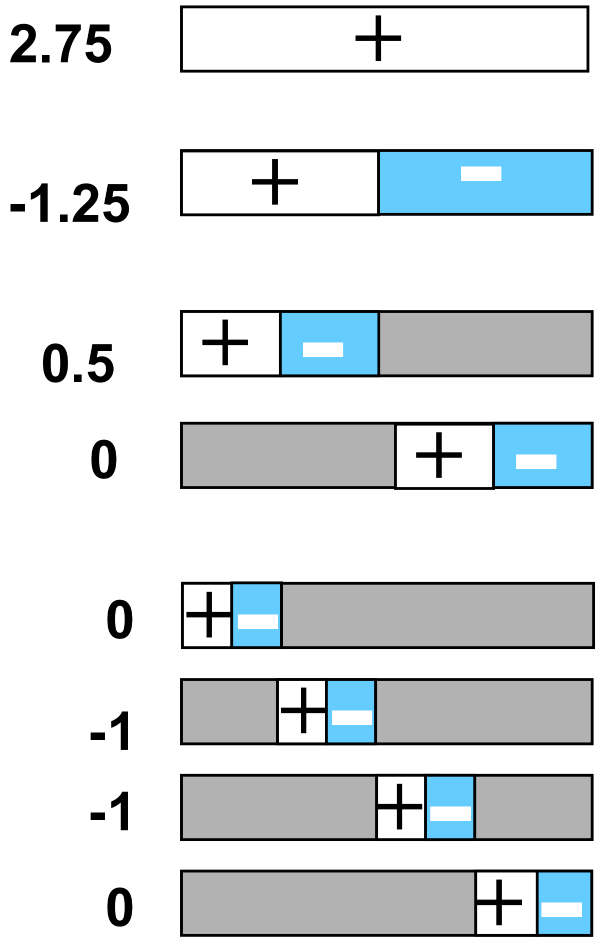
Haar Wavelet Coefficients

- Hierarchical decomposition structure (a.k.a. "error tree")



Original frequency distribution

Coefficient "Supports"

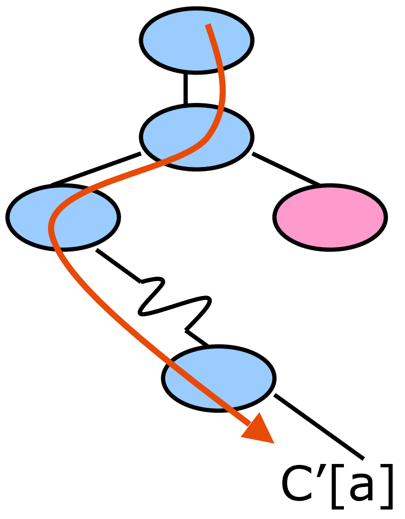


Wavelet-based Histograms [MVW98]

- Problem: Range-query selectivity estimation
- Key idea: Use a compact subset of Haar/linear wavelet coefficients for approximating frequency distribution
- Steps
  - Compute cumulative frequency distribution C
  - Compute Haar (or linear) wavelet transform of C
  - Coefficient *thresholding* : only m<<n coefficients can be kept
    - Take largest coefficients in *absolute normalized value*
      - Haar basis: divide coefficients at resolution j by $\sqrt{2^j}$
      - *Optimal* in terms of the overall Mean Squared (L2) Error
    - Greedy heuristic methods
      - Retain coefficients leading to large error reduction
      - Throw away coefficients that give small increase in error

Using Wavelet-based Histograms

- Selectivity estimation: count(a<= R.e<= b) = C'[b] - C'[a-1]
  - C' is the (approximate) "reconstructed" cumulative distribution
  - Time: O(min{m, logN}), where m = size of wavelet synopsis (number of coefficients), N= size of domain



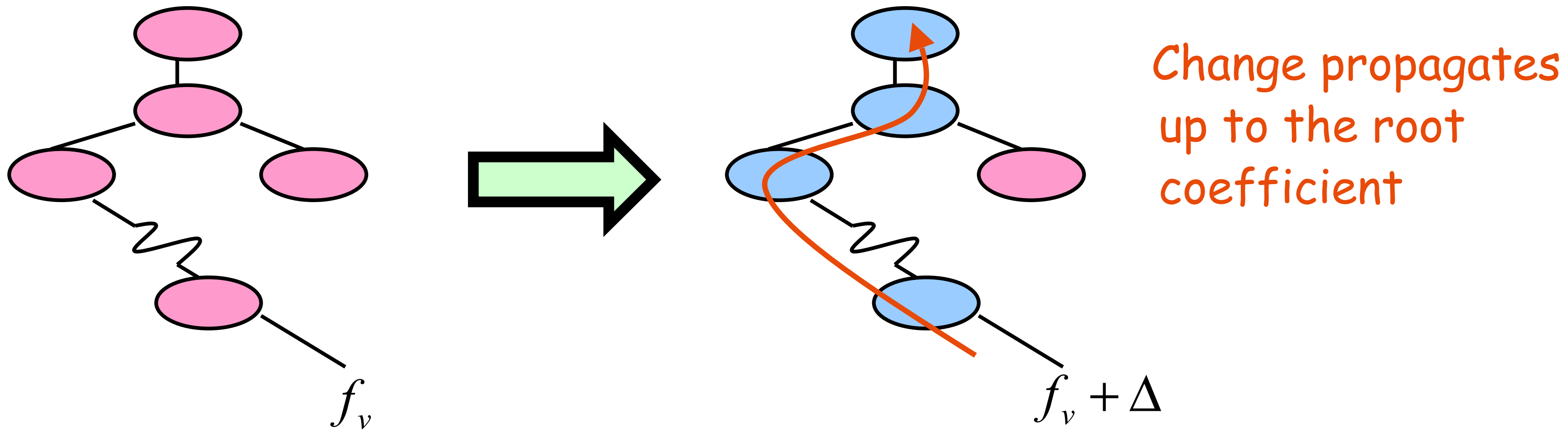
- At most logN+1 coefficients are needed to reconstruct any C' value

- Empirical results over synthetic data
  - Improvements over random sampling and histograms

Dynamic Maintenance of Wavelet-based Histograms [MVW00]

Lucent Technologies Bell Labs Innovations CORNELL

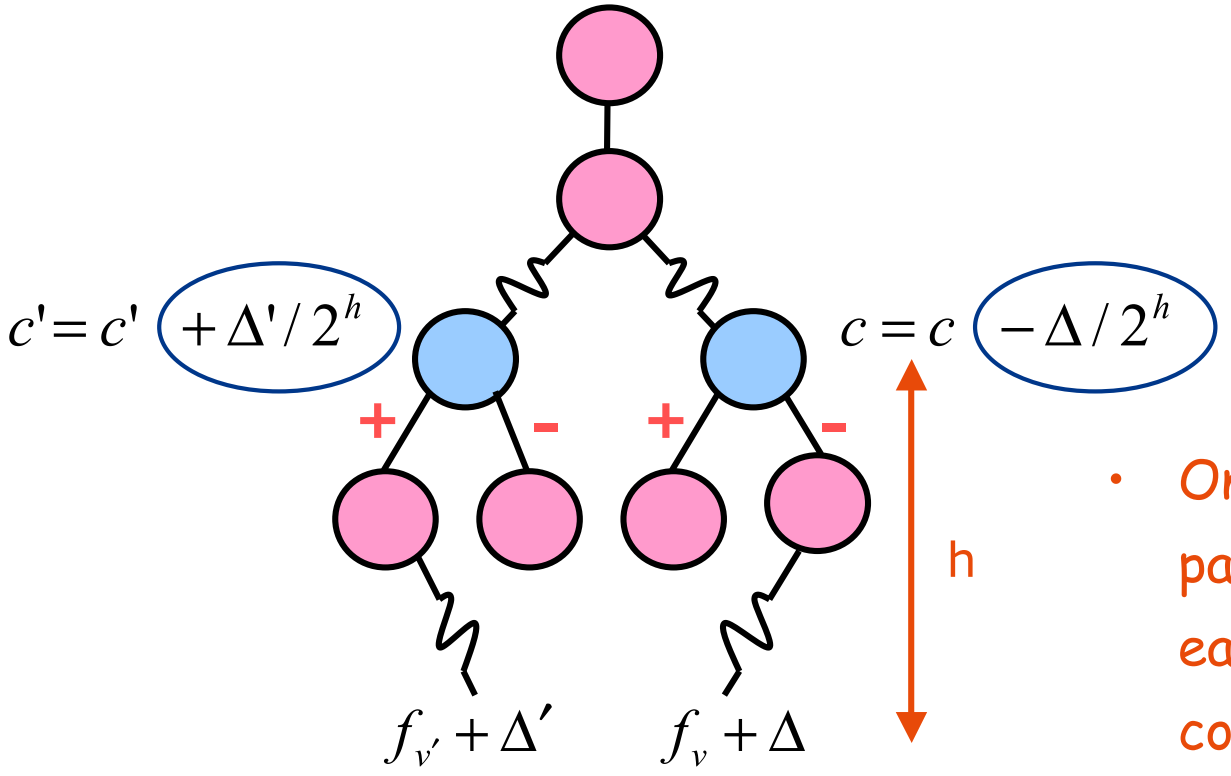
- Build Haar-wavelet synopses on the original frequency distribution
  - Similar accuracy with CDF, makes maintenance simpler
- Key issues with dynamic wavelet maintenance
  - Change in single distribution value can affect the values of many coefficients (path to the root of the decomposition tree)



  - As distribution changes, "most significant" (e.g., largest) coefficients can also change!
    - Important coefficients can become unimportant, and vice-versa

Effect of Distribution Updates

- Key observation: for each coefficient c in the Haar decomposition tree
  - c = (AVG(leftChildSubtree(c)) - AVG(rightChildSubtree(c))) / 2



- Only coefficients on path(v) are affected and each can be updated in constant time

Maintenance Algorithm [MWV00] - Simplified Version

- Histogram H: Top m wavelet coefficients
- For each new stream element (with value v)
  - For each coefficient c on path(v) and with "height" h
    - If c is in H, update c (by adding or substracting $1/2^h$)
  - For each coefficient c on path(v) and not in H
    - Insert c into H with probability proportional to $1/(\min(H)*2^h)$ (*Probabilistic Counting* [FM85])
      - Initial value of c: min(H), the minimum coefficient in H
    - If H contains more than m coefficients
      - Delete minimum coefficient in H

Outline

- Introduction & motivation
  - Stream computation model, Applications
- Basic stream synopses computation
  - Samples, Equi-depth histograms, Wavelets
- Mining data streams
  - Decision trees, clustering
- Sketch-based computation techniques
  - Self-joins, Joins, Wavelets, V-optimal histograms
- Advanced techniques
  - Sliding windows, Distinct values, Hot lists
- Future directions & Conclusions

Clustering Data Streams [GMMO01]

K-median problem definition:

- Data stream with points from metric space
- Find k centers in the stream such that the sum of distances from data points to their closest center is minimized.

Previous work: Constant-factor approximation algorithms

Two-step algorithm:

STEP 1: For each set of M records, S_i, find O(k) centers in S_1, …, S_l

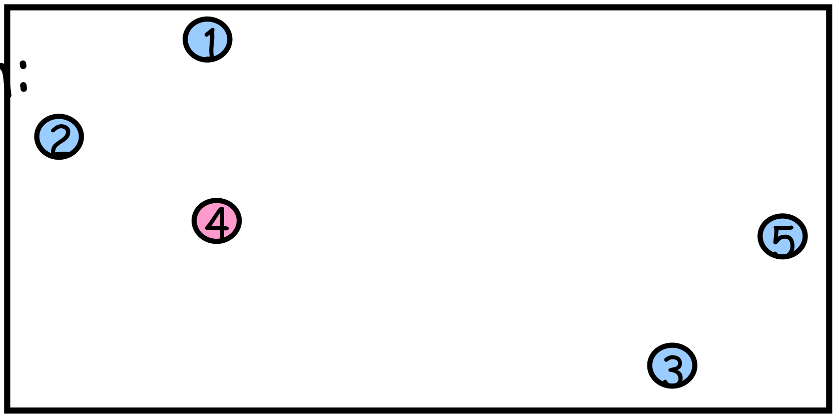
- Local clustering: Assign each point in S_i to its closest center

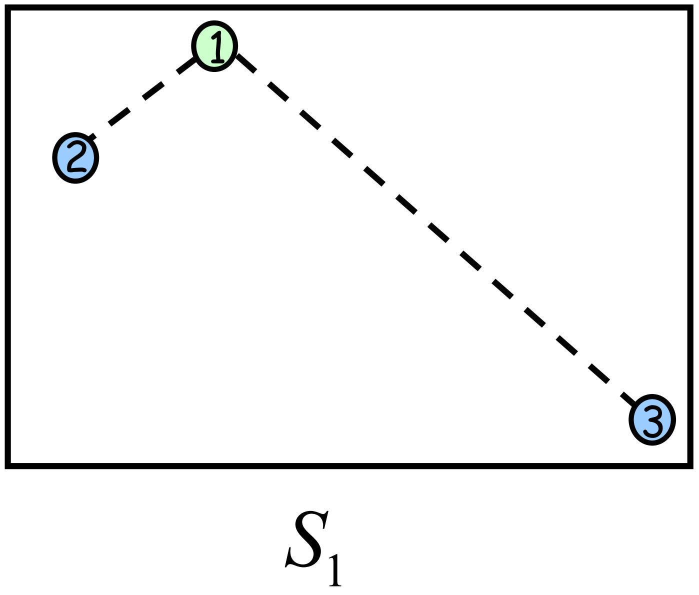
STEP 2: Let S' be centers for S_1, …, S_l with each center weighted by number of points assigned to it. Cluster S' to find k centers

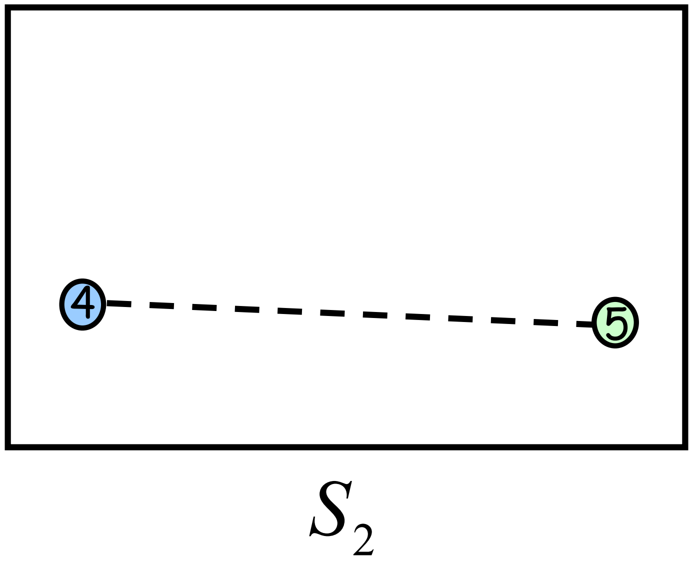
Algorithm forms a building block for more sophisticated algorithms (see paper).

One-Pass Algorithm - First Phase (Example)

- M= 3, k=1, Data Stream:

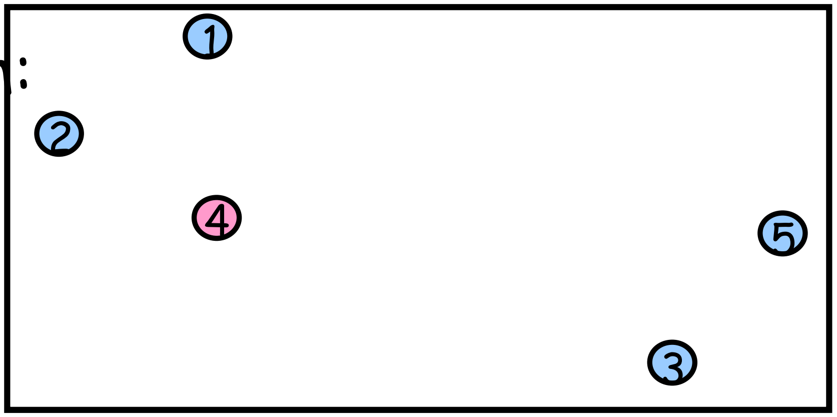


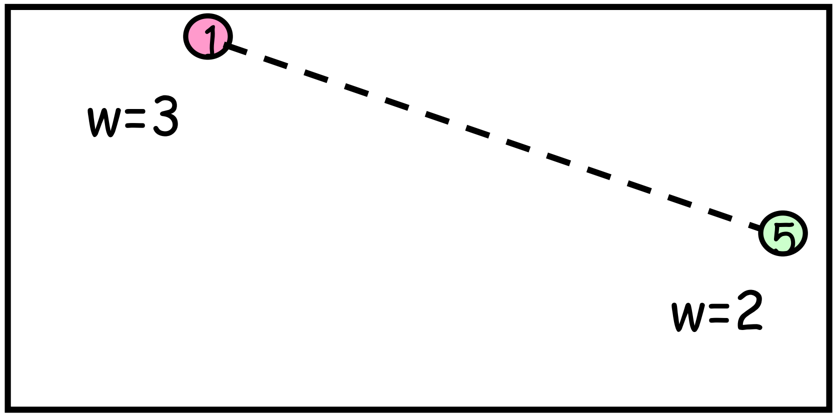




One-Pass Algorithm - Second Phase (Example)

- M= 3, k=1, Data Stream:

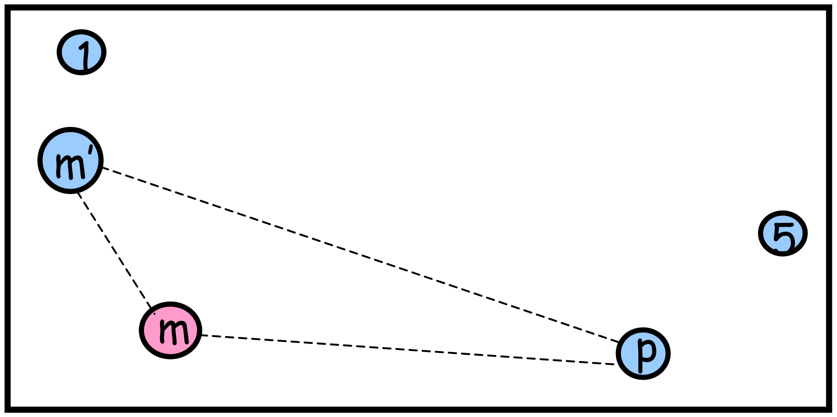




S'

Analysis

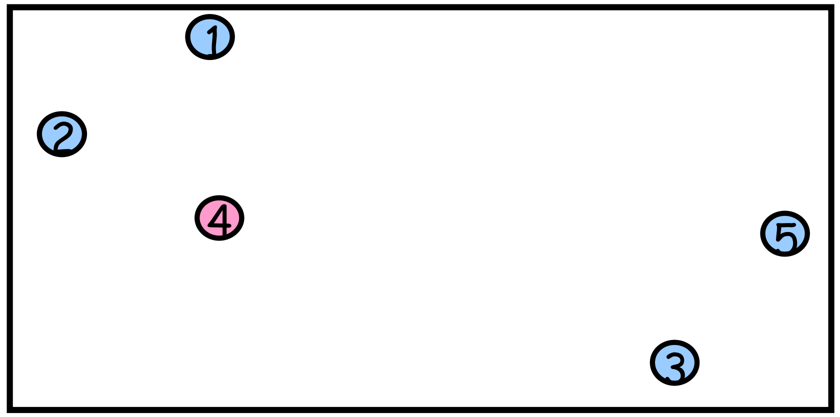
- Observation 1: Given dataset D and solution with cost C where medians do not belong to D, then there is a solution with cost 2C where the medians belong to D.



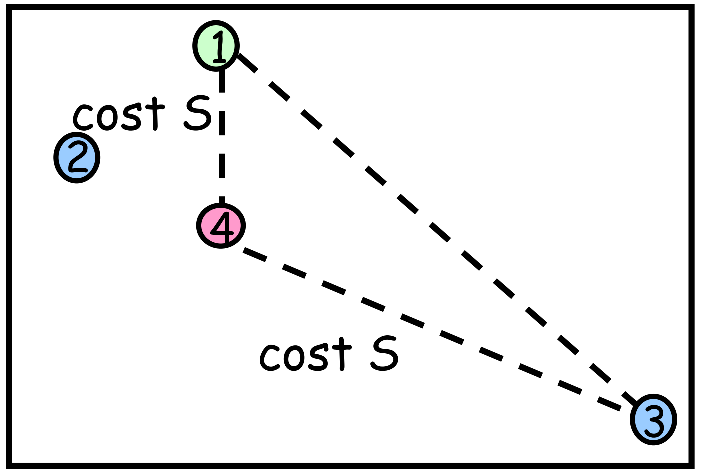
- Argument: Let m be the old median. Consider m' in D closest to the m, and a point p.
  - If p is closest to the median: DONE.
  - If is not closest to the median: d(p,m') <= d(p,m) + d(m,m') <= 2*d(p,m)

Analysis: First Phase

- Observation 2: The sum of the optimal solution costs for the k-median problem for S_1, …, S_l is at most twice the cost of the optimal solution for S



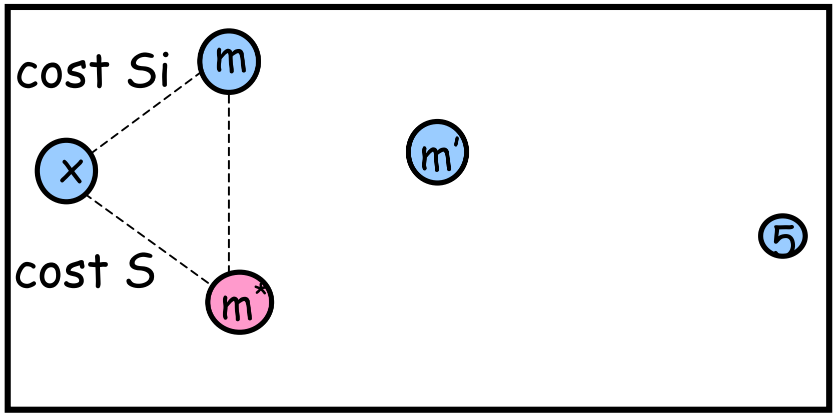
Data Stream



S_1

Analysis: Second Phase

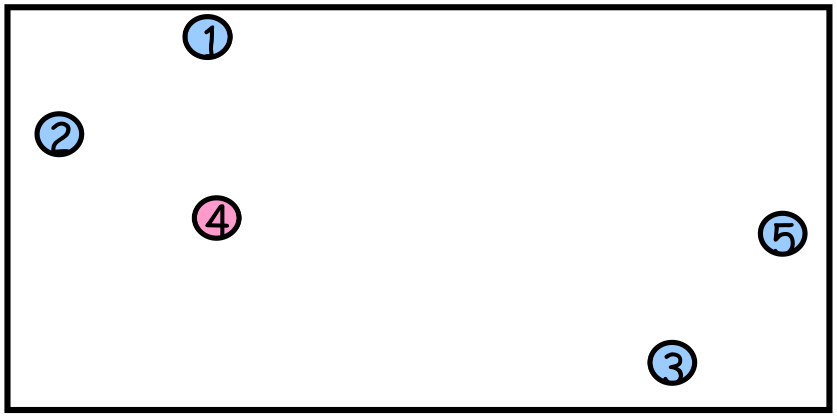
- Observation 3: Cluster weighted medians S'
  - Consider point x with median m* in S and median m in S_i.
    Let m belong to median m' in S'
    Cost due to x in S' = d(m,m')
    Note that d(m,m*) <= d(m,x) + d(x,m*)
      Optimal cost (with medians m* in S) <= sum cost(Si) + cost(S)



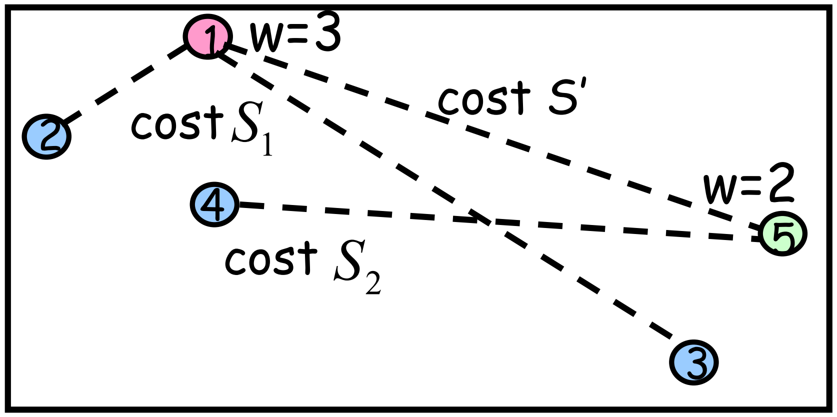
  - Use Observation 1 to construct solution for medians m' in S' with additional factor 2.

Overall Analysis of Algorithm

- Final Result:
  Cost of final solution is at most the sum of costs of S′ and $S_1, \ldots, S_l$, which is at most a constant times (8) cost of S



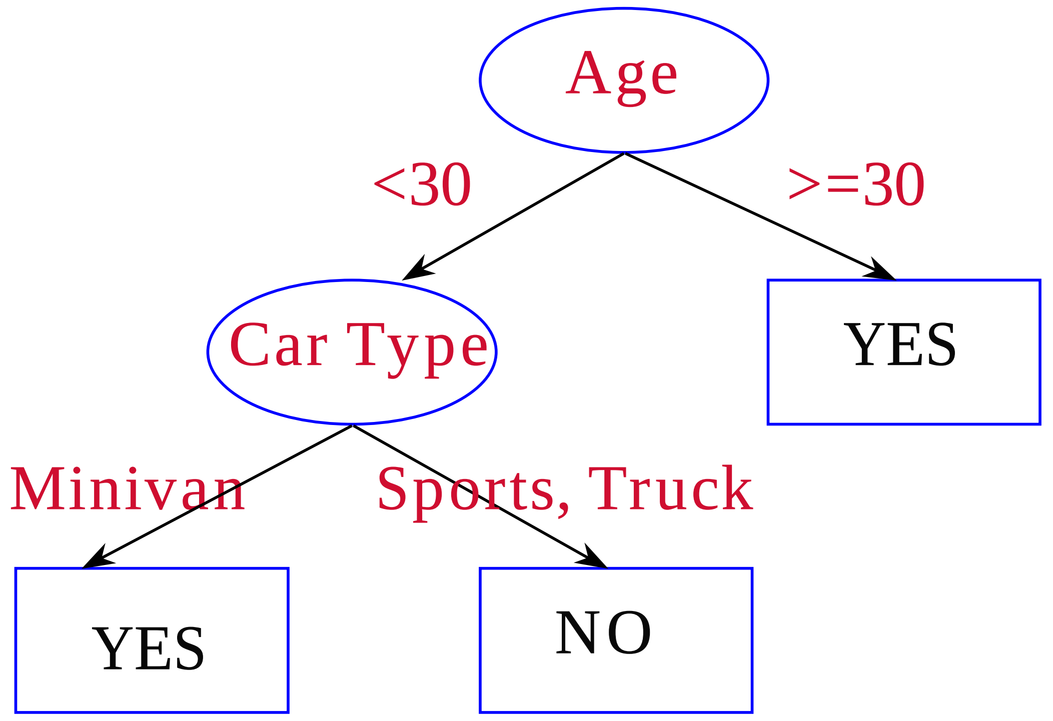
Data Stream

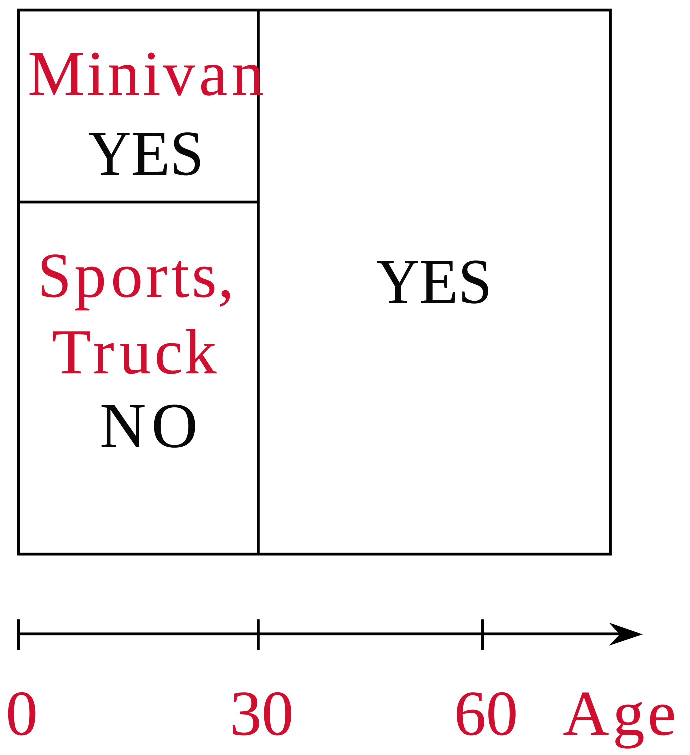


S′

- If constant factor approximation algorithm is used to cluster $S_1, \ldots, S_l$ then simple algorithm yields constant factor approximation
- Algorithm can be extended to cluster in more than 2 phases

Decision Trees





Decision Tree Construction

- Top-down tree construction schema:
  - Examine training database and find best splitting predicate for the root node
  - Partition training database
  - Recurse on each child node

```
BuildTree(Node t, Training database D, Split Selection Method S)
(1) Apply S to D to find splitting criterion
(2) if (t is not a leaf node)
(3)         Create children nodes of t
(4)         Partition D into children partitions
(5)         Recurse on each partition
(6) endif
```

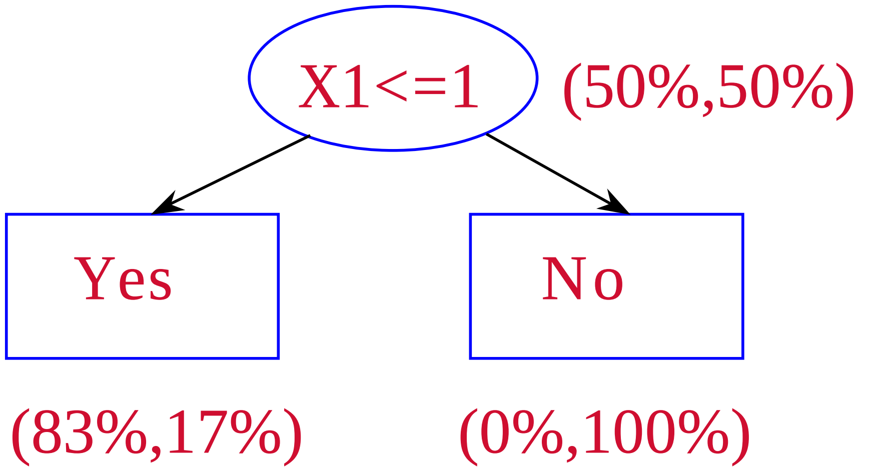
Decision Tree Construction *(cont.)*

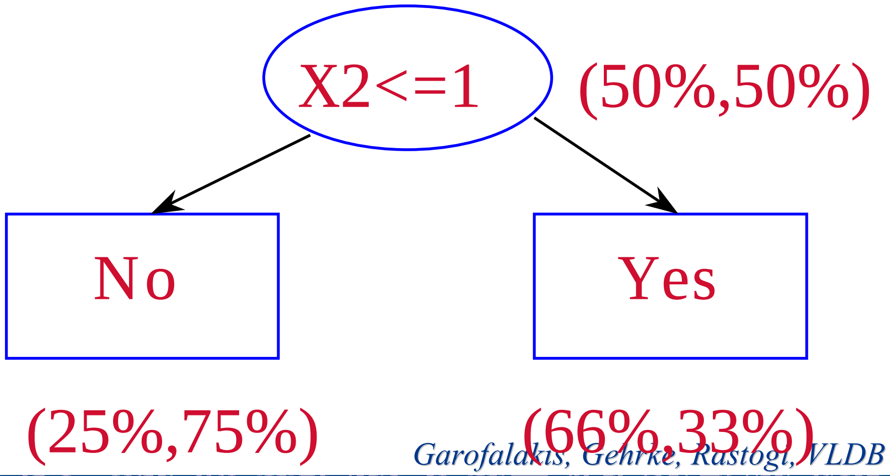
- Three algorithmic components:
  - Split selection (CART, C4.5, QUEST, CHAID, CRUISE, …)
  - Pruning (direct stopping rule, test dataset pruning, cost-complexity pruning, statistical tests, bootstrapping)
  - Data access (CLOUDS, SLIQ, SPRINT, RainForest, BOAT, UnPivot operator)

- Split selection
  - Multitude of split selection methods in the literature
  - Impurity-based split selection: C4.5

Intuition: Impurity Function

X1	X2	Class
1	1	Yes
1	2	Yes
1	2	Yes
1	2	Yes
1	2	Yes
1	1	No
2	1	No
2	1	No
2	2	No
2	2	No





Impurity Function

Let $p(j|t)$ be the proportion of class j training records at node t. Then the node impurity measure at node t:

$i(t) = \text{phi}(p(1|t), \ldots, p(J|t))$ [estimated by empirical prob.]

Properties:

- phi is symmetric, maximum value at arguments $(J^{-1}, \ldots, J^{-1})$, $\text{phi}(1,0,\ldots,0) = \ldots = \text{phi}(0,\ldots,0,1) = 0$

The *reduction in impurity* through splitting predicate s on attribute X:

$\Delta(s,X,t) = \text{phi}(t) - p_L\, \text{phi}(t_L) - p_R\, \text{phi}(t_R)$

Split Selection

Select split attribute and predicate:

- For each categorical attribute X, consider making one child node per category
- For each numerical or ordered attribute X, consider all binary splits s of the form X <= x, where x in dom(X)

At a node t, select split s* such that
Δ(s*,X*,t) is maximal over all
s,X considered

Estimation of empirical probabilities:
Use sufficient statistics

Age	Yes	No
20	15	15
25	15	15
30	15	15
40	15	15

Car	Yes	No
Sport	20	20
Truck	20	20
Minivan	20	20

VFDT/CVFDT [DH00,DH01]

- VFDT:
  - Constructs model from data stream instead of static database
  - Assumes the data arrives iid
  - With high probability, constructs the identical model that a traditional (greedy) method would learn
- CVFDT: Extension to time changing data

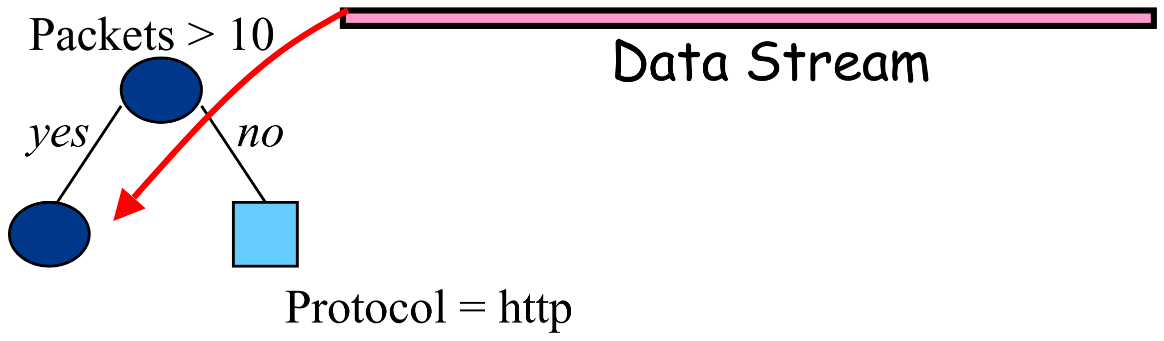
VFDT (Contd.)

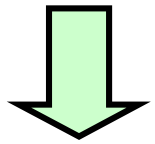
- Initialize T to root node with counts 0
- For each record in stream
  - Traverse T to determine appropriate leaf L for record
  - Update (attribute, class) counts in L and compute best split function $\Delta(s^*, X, L)$ for each attribute X_i
  - If there exists i: $\Delta(s^*, X_i, L) - \Delta(s_i^*, X, L) > \varepsilon$ for all X_i neq X -- (1)
    - split L using attribute X_i
- Compute value for ε using Hoeffding Bound
  - Hoeffding Bound: If $\Delta(s, X, L)$ takes values in range R, and L contains m records, then with probability 1-δ, the computed value of $\Delta(s, X, L)$ (using m records in L) differs from the true value by at most ε

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2m}}$$

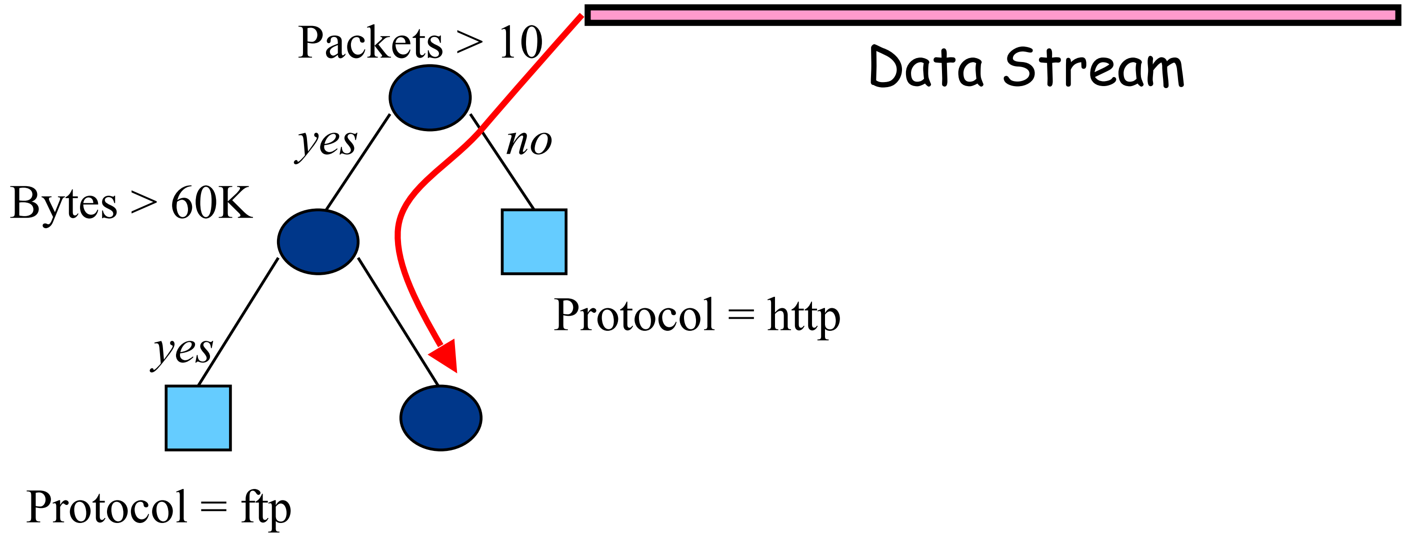
  - Hoeffding Bound guarantees that if (1) holds, then X_i is correct choice for split with probability 1-δ

Single-Pass Algorithm (Example)





$\Delta(\text{Bytes}) - \Delta(\text{Packets}) > \varepsilon$



Analysis of Algorithm

- Result: Expected probability that constructed decision tree classifies a record differently from conventional tree is less than δ/p
  - Here p is probability that a record is assigned to a leaf at each level

Comparison

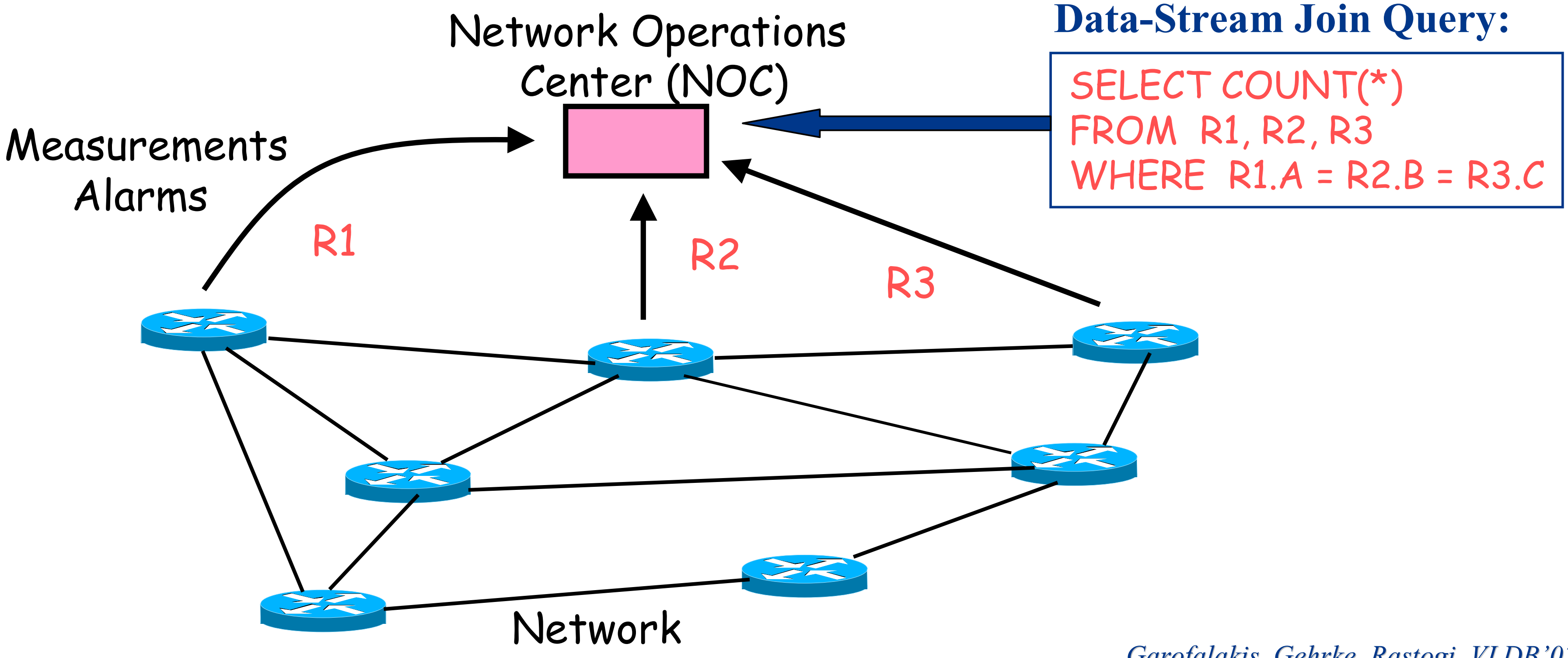
- Approach to decision trees:
  Use inherent partially incremental offline construction of the data mining model to extend it to the data stream model
  - Construct tree in the same way, but wait for significant differences
  - Instead of re-reading dataset, use new data from the stream
  - "Online aggregation model"
- Approach to clustering:
  Use offline construction as a building block
  - Build larger model out of smaller building blocks
  - Argue that composition does not loose too much accuracy
  - "Composing approximate query operators"?

Outline

- Introduction & motivation
  - Stream computation model, Applications
- Basic stream synopses computation
  - Samples, Equi-depth histograms, Wavelets
- Mining data streams
  - Decision trees, clustering, association rules
- Sketch-based computation techniques
  - Self-joins, Joins, Wavelets, V-optimal histograms
- Advanced techniques
  - Distinct values, Sliding windows, Hot lists
- Future directions & Conclusions

Query Processing over Data Streams

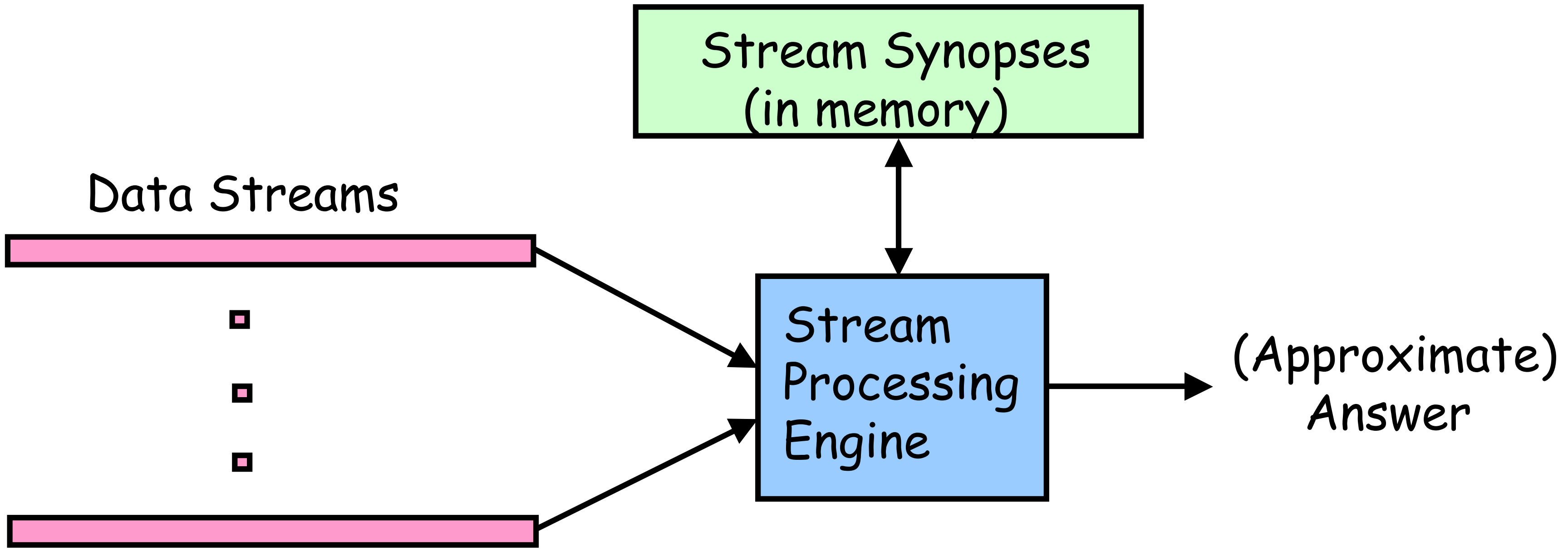
- Stream-query processing arises naturally in Network Management
  - Data tuples arrive continuously from different parts of the network
  - Archival storage is often off-site (expensive access)
  - Queries can only look at the tuples *once, in the fixed order of arrival* and with *limited available memory*



```
SELECT COUNT(*)
FROM  R1, R2, R3
WHERE  R1.A = R2.B = R3.C
```

Data Stream Processing Model

- Approximate query answers often suffice (e.g., trend/pattern analyses)
  - Build small *synopses* of the data streams online
  - Use synopses to provide (good-quality) approximate answers



- Requirements for stream synopses
  - *Single Pass:* Each tuple is examined at most once, in fixed (arrival) order
  - *Small Space:* Log or poly-log in data stream size
  - *Real-time:* Per-record processing time (to maintain synopsis) must be low

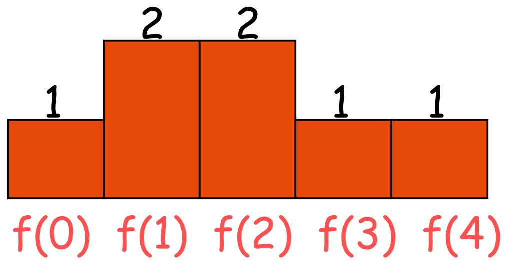
Stream Data Synopses

- Conventional data summaries fall short
  - Quantiles and 1-d histograms: Cannot capture attribute correlations
  - Samples (e.g., using Reservoir Sampling) perform poorly for joins
  - Multi-d histograms/wavelets: Construction requires multiple passes over the data
- Different approach: *Randomized sketch synopses*
  - Only logarithmic space
  - *Probabilistic guarantees* on the quality of the approximate answer
- *Overview*
  - Basic technique
  - Extension to relational query processing over streams
  - Extracting wavelets and histograms from sketches
  - Extensions (stable distributions, distinct values, quantiles)

Randomized Sketch Synopses for Streams

- *Goal:* Build small-space summary for distribution vector f(i) (i=0,..., N-1) seen as a stream of i-values

Data stream: 2, 0, 1, 3, 1, 2, 4, . . .



- *Basic Construct: Randomized Linear Projection of f()* = inner/dot product of f-vector

$$< f, \xi > = \sum f(i)\xi_i$$

where ξ = vector of random values from an appropriate distribution

  - Simple to compute over the stream: Add ξ_i whenever the i-th value is seen

Data stream: 2, 0, 1, 3, 1, 2, 4, . . . $\longrightarrow$ $\xi_0 + 2\xi_1 + 2\xi_2 + \xi_3 + \xi_4$

  - Generate ξ_i's in small space using pseudo-random generators
  - *Tunable probabilistic guarantees* on approximation error

- Used for low-distortion vector-space embeddings [JL84]
  - Applicability to bounded-space stream computation in [AMS96]

Sketches for 2nd Moment Estimation over Streams [AMS96]

- *Problem:* Tuples of relation R are streaming in -- compute the 2nd frequency moment of attribute R.A, i.e.,

$$F_2(R.A) = \sum_{0}^{N-1} [f(i)]^2$$, where f(i) = frequency(i-th value of R.A)

- $F_2(R.A) = \text{COUNT}(\, R \bowtie_{A} R \,)$ (size of the *self-join* on R.A)

- Exact solution: too expensive, requires O(N) space!!
  - How do we do it in small (O(logN)) space??

Sketches for 2nd Moment Estimation over Streams [AMS96] *(cont.)*

- ***Key Intuition:*** Use randomized linear projections of f() to define a random variable X such that
  - X is easily computed over the stream (in small space)
  - E[X] = F2 *(unbiased estimate)*
  - Var[X] is small

$\longrightarrow$ *Probabilistic Error Guarantees*

- ***Technique***
  - Define a family of *4-wise independent {-1, +1} random variables* $\{\xi_i : i = 0,\dots,N-1\}$
    - P[ξ_i=1] = P[ξ_i=-1] = 1/2
    - Any 4-tuple $\{\xi_i, \xi_j, \xi_k, \xi_l\}, i \neq j \neq k \neq l$ is mutually independent
    - Generate ξ_i values *on the fly* : pseudo-random generator using only O(logN) space (for seeding)!

Sketches for 2nd Moment Estimation over Streams [AMS96] (cont.)

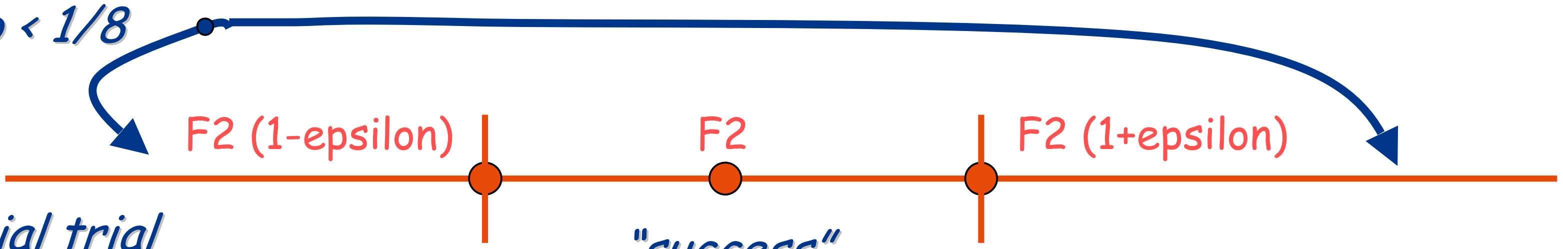
- *Technique (cont.)*
  - Compute the random variable Z = $< f, \xi > = \sum_{0}^{N-1} f(i)\xi_i$
    - Simple linear projection: just add ξ_i to Z whenever the i-th value is observed in the R.A stream
  - Define X = Z^2
- Using 4-wise independence, show that
  - E[X] = F_2 and Var[X] $\leq 2 \cdot F_2^2$
- By Chebyshev: $P[|X - F_2| > \varepsilon \cdot F_2] < \dfrac{Var[X]}{\varepsilon^2 \cdot F_2^2} \leq \dfrac{2}{\varepsilon^2}$

Sketches for 2nd Moment Estimation over Streams [AMS96] *(cont.)*

- ***Boosting Accuracy and Confidence***
  - Build several *independent, identically distributed (iid)* copies of X
  - Use averaging and median-selection operations

  - Y = *average* of $s_1 = 16/\varepsilon^2$ iid copies of X (=> Var[Y] = Var[X]/s1)
    - By Chebyshev: $P[|Y - F_2| > \varepsilon \cdot F_2] < \frac{1}{8}$
  - W = *median* of $s_2 = 2 \cdot \log(1/\delta)$ iid copies of Y

"failure", Prob < 1/8



Each Y = Binomial trial

"success"

$P[|W - F_2| > \varepsilon \cdot F_2] =$ Prob[# failures in s2 trials $\geq$ s2/2 = (1+3) s2/8]

$\leq \delta$ (by *Chernoff bounds*)

Sketches for 2nd Moment Estimation over Streams [AMS96] *(cont.)*

- Total space = O(s1*s2*logN)
  - *Remember:* O(logN) space for "seeding" the construction of each X

- ***Main Theorem***
  - Construct approximation to F2 within a relative error of ε with probability $\geq 1-\delta$ using only $O(\log N \cdot \log(1/\delta)/\varepsilon^2)$ space

- [AMS96] also gives results for other moments and space-complexity lower bounds (communication complexity)
  - Results for F2 approximation are space-optimal (up to a constant factor)

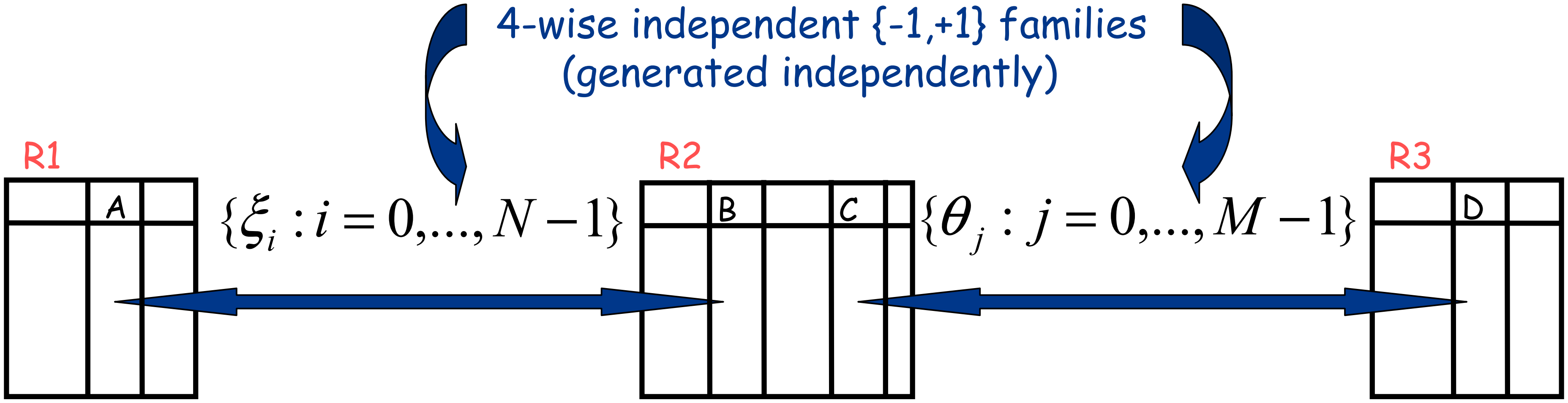
Sketches for Stream Joins and Multi-Joins [AGM99, DGG02]

```
SELECT COUNT(*)/SUM(E)
FROM  R1, R2, R3
WHERE  R1.A = R2.B, R2.C = R3.D
```

$$\text{COUNT} = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} f_1(i) f_2(i,j) f_3(j)$$

(fk() denotes frequencies in Rk)

4-wise independent {-1,+1} families (generated independently)



R1
A
R2
B
C
R3
D
$\{\xi_i : i = 0,\ldots,N-1\}$
$\{\theta_j : j = 0,\ldots,M-1\}$

$$Z_1 = \sum_{i=0}^{N-1} f_1(i)\xi_i \qquad Z_2 = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} f_2(i,j)\xi_i\theta_j \qquad Z_3 = \sum_{j=0}^{M-1} f_3(j)\theta_j$$

Update: R2-tuple with (B,C) = (i,j) $\Rightarrow Z_2 += \xi_i\theta_j$

- Define X = $Z_1 Z_2 Z_3$ -- E[X] = COUNT *(unbiased),* O(logN+logM) space

Sketches for Stream Joins and Multi-Joins [AGM99, DGG02] *(cont.)*

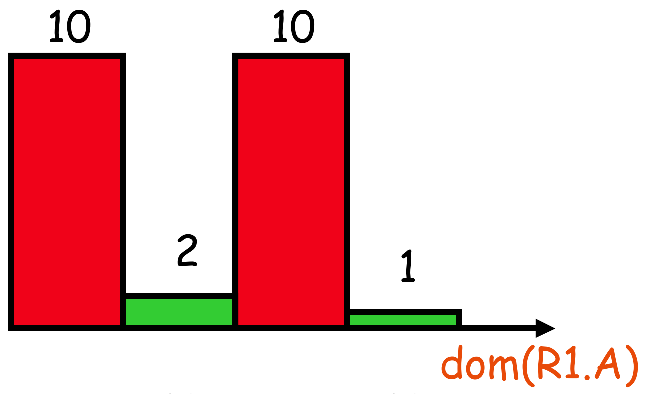
```
SELECT COUNT(*)
FROM  R1, R2, R3
WHERE  R1.A = R2.B, R2.C = R3.D
```

- Define X = $Z_1 Z_2 Z_3$, E[X] = COUNT
- *Unfortunately,* Var[X] increases with the number of joins!!

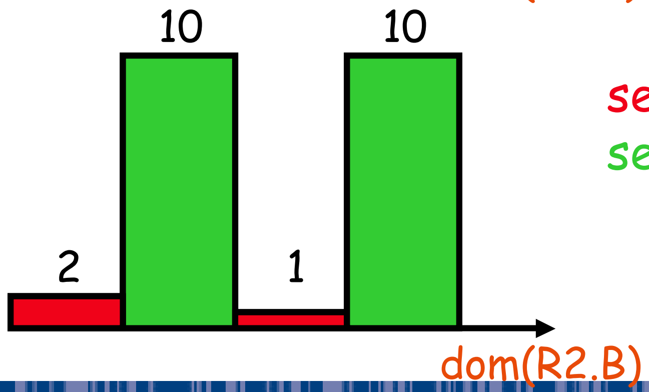
- Var[X] = O($\prod$ self-join sizes) = O($F_2(R_1.A) F_2(R_2.B, R_2.C) F_2(R_3.D)$)
- By Chebyshev: Space needed to guarantee high (constant) relative error probability for X is $O(Var[X]/COUNT^2)$
  - Strong guarantees in limited space only for joins that are "large" (wrt $\prod$ self-join sizes)!
- Proposed solution: ***Sketch Partitioning [DGG02]***

Overview of Sketch Partitioning [DGG02]

- *Key Intuition:* Exploit coarse statistics on the data stream to *intelligently partition the join-attribute space* and the sketching problem in a way that provably tightens our error guarantees
  - Coarse historical statistics on the stream or collected over an initial pass
  - Build independent sketches for each partition (Estimate = $\sum$partition sketches, Variance = $\sum$partition variances)



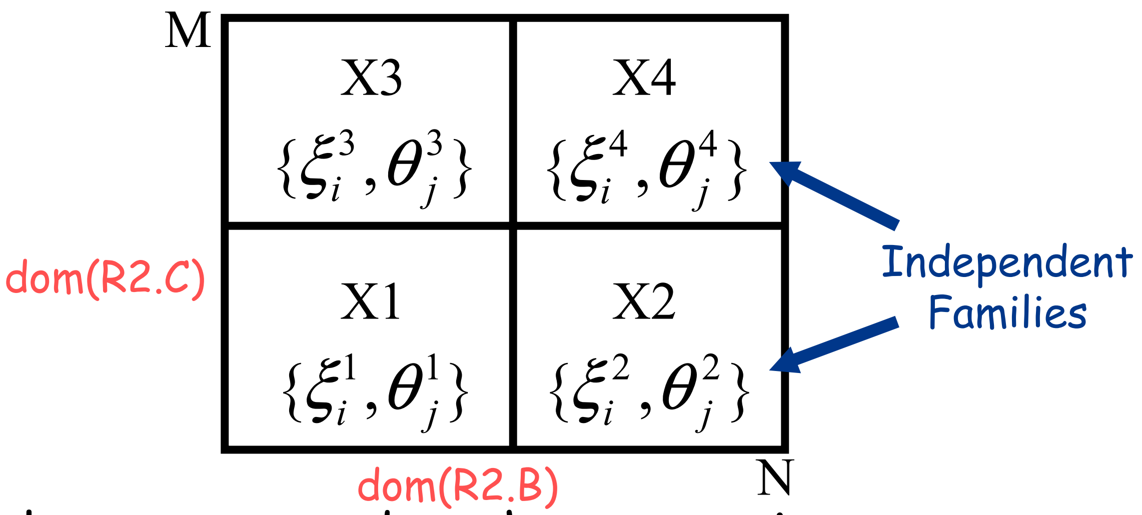
self-join(R1.A)*self-join(R2.B) = 205*205 = 42K



self-join(R1.A)*self-join(R2.B) +
self-join(R1.A)*self-join(R2.B) = 200*5 +200*5 = 2K

Overview of Sketch Partitioning [DGG02] (cont.)

```
SELECT COUNT(*)
FROM  R1, R2, R3
WHERE  R1.A = R2.B, R2.C = R3.D
```


M
X3
$\{\xi_i^3, \theta_j^3\}$
X4
$\{\xi_i^4, \theta_j^4\}$
X1
$\{\xi_i^1, \theta_j^1\}$
X2
$\{\xi_i^2, \theta_j^2\}$
dom(R2.C)
dom(R2.B)
N
Independent Families

- ***Maintenance:*** Incoming tuples are mapped to the appropriate partition(s) and the corresponding sketch(es) are updated
  - Space = O(k(logN+logM)) (k=4= no. of partitions)
- Final estimate X = X1+X2+X3+X4 -- Unbiased, Var[X] = $\sum$ Var[Xi]
- Improved error guarantees
  - Var[X] is smaller (by *intelligent domain partitioning*)
  - "Variance-aware" boosting
    - More space for iid sketch copies to regions of high expected variance (self-join product)

Overview of Sketch Partitioning [DGG02] (cont.)

- *Space allocation among partitions:* Easy to solve optimally once the domain partitioning is fixed
- *Optimal domain partitioning:* Given a K, find a K-partitioning that minimizes

$$\sum_{1}^{K} \sqrt{Var[X_i]} \approx \sum_{1}^{K} \sqrt{\prod size(selfJoin)}$$

- Can solve optimally for *single-join queries* (using Dynamic Programming)
- *NP-hard* for queries with ≥ 2 joins!
- Proposed an efficient DP heuristic (optimal if join attributes in each relation are independent)

- *More details in the paper . . .* ☺

Stream Wavelet Approximation using Sketches [GKM01]

- Single-join approximation with sketches [AGM99]
  - Construct approximation to |R1 $\bowtie$ R2| = $\sum f_1(i) f_2(i)$ within a relative error of ε with probability $\geq 1-\delta$ using space $O(\log N \cdot \log(1/\delta)/(\varepsilon^2 \lambda^2))$, where

$$\lambda \leq \frac{|\sum f_1(i) f_2(i)|}{\sqrt{\sum f_1^2(i) \cdot \sum f_2^2(i)}}$$ = |R1 $\bowtie$ R2| / Sqrt($\prod$ self-join sizes)

- Observation: |R1 $\bowtie$ R2| = $\sum f_1(i) f_2(i) = < f_1, f_2 >$ *= inner product!!*
  - General result for inner-product approximation using sketches

- Other inner products of interest: *Haar wavelet coefficients!*
  - Haar wavelet decomposition = inner products of signal/distribution with specialized (wavelet basis) vectors

Haar Wavelet Decomposition

- **Wavelets**: mathematical tool for hierarchical decomposition of functions/signals
- **Haar wavelets**: simplest wavelet basis, easy to understand and implement
  - *Recursive pairwise averaging and differencing* at different resolutions

Resolution	Averages	Detail Coefficients
3	D = [2, 2, 0, 2, 3, 5, 4, 4]	----
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[1.5, 4]	[0.5, 0]
0	[2.75]	[-1.25]

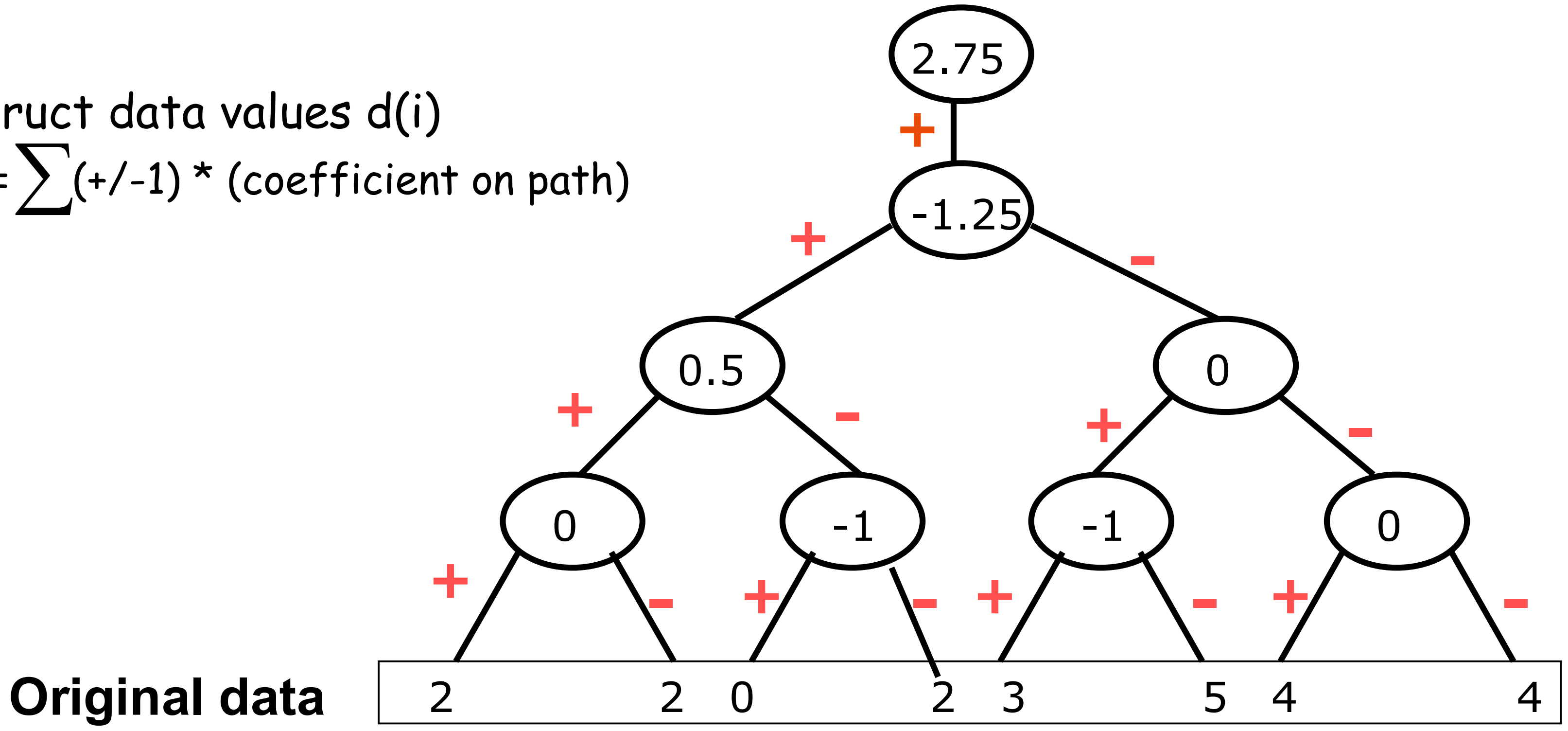
Haar wavelet decomposition: [2.75, -1.25, 0.5, 0, 0, -1, -1, 0]

- Compression by ignoring small coefficients

Haar Wavelet Coefficients

- Hierarchical decomposition structure (a.k.a. ***Error Tree***)

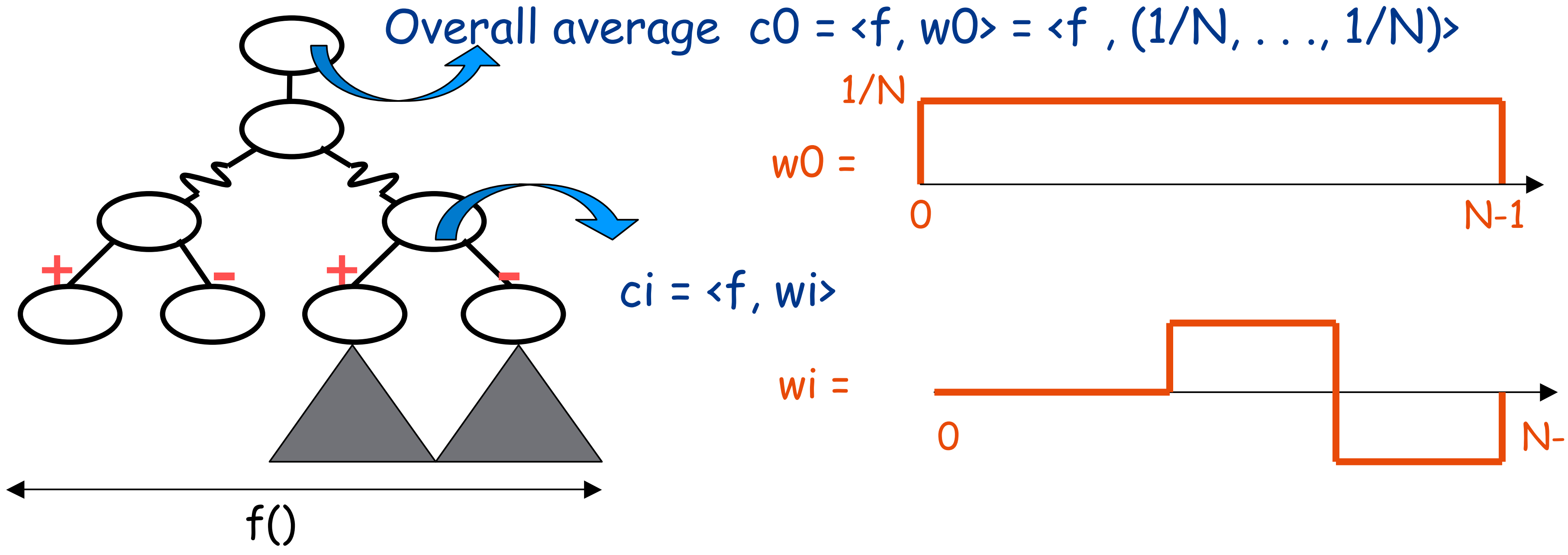
- Reconstruct data values d(i)
  - d(i) = $\sum$(+/-1) * (coefficient on path)



- Coefficient *thresholding* : only B<<|D| coefficients can be kept
  - B is determined by the available synopsis space
  - B largest coefficients in *absolute normalized value*
  - *Provably optimal* in terms of the overall Sum Squared (L2) Error

Stream Wavelet Approximation using Sketches [GKM01] *(cont.)*

- Each (normalized) coefficient ci in the Haar decomposition tree
  - ci = NORMi * (AVG(leftChildSubtree(ci)) - AVG(rightChildSubtree(ci))) / 2



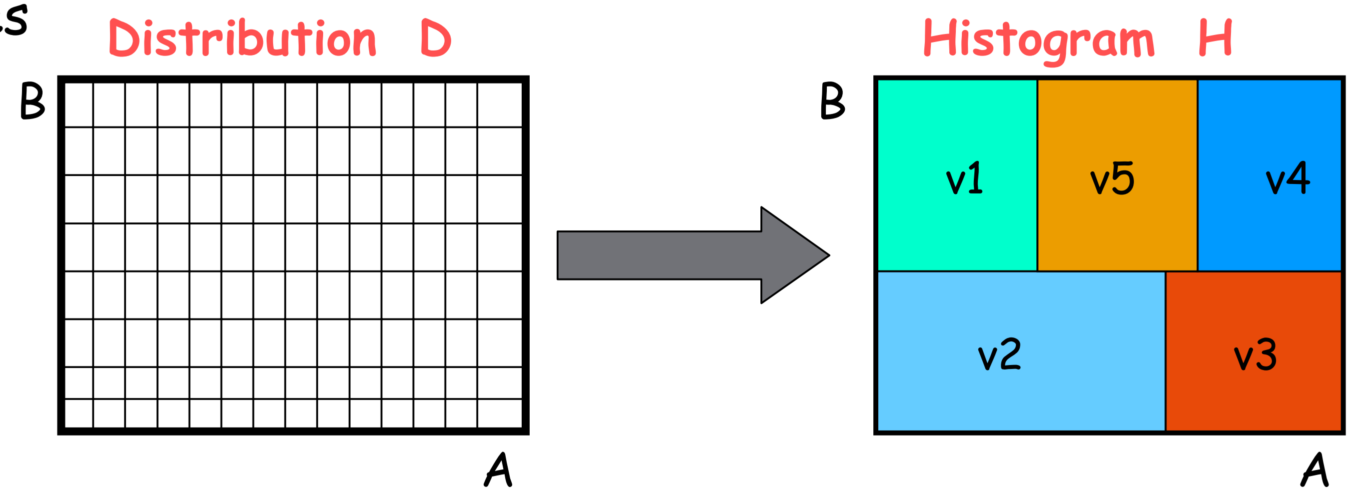
- Use sketches of f() and wavelet-basis vectors to extract "large" coefficients
- *Key:* "Small-B Property" = Most of f()'s "energy" = $\| f \|_2^2 = \sum f^2(i)$ is concentrated in a small number B of large Haar coefficients

Stream Wavelet Approximation using Sketches [GKM01]: The Method

- *Input:* "Stream of tuples" rendering of a distribution f() that has a B-Haar coefficient representation with energy $\geq \eta \cdot \| f \|_2^2$
- Build sufficient sketches on f() to accurately (within ε, δ) estimate *all* Haar coefficients ci = <f, wi> such that |ci| $\geq \sqrt{\varepsilon\eta/B}\, \| f \|_2^2$
  - By the single-join result (with $\lambda = \sqrt{\varepsilon\eta/B}$) the space needed is
    $$O(\log N \cdot \log(N/\delta) \cdot B/(\varepsilon^3\eta))$$
  - N/δ comes from "union bound" (need *all* coefficients with probability $1-\delta$)
- Keep largest B estimated coefficients with absolute value $\geq \sqrt{\varepsilon\eta/B}\, \| f \|_2^2$
- *Theorem:* The resulting approximate representation of (at most) B Haar coefficients has energy $\geq (1-\varepsilon)\eta \cdot \| f \|_2^2$ with probability $\geq 1-\delta$
- *First provable guarantees* for Haar wavelet computation over data streams

Multi-d Histograms over Streams using Sketches [TGI02]

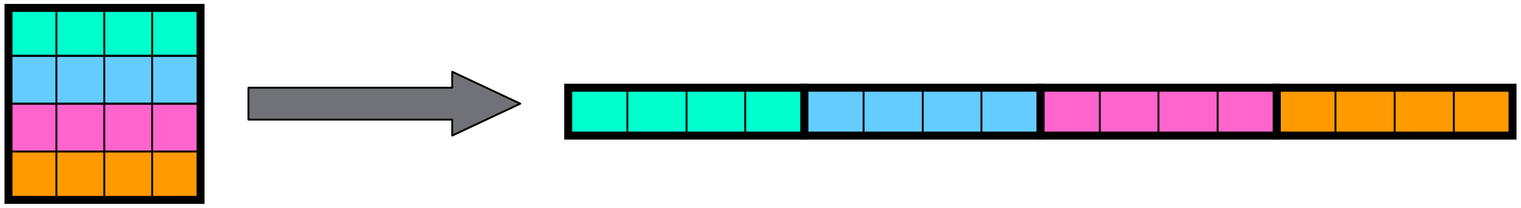
- Multi-dimensional histograms: Approximate *joint data distribution* over multiple attributes



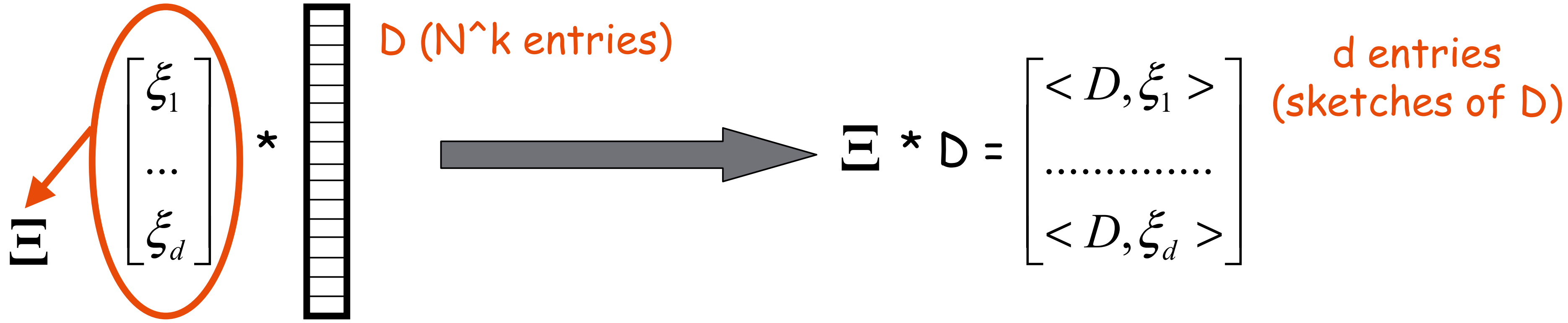
- "Break" multi-d space into *hyper-rectangles (buckets)* & use a single frequency parameter (e.g., average frequency) for each
  - Piecewise constant approximation
  - Useful for query estimation/optimization, approximate answers, etc.
- Want a histogram H that minimizes L2 error in approximation, i.e., $\| D - H \|_2 = \sum (d_i - h_i)^2$ for a given number of buckets *(V-Optimal)*
  - Build over a stream of data tuples??

Multi-d Histograms over Streams using Sketches [TGI02] (cont.)

- View distribution and histograms over {0,...,N-1}x...x{0,...,N-1} as N^k-dimensional vectors



- Use sketching to reduce vector dimensionality from N^k to (small) d



$$\Xi \; \begin{bmatrix} \xi_1 \\ \ldots \\ \xi_d \end{bmatrix} * D \longrightarrow \Xi * D = \begin{bmatrix} <D,\xi_1> \\ \ldots\ldots\ldots\ldots \\ <D,\xi_d> \end{bmatrix}$$

- *Johnson-Lindenstrauss Lemma[JL84]:* Using d=$O(bk\log N/\varepsilon^2)$ guarantees that L2 distances with any b-bucket histogram H are approximately preserved with high probability; that is, $\| \Xi \cdot D - \Xi \cdot H \|_2$ is within a relative error of ε from $\| D - H \|_2$ for any b-bucket H

Multi-d Histograms over Streams using Sketches [TGI02] (cont.)

- *Algorithm*
  - Maintain sketch $\Xi \cdot D$ of the distribution D on-line
  - Use the sketch to find histogram H such that $\| \Xi \cdot D - \Xi \cdot H \|_2$ is minimized
    - Start with H = ϕ and choose buckets one-by-one *greedily*
    - At each step, select the bucket β that minimizes $\| \Xi \cdot D - \Xi \cdot (H \cup \beta) \|_2$
- Resulting histogram H: *Provably near-optimal* wrt minimizing $\| D - H \|_2$ (with high probability)
  - *Key:* L2 distances are approximately preserved (by [JL84])
- Various heuristics to improve running time
  - Restrict possible bucket hyper-rectangles
  - Look for "good enough" buckets

Extensions: Sketching with Stable Distributions [Ind00]

- *Idea:* Sketch the incoming stream of values rendering the distribution f() using random vectors ξ from "special" distributions
- *p-stable distribution* Δ
    - If X1,..., Xn are iid with distribution Δ, a1,..., an are any real numbers
    - Then, $\sum a_i X_i$ has the *same distribution* as $\left(\sum |a_i|^p\right)^{1/p} X$, where X has distribution Δ
- Known to exist for any p∈ (0,2]
    - p=1: Cauchy distribution
    - p=2: Gaussian (Normal) distribution
- For p-stable ξ : Know the *exact distribution* of $< f, \xi >= \sum f(i)\xi_i$
    - Basically, sample from $\left(\sum |f(i)|^p\right)^{1/p} X$ where X = p-stable random var.
    - Stronger than reasoning with just expectation and variance!
    - NOTE: $\left(\sum |f(i)|^p\right)^{1/p} = \| f \|_p$ the Lp norm of f()

Extensions: Sketching with Stable Distributions [Ind00] *(cont.)*

- Use $O(\log(1/\delta)/\varepsilon^2)$ independent sketches with p-stable ξ's to approximate the Lp norm of the f()-stream ($\|f\|_p$) within ε with probability $\geq 1-\delta$
  - Use the samples of $\|f\|_p \Delta$ to estimate $\|f\|_p$
  - Works for any p $\in$(0,2] (extends [AMS96], where p=2)
  - Describe pseudo-random generator for the p-stable ξ's

- [CDI02] uses the same basic technique to estimate the Hamming (L0) norm over a stream
  - Hamming norm = *number of distinct values* in the stream
    - Hard estimation problem!
  - *Key observation:* Lp norm with p->0 gives good approximation to Hamming
    - Use p-stable sketches with very small p (e.g., 0.02)

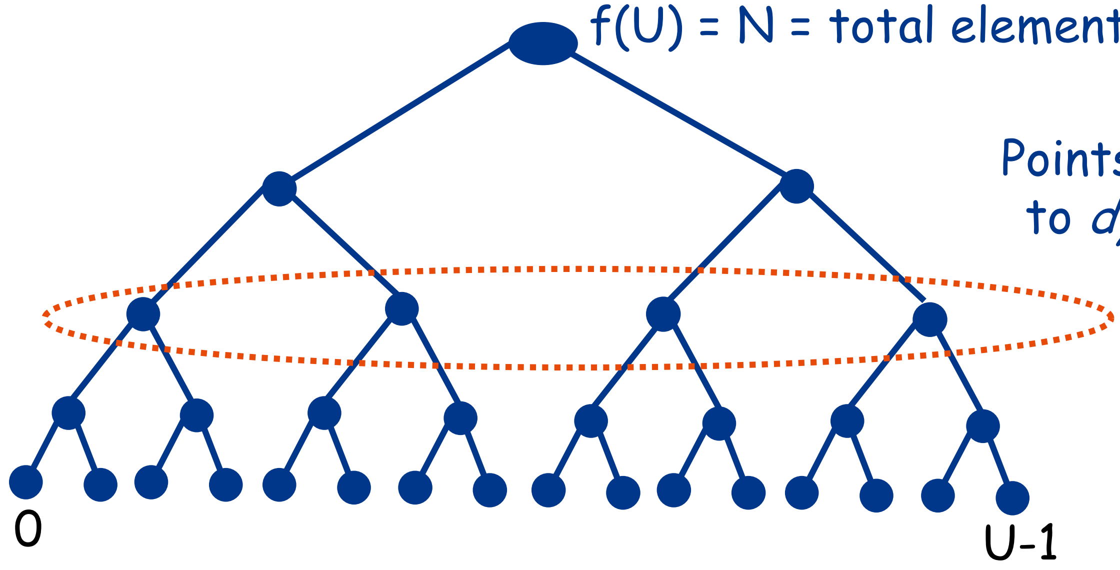
Key Benefit of Linear-Projection Summaries: Deletions!

- Straightforward to handle *item deletions* in the stream
  - To delete element i (f(i) = f(i) –1) simply subtract ξ_i from the running randomized linear projection estimate
  - Applies to all techniques described earlier

- [GKM02] use randomized linear projections for quantile estimation
  - *First* method to provide guaranteed-error quantiles in small space in the presence of general transactions (inserts + deletes)
  - Earlier techniques
    - Cannot be extended to handle deletions, or
    - Require re-scanning the data to obtain fresh sample

Random-Subset-Sums (RSSs) for Quantile Estimation [GKM02]

- *Key Idea:* Maintain frequency sums for random subsets of intervals at multiple resolutions



Points at different levels correspond to *dyadic intervals:* [k2^i, (k+1)2^i)

1 + log|U| levels

Random-Subset-Sum (RSS) Synopsis

- For each level j
  - Pick a random subset S of points (intervals): each point is chosen w/ prob. ½
  - Maintain the sum of all frequencies in S's intervals: $f(S) = \sum_{I \in S} f(I)$
  - Repeat to boost accuracy & confidence

Random-Subset-Sums (RSSs) for Quantile Estimation [GKM02] *(cont.)*

- Each RSS is a randomized linear projection of the frequency vector f()
  - ξ_i = 1 if i belongs in the union of intervals in S; 0 otherwise
- *Maintenance:* Insert/Delete element i
  - Find dyadic intervals containing i (check high-order bits of binary(i))
  - Update (+1/-1) all RSSs whose subsets contain these intervals
- Making it work in *small space & time*
  - Cannot explicitly maintain the random subsets S (O(|U|) space!)
  - Instead, use a O(log|U|) size seed and a pseudo-random function to determine each random subset S
    - pairwise independence amongst the members of S is sufficient
    - Membership can be tested in only O(log|U|) time

Random-Subset-Sums (RSSs) for Quantile Estimation [GKM02] *(cont.)*

Estimating f(I), I = interval

- *For a dyadic interval I:* Go to the appropriate level, and use the RSSs to compute the conditional expectation $E[f(S) \mid I \in S]$
  - Only use the maintained RSSs whose subset contains S (about half the RSSs at that level)
  - Note that: $E[f(S) \mid I \in S] = f(I) + \frac{1}{2} f(U - I) = \frac{1}{2} f(I) + \frac{N}{2}$
  - Use this expression to obtain an estimate for f(I)

- *For an arbitrary interval I:* Write I as the disjoint union of at most O(log|U|) dyadic intervals
  - Add up the estimates for all dyadic-interval components
  - Variance of the estimate increases by O(log|U|)

- Use averaging and median-selection over iid copies (as in [AMS96]) to boost accuracy and confidence

Random-Subset-Sums (RSSs) for Quantile Estimation [GKM02] *(cont.)*

Estimating approximate quantiles

- Want a value v such that: $f([0..v]) \in \phi N \pm \varepsilon N$
  - Use f(I) estimates in a *binary search* over the domain [0…U-1]

- *Theorem:* The RSS method computes an ε-approximate quantile over a stream of insertions/deletions with probability $\geq 1-\delta$ using space of $O(\log^2 |U| \cdot \log(\log |U| / \delta) / \varepsilon^2)$

- *First* technique to deal with general transaction streams
- RSS synopses are *composable*
  - Can be computed independently over different parts of the stream (e.g., in a distributed setting)
  - RSSs for the entire stream can be composed by simple summation
  - Another benefit of linear projections!!

More work on Sketches...

- Low-distortion vector-space embeddings (JL Lemma) [Ind01] and applications
  - E.g., approximate nearest neighbors [IM98]
- Discovering patterns and periodicities in time-series databases [IKM00, CIK02]
- Maintaining *top-k* item frequencies over a stream [CCF02]
- Data cleaning [DJM02]
- Other sketching references
  - Histogram/wavelet extraction [GGI02, GIM02]
  - Stream norm computation [FKS99]

Outline

- Introduction & motivation
  - Stream computation model, Applications
- Basic stream synopses computation
  - Samples, Equi-depth histograms, Wavelets
- Mining data streams
  - Decision trees, clustering
- Sketch-based computation techniques
  - Self-joins, Joins, Wavelets, V-optimal histograms
- Advanced techniques
  - Distinct values, Sliding windows
- Future directions & Conclusions

Distinct Value Estimation

- Problem: Find the number of distinct values in a stream of values with domain [0,...,N-1]
  - Zeroth frequency moment F_0 , L0 (Hamming) stream norm
  - Statistics: number of *species or classes* in a population
  - Important for query optimizers
  - *Network monitoring:* distinct destination IP addresses, source/destination pairs, requested URLs, etc.

- Example (N=8)

Data stream: $\boxed{3 \quad 0 \quad 5 \quad 3 \quad 0 \quad 1 \quad 7 \quad 5 \quad 1 \quad 0 \quad 3 \quad 7}$

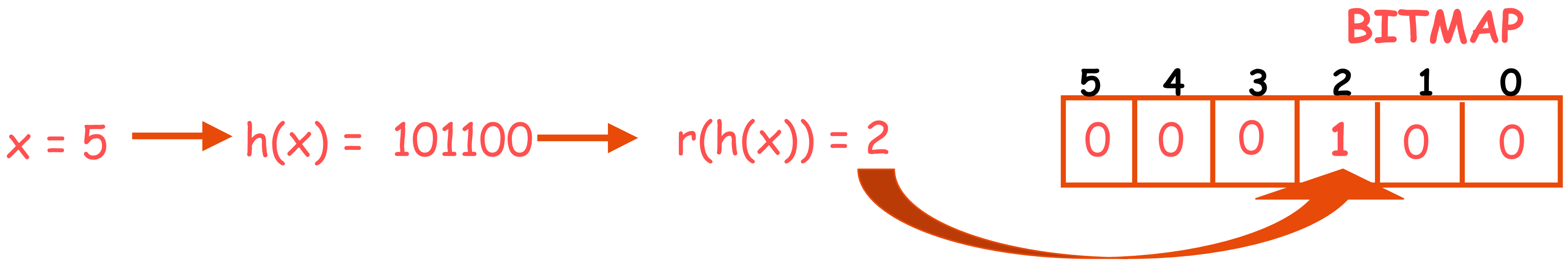
Number of distinct values: 5

Distinct Value Estimation

- **Uniform Sampling-based approaches**
  - Collect and store uniform random sample, apply an appropriate estimator
  - *Extensive* literature (see, e.g., [CCM00]) – hard problem for sampling!!
    - Many estimators proposed, but estimates are often inaccurate
    - [CCM00] proved must examine (sample) almost the entire table to guarantee the estimate is within a factor of 10 with probability > 1/2, regardless of the function used!
- ***One-pass approaches*** **(single scan + incremental maintenance)**
  - Hash functions to map domain values values to bit positions in a bitmap [FM85, AMS96]
  - Extension to handle predicates *("distinct values queries")* [Gib01]

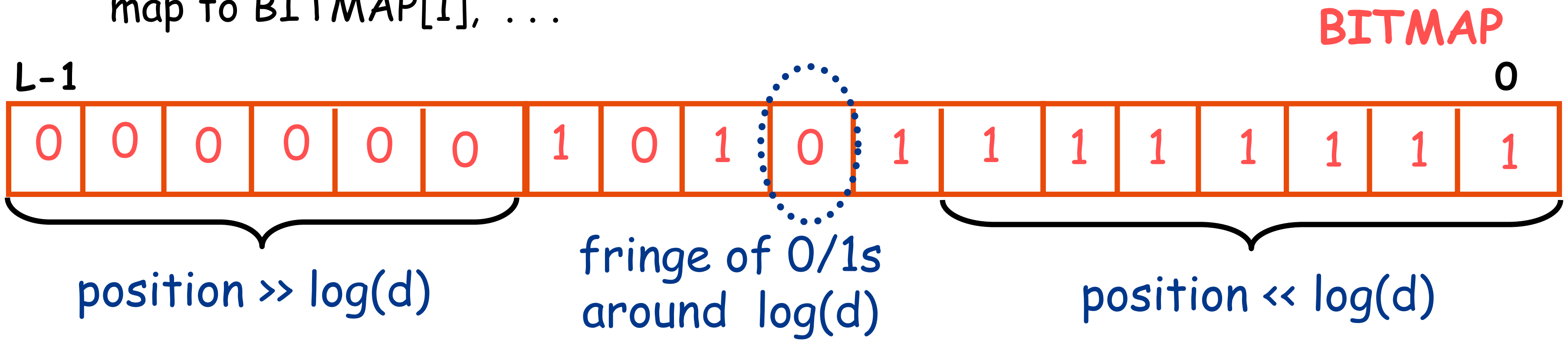
Distinct Value Estimation Using Hashing [FM85]

- Assume a hash function h(x) that maps incoming values x in [0,…, N-1] *uniformly* across [0,…, 2^L-1], where L = O(logN)
- Let r(y) denote the position of the least-significant 1 bit in the binary representation of y
  - A value x is mapped to r(h(x))
- We maintain a BITMAP array of L bits, initialized to 0
  - For each incoming value x, set BITMAP[r(h(x))] = 1



Distinct Value Estimation Using Hashing [FM85] *(cont.)*

- By uniformity through h(x): Prob[BITMAP[k]=1] = Prob[10^k] = $\frac{1}{2^{k+1}}$
  - Assuming d distinct values: expect d/2 to map to BITMAP[0] , d/4 to map to BITMAP[1], . . .



- Let R = position of rightmost zero in BITMAP
  - Use as indicator of log(d)
- [FM85] prove that E[R] = $\log(\phi d)$, where $\phi = .7735$
  - Estimate d = $2^R / \phi$
  - Averaging over several iid instances (different hash functions) to reduce estimator variance

Distinct Value Estimation

- [FM85] assume "ideal" hash functions h(x) (N-wise independence)
  - [AMS96] prove a similar result using simple linear hash functions (only pairwise independence)
    - h(x) = $(a \cdot x + b) \bmod N$, where a, b are random binary vectors in [0,…,2^L-1]
- [CDI02] Hamming norm estimation using p-stable sketching with p->0
  - Based on randomized linear projections $\Longrightarrow$ can readily handle deletions
  - Also, *composable:* Hamming norm estimation over *multiple* streams
    - E.g., number of positions where two streams differ

Generalization: Distinct Values Queries

- SELECT COUNT(DISTINCT target-attr)
- FROM relation
- WHERE predicate

Template

- SELECT COUNT(DISTINCT o_custkey)
- FROM orders
- WHERE o_orderdate >= '2002-01-01'

TPC-H example

  - "How many distinct customers have placed orders this year?"
  - Predicate not necessarily only on the DISTINCT target attribute

- *Approximate answers with error guarantees over a stream of tuples?*

Distinct Sampling [Gib01]

Key Ideas

- Use FM-like technique to collect a specially-tailored sample over the *distinct values in the stream*
  - Uniform random sample of the distinct values
  - Very different from traditional URS: each distinct value is chosen uniformly regardless of its frequency
  - DISTINCT query answers: simply scale up sample answer by sampling rate

- To handle additional predicates
  - *Reservoir sampling* of tuples for each distinct value in the sample
  - Use reservoir sample to evaluate predicates

Building a Distinct Sample [Gib01]

- Use FM-like hash function h() for each streaming value x
  - Prob[h(x) = k] = $\frac{1}{2^{k+1}}$
- *Key Invariant:* *"All values with h(x) >= level (and only these) are in the distinct sample"*

```
DistinctSampling( B , r )
// B = space bound,  r = tuple-reservoir size for each distinct value
level = 0;  S = $\phi$
for each new tuple t do
    let x = value of DISTINCT target attribute in t
    if  h(x) >= level  then    // x belongs in the distinct sample
        use  t  to update the reservoir sample of tuples for x
    if  |S| >= B then   // out of space
        evict from S all  tuples with  h(target-attribute-value) = level
        set  level = level + 1
```

Using the Distinct Sample [Gib01]

- If level = l for our sample, then we have selected all distinct values x such that h(x) >= l
  - Prob[h(x) >= l] = $\frac{1}{2^l}$
  - By h()'s randomizing properties, we have uniformly sampled a 2^{-l} fraction of the distinct values in our stream

Our sampling rate!

- *Query Answering:* Run distinct-values query on the distinct sample and scale the result up by 2^l
- *Distinct-value estimation:* Guarantee ε relative error with probability 1 - δ using O(log(1/δ)/ε^2) space
  - For q% selectivity predicates the space goes up inversely with q
- *Experimental results:* 0-10% error vs. 50-250% error for previous best approaches, using 0.2% to 10% synopses

Distinct Sampling Example

- B=3, N=8 (r = 0 to simplify example)

Data stream: 3 0 5 3 0 1 7 5 1 0 3 7

hash:

0	1	3	5	7
0	1	0	1	0

Data stream: 1 7 5 1 0 3 7

S={3,0,5}, level = 0

$\Downarrow$

S={1,5}, level = 1

- Computed value: 4

Sliding Window Model

- Model
  - At every time t, a data record arrives
  - The record "expires" at time t+N (N is the window length)
- When is it useful?
  - Make decisions based on "recently observed" data
  - Stock data
  - Sensor networks

Remark: Data Stream Models

Tuples arrive $X_1, X_2, X_3, \ldots, X_t, \ldots$

- Function f(X,t,NOW)
  - Input at time t: $f(X_1,1,t), f(X_2,2,t). f(X_3,3,t), \ldots, f(X_t,t,t)$
  - Input at time t+1: $f(X_1,1,t+1), f(X_2,2,t+). f(X_3,3,t+1), \ldots, f(X_{t+1},t+1,t+1)$
- Full history: F == identity
- Partial history: Decay
  - Exponential decay: $f(X,t, NOW) = 2^{-(NOW-t)}*X$
    - Input at time t: $2^{-(t-1)}*X_1, 2^{-(t-2)}*X_{2,}, \ldots, \frac{1}{2} * X_{t-1},X_t$
    - Input at time t+1: $2^{-t}*X_1, 2^{-(t-1)}*X_{2,}, \ldots, 1/4 * X_{t-1}, \frac{1}{2} *X_{t,} X_{t+1}$
  - Sliding window (special type of decay):
    - $f(X,t,NOW) = X$ if $NOW-t < N$
    - $f(X,t,NOW) = 0$, otherwise
    - Input at time t: $X_1, X_2, X_3, \ldots, X_t$
    - Input at time t+1: $X_2, X_3, \ldots, X_{t,} X_{t+1,}$

Simple Example: Maintain Max

- Problem: Maintain the maximum value over the last N numbers.
- Consider all non-decreasing arrangements of N numbers (Domain size R):
  - There are ((N+R) choose N) arrangement
  - Lower bound on memory required:
    log(N+R choose N) >= N*log(R/N)
  - So if R=poly(N), then lower bound says that we have to store the last N elements (Ω(N log N) memory)

Statistics Over Sliding Windows

- Bitstream: Count the number of ones [DGIM02]
  - Exact solution: $\Theta(N)$ bits
  - Algorithm BasicCounting:
    - $1 + \varepsilon$ approximation (relative error!)
    - Space: $O(1/\varepsilon\ (\log^2 N))$ bits
    - Time: $O(\log N)$ worst case, $O(1)$ amortized per record
  - Lower Bound:
    - Space: $\Omega(1/\varepsilon\ (\log^2 N))$ bits

Approach 1: Temporal Histogram

Example: … 01101010011111110110 0101 …

Equi-width histogram:

… 0110 1010 0111 1111 0110 0101 …

- Issues:
  - Error is in the last (leftmost) bucket.
  - Bucket counts (left to right): $C_m, C_{m-1}, \ldots, C_2, C_1$
  - Absolute error <= $C_m/2$.
  - Answer >= $C_{m-1}+\ldots+C_2+C_1+1$.
  - Relative error <= $C_m/2(C_{m-1}+\ldots+C_2+C_1+1)$.
  - Maintain: $C_m/2(C_{m-1}+\ldots+C_2+C_1+1) <= \varepsilon$ (=1/k).

Naïve: Equi-Width Histograms

- Goal: Maintain $C_m/2 <= \varepsilon\ (C_{m-1}+...+C_2+C_1+1)$

Problem case:

... 0110 1010 0111 1111 0110 1111 0000 0000 0000 0000 ...

- Note:
  - Every Bucket will be the last bucket sometime!
  - New records may be all zeros Ł

    For **every** bucket i, require $C_i/2 <= \varepsilon\ (C_{i-1}+...+C_2+C_1+1)$

Exponential Histograms

- Data structure invariant:
  - Bucket sizes are non-decreasing powers of 2
  - For every bucket other than the last bucket, there are at least k/2 and at most k/2+1 buckets of that size
  - Example: k=4: (1,1,2,2,2,4,4,4,8,8,..)
- Invariant implies:
  - Case 1: $C_i > C_i$-1: $C_i=2^j$, $C_{i-1}=2^{j-1}$
    $C_{i-1}+\ldots+C_2+C_1+1 >= k*(\Sigma(1+2+4+..+2^{j-1})) >= k*2^j >= k*C_i$
  - Case 2: $C_i = C_i$-1: $C_i=2^j$, $C_{i-1}=2^j$
    $C_{i-1}+\ldots+C_2+C_1+1 >= k*(\Sigma(1+2+4+..+2^{j-1})) + 2^j >= k*2^j/2 >= k*C_i/2$

Complexity

- Number of buckets m:
  - m <= [# of buckets of size j]*[# of different bucket sizes]

    <= (k/2 +1) * ((log(2N/k)+1) = O(k* log(N))
- Each bucket requires O(log N) bits.
- Total memory:

  $O(k \log^2 N) = O(1/\varepsilon * \log^2 N)$ bits
- Invariant maintains error guarantee!

Algorithm

Data structures:

- For each bucket: timestamp of most recent 1, size
- LAST: size of the last bucket
- TOTAL: Total size of the buckets

New element arrives at time t

- If last bucket expired, update LAST and TOTAL
- If (element == 1)
  Create new bucket with size 1; update TOTAL
- Merge buckets if there are more than k/2+2 buckets of the same size
- Update LAST if changed

Anytime estimate: TOTAL – (LAST/2)

Example Run

- If last bucket expired, update LAST and TOTAL
- If (element == 1)

  Create new bucket with size 1; update TOTAL
- Merge buckets if there are more than k/2+2 buckets of the same size
- Update LAST if changed

32,16,8,8,4,4,2,1,1

32,16,8,8,4,4,2,2,1

32,16,8,8,4,4,2,2,1,1

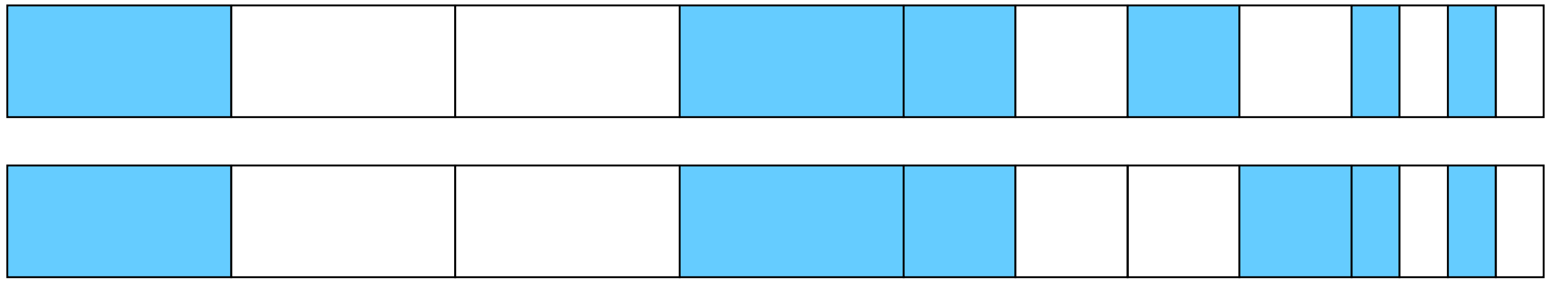
32,16,16,8,4,2,1

Lower Bound

- Argument: Count number of different arrangements that the algorithm needs to distinguish
  - log(N/B) blocks of sizes B,2B,4B,…,2^iB from right to left.
  - Block i is subdivided into B blocks of size 2^i each.
  - For each block (independently) choose k/4 sub-blocks and fill them with 1.
- Within each block: (B choose k/4) ways to place the 1s
- (B choose k/4)$^{\log(N/B)}$ distinct arrangements

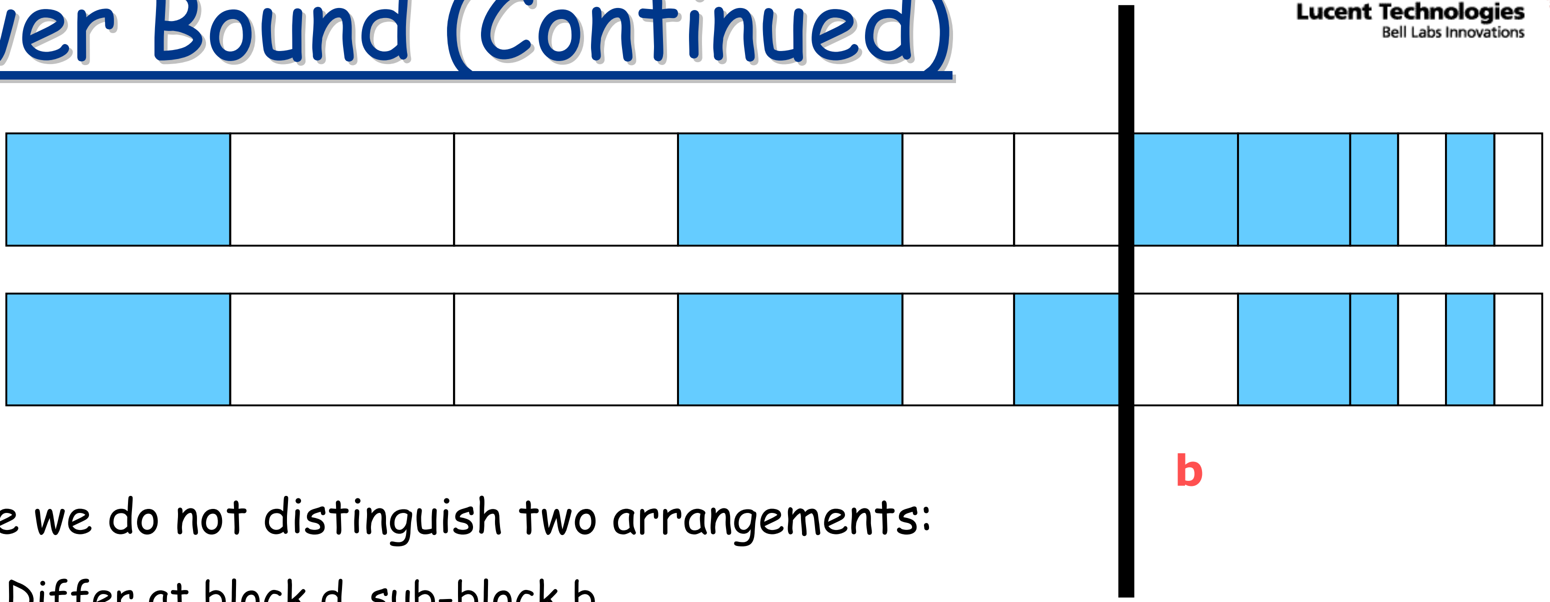
Lower Bound (Continued)

- Example:



- Show: An algorithm has to distinguish between any such two arrangements

Lower Bound (Continued)



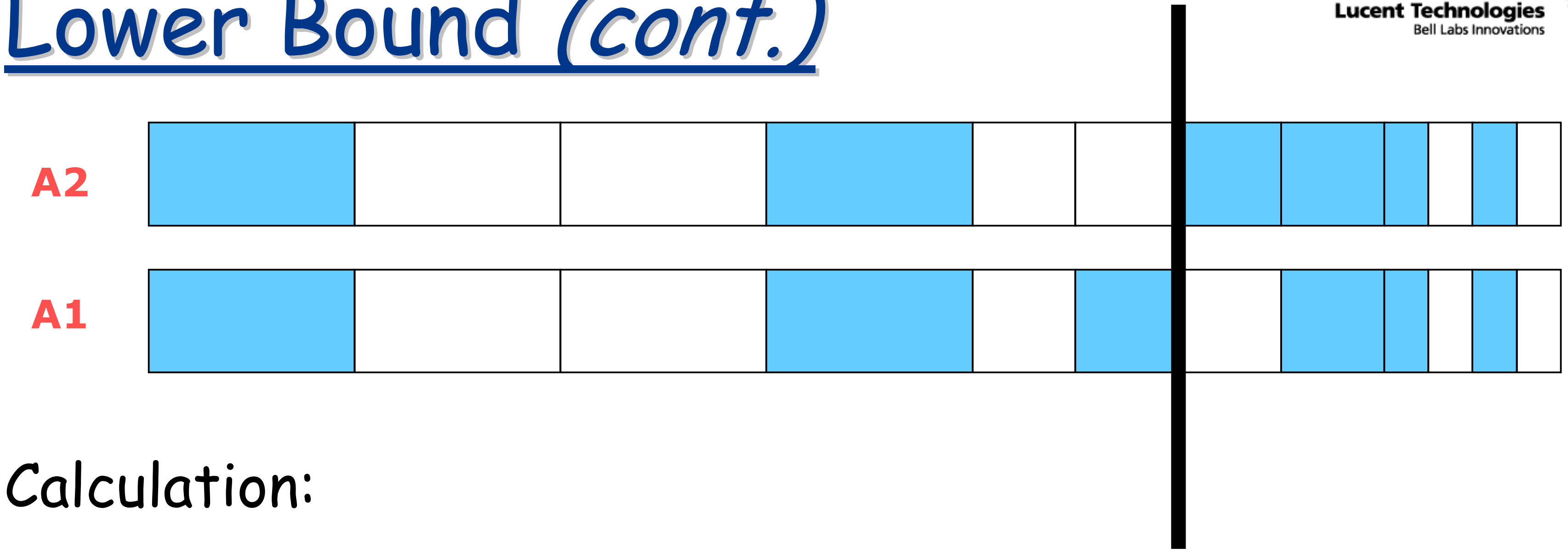
Assume we do not distinguish two arrangements:

- Differ at block d, sub-block b

Consider time when b expires

- We have c full sub-blocks in A1, and c+1 full sub-blocks in A2 [note: c+1<=k/4]
- A1: $c2^d$+sum1 to d-1 k/4*$(1+2+4+..+2^{d-1})$
  $= c2^d+k/2*(2^d-1)$
- A2: $(c+1)2^d+k/4*(2^d-1)$
- Absolute error: 2^{d-1}
- Relative error for A2:
  $2^{d-1}/[(c+1)2^d+k/4*(2^d-1)] >= 1/k = \varepsilon$

Lower Bound (cont.)



Calculation:

- A1: $c2^d$+sum1 to d-1 k/4*($1+2+4+..+2^{d-1}$)
  = $c2^d + k/2*(2^d-1)$
- A2: $(c+1)2^d + k/4*(2^d-1)$
- Absolute error: 2^{d-1}
- Relative error:
  $2^{d-1}/[(c+1)2^d+k/4*(2^d-1)]$ >=
  $2^{d-1}/[2*k/4*\ 2^d] = 1/k = \varepsilon$

More Sliding Window Results

- Maintain the sum of last N positive integers in range {0,…,R}.
- Results:
  - $1 + \varepsilon$ approximation.
  - $1/\varepsilon(\log N)(\log N + \log R)$ **bits**.
  - $O(\log R/\log N)$ amortized, $(\log N + \log R)$ worst case.
- Lower Bound:
  - $1/\varepsilon(\log N)(\log N + \log R)$ **bits**.
- Variance
- Clusters

Outline

- Introduction & motivation
  - Stream computation model, Applications
- Basic stream synopses computation
  - Samples, Equi-depth histograms, Wavelets
- Mining data streams
  - Decision trees, clustering
- Sketch-based computation techniques
  - Self-joins, Joins, Wavelets, V-optimal histograms
- Advanced techniques
  - Distinct values, Sliding windows
- Future directions & Conclusions

Future Research Directions

Three favorite problems; generic laundry list follows:

- Appropriate "stream algebra" (operators + composition rules)
  - Progress: Aurora, Telegraph, STREAM
- Lower bounds & tradeoffs for data-streaming problems
  - E.g., no. of passes vs. space requirements ("p passes $\Rightarrow$ f(N,p) space")
- Making sketches ready for prime-time
  - Approximating *set-valued* query results
  - *Multiple* standing queries
  - Beyond relational tuples and numeric attributes
  - Most appropriate sketching technique for incorporation in DBMSs?

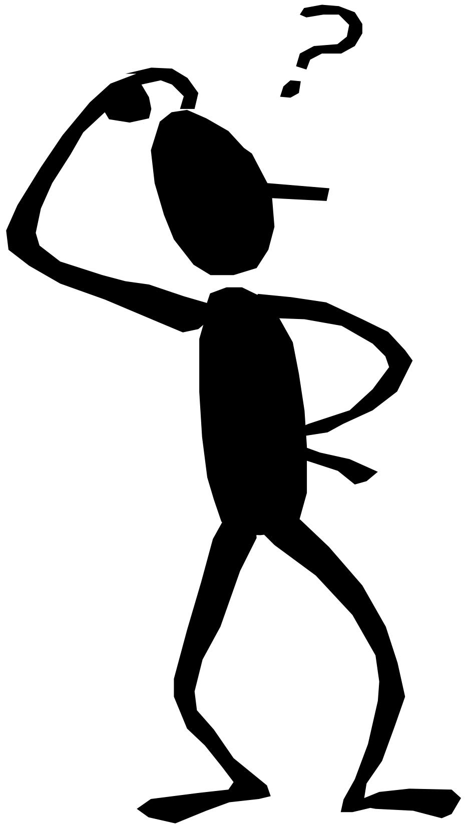
Data Streaming - Future Research Laundry List

- Stream processing system architectures
- Memory management for stream processing
- Integration of stream processing and databases
- Stream indexing, searching, and similarity matching
- Exploiting prior knowledge for stream computation
- User-interface issues
  - Exposing approximation model to the user
- Content-based routing, filtering, and correlation of XML data streams
- Novel stream processing applications
  - Sensor networks, financial analysis, etc.

Conclusions

- Querying and finding patterns in massive streams is a real problem with many "real-world" applications
- Fundamentally rethink data-management issues under stringent constraints
  - Single-pass algorithms with limited memory resources
- A lot of progress in the last few years
  - Algorithms, system models & architectures
    - Aurora (Brandeis/Brown/MIT)
    - Niagara (Wisconsin)
    - STREAM (Stanford)
    - Telegraph (Berkeley)
- Commercial acceptance still lagging, but will most probably grow in coming years
  - Specialized systems (e.g., fraud detection), but still far from "DSMSs"
- Great Promise: *Still lots of interesting research to be done!!*

Thank you!



- Updated slides & references available from

http://www.bell-labs.com/~{minos, rastogi}

http://www.cs.cornell.edu/johannes/

References (1)

- [AGM99] N. Alon, P.B. Gibbons, Y. Matias, M. Szegedy. Tracking Join and Self-Join Sizes in Limited Storage. ACM PODS, 1999.
- [AMS96] N. Alon, Y. Matias, M. Szegedy. The space complexity of approximating the frequency moments. ACM STOC, 1996.
- [CIK02] G. Cormode, P. Indyk, N. Koudas, S. Muthukrishnan. Fast mining of tabular data via approximate distance computations. IEEE ICDE, 2002.
- [CMN98] S. Chaudhuri, R. Motwani, and V. Narasayya. "Random Sampling for Histogram Construction: How much is enough?". ACM SIGMOD 1998.
- [CDI02] G. Cormode, M. Datar, P. Indyk, S. Muthukrishnan. Comparing Data Streams Using Hamming Norms. VLDB, 2002.
- [DGG02] A. Dobra, M. Garofalakis, J. Gehrke, R. Rastogi. Processing Complex Aggregate Queries over Data Streams. ACM SIGMOD, 2002.
- [DJM02] T. Dasu, T. Johnson, S. Muthukrishnan, V. Shkapenyuk. Mining database structure or how to build a data quality browser. ACM SIGMOD, 2002.
- [DH00] P. Domingos and G. Hulten. Mining high-speed data streams. ACM SIGKDD, 2000.
- [EKSWX98] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental Clustering for Mining in a Data Warehousing Environment. VLDB 1998.
- [FKS99] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan. An approximate L1-difference algorithm for massive data streams. IEEE FOCS, 1999.
- [FM85] P. Flajolet, G.N. Martin. "Probabilistic Counting Algorithms for Data Base Applications". JCSS 31(2), 1985.

References (2)

- [Gib01] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports, VLDB 2001.
- [GGI02] A.C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. ACM STOC, 2002.
- [GGRL99] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh: BOAT-Optimistic Decision Tree Construction. SIGMOD 1999.
- [GK01] M. Greenwald and S. Khanna. "Space-Efficient Online Computation of Quantile Summaries". ACM SIGMOD 2001.
- [GKM01] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, M. Strauss. Surfing Wavelets on Streams: One Pass Summaries for Approximate Aggregate Queries. VLDB 2001.
- [GKM02] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, M. Strauss. "How to Summarize the Universe: Dynamic Maintenance of Quantiles". VLDB 2002.
- [GKS01b] S. Guha, N. Koudas, and K. Shim. "Data Streams and Histograms". ACM STOC 2001.
- [GM98] P. B. Gibbons and Y. Matias. "New Sampling-Based Summary Statistics for Improving Approximate Query Answers". ACM SIGMOD 1998.
  - Proposes the "concise sample" and "counting sample" techniques for improving the accuracy of sampling-based estimation for a given amount of space for the sample synopsis.
- [GMP97] P. B. Gibbons, Y. Matias, and V. Poosala. "Fast Incremental Maintenance of Approximate Histograms". VLDB 1997.
- [GT01] P.B. Gibbons, S. Tirthapura. "Estimating Simple Functions on the Union of Data Streams". ACM SPAA, 2001.

References (3)

- [HHW97] J. M. Hellerstein, P. J. Haas, and H. J. Wang. “Online Aggregation”. ACM SIGMOD 1997.
- [HSD01] Mining Time-Changing Data Streams. G. Hulten, L. Spencer, and P. Domingos. ACM SIGKD 2001.
- [IKM00] P. Indyk, N. Koudas, S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. VLDB, 2000.
- [Ind00] P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings, and Data Stream Computation. IEEE FOCS, 2000.
- [IP95] Y. Ioannidis and V. Poosala. “Balancing Histogram Optimality and Practicality for Query Result Size Estimation”. ACM SIGMOD 1995.
- [IP99] Y.E. Ioannidis and V. Poosala. “Histogram-Based Approximation of Set-Valued Query Answers”. VLDB 1999.
- [JKM98] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. “Optimal Histograms with Quality Guarantees”. VLDB 1998.
- [JL84] W.B. Johnson, J. Lindenstrauss. Extensions of Lipshitz Mapping into Hilbert space. Contemporary Mathematics, 26, 1984.
- [Koo80] R. P. Kooi. “The Optimization of Queries in Relational Databases”. PhD thesis, Case Western Reserve University, 1980.
- [MRL98] G.S. Manku, S. Rajagopalan, and B. G. Lindsay. “Approximate Medians and other Quantiles in One Pass and with Limited Memory”. ACM SIGMOD 1998.
- [MRL99] G.S. Manku, S. Rajagopalan, B.G. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. ACM SIGMOD, 1999.

References (4)

- [MVW98] Y. Matias, J.S. Vitter, and M. Wang. "Wavelet-based Histograms for Selectivity Estimation". ACM SIGMOD 1998.
- [MVW00] Y. Matias, J.S. Vitter, and M. Wang. "Dynamic Maintenance of Wavelet-based Histograms". VLDB 2000.
- [PIH96] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. "Improved Histograms for Selectivity Estimation of Range Predicates". ACM SIGMOD 1996.
- [PJO99] F. Provost, D. Jenson, and T. Oates. Efficient Progressive Sampling. KDD 1999.
- [Poo97] V. Poosala. "Histogram-Based Estimation Techniques in Database Systems". PhD Thesis, Univ. of Wisconsin, 1997.
- [PSC84] G. Piatetsky-Shapiro and C. Connell. "Accurate Estimation of the Number of Tuples Satisfying a Condition". ACM SIGMOD 1984.
- [SDS96] E.J. Stollnitz, T.D. DeRose, and D.H. Salesin. "Wavelets for Computer Graphics". Morgan-Kauffman Publishers Inc., 1996.
- [T96] H. Toivonen. Sampling Large Databases for Association Rules. VLDB 1996.
- [TGI02] N. Thaper, S. Guha, P. Indyk, N. Koudas. Dynamic Multidimensional Histograms. ACM SIGMOD, 2002.
- [U89] P. E. Utgoff. Incremental Induction of Decision Trees. Machine Learning, 4, 1989.
- [U94] P. E. Utgoff: An Improved Algorithm for Incremental Induction of Decision Trees. ICML 1994.
- [Vit85] J. S. Vitter. "Random Sampling with a Reservoir". ACM TOMS, 1985.

This is only a partial list of references on Data Streaming. Further important references can be found, e.g., in the proceedings of KDD, SIGMOD, PODS, VLDB, ICDE, STOC, FOCS, and other conferences or journals, as well as in the reference lists given in the above papers.