Chapter 6:

Stream Applications & Algorithms

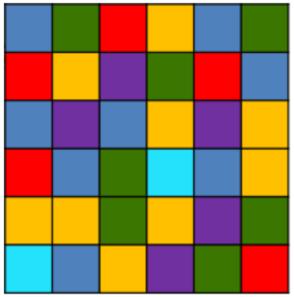
Today's Lesson

Stream Applications and Algorithms

- Maintaining Histograms
- Change Detection
- Clustering
- Frequent Itemset Mining

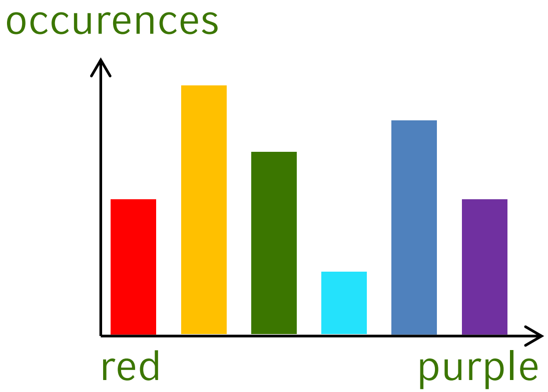
Maintaining Histograms

- Histograms are *graphical representations* of the distribution of numerical data



- Histograms estimate the probability distribution of a random variable

- Used for approximative query answering with error guarantees



Maintaining Histograms

- Histograms are defined by non-overlapping intervals
- An interval is defined by its boundaries and its frequency count
- In case of streams:
  One never observes all values of a random variable

→ K-bucket histogram defined as
$]-\infty, b_1],]b_1, b_2], \dots,]b_{k-1}, \infty[$ buckets with frequency counts $f_1, f_2, \dots, f_k$

Maintaining Histograms

In general: two types of histogram maintanence techniques

1. *Equal-width* histograms:
   The range of observed values is divided into equi-sized intervals $(\forall i,j\colon (b_i, b_{i+1}) = (b_j, b_{j+1}))$

2. *Equal-frequency* histograms:
   The range of observed values is divided into k intervals such that the counts in each interval are equal
   $(\forall i,j\colon (f_i = f_j))$

Maintaining Histograms

K-buckets Histograms (Gibbons et al., 1997)

- Incremental maintenance of histograms applicable for *Insert-Delete Models*

- Setting: Pre-defined number of intervals k and continuously occuring inserts and deletes as given in a sliding window approach

- Histogram maintenance based on two operations
  - Split & Merge Operation
  - Merge & Split Operation

Maintaining Histograms

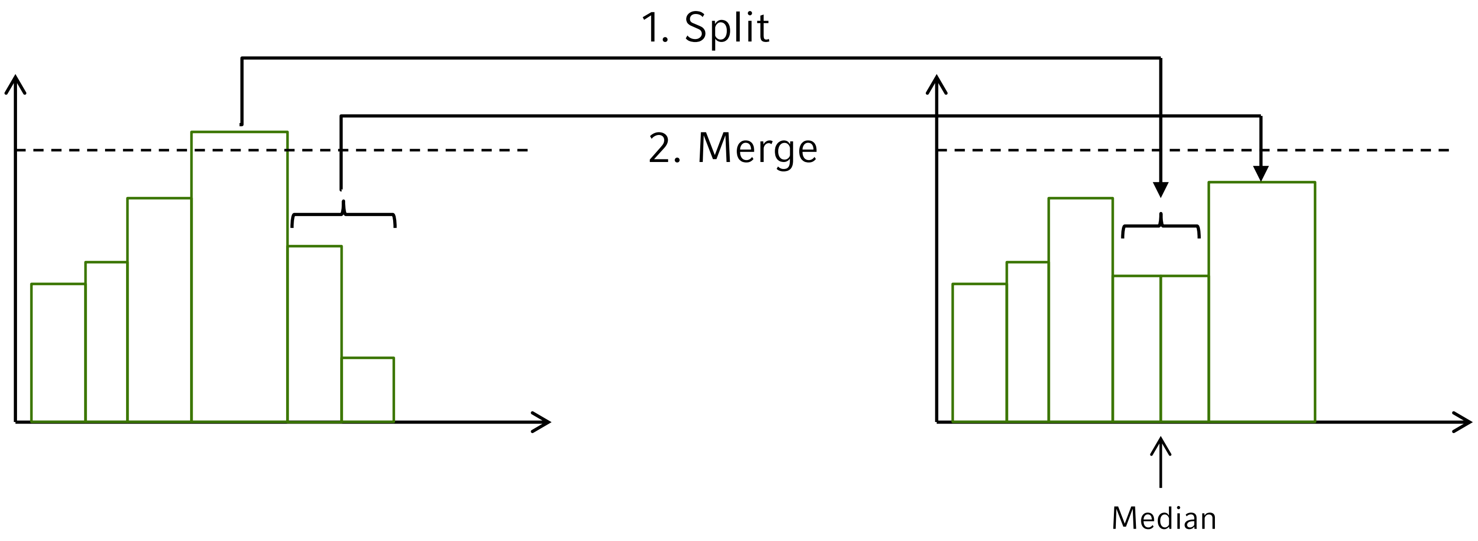
K-buckets Histograms (Gibbons et al., 1997)

1. *Split & Merge Operation*:
   - Occurs with inserts
   - Triggered whenever the count in a bucket is greater than a given threshold
   - Split overflowed bucket into two and merge two consecutive buckets

Maintaining Histograms

K-buckets Histograms (Gibbons et al., 1997)

1. *Split & Merge Operation*:



Maintaining Histograms

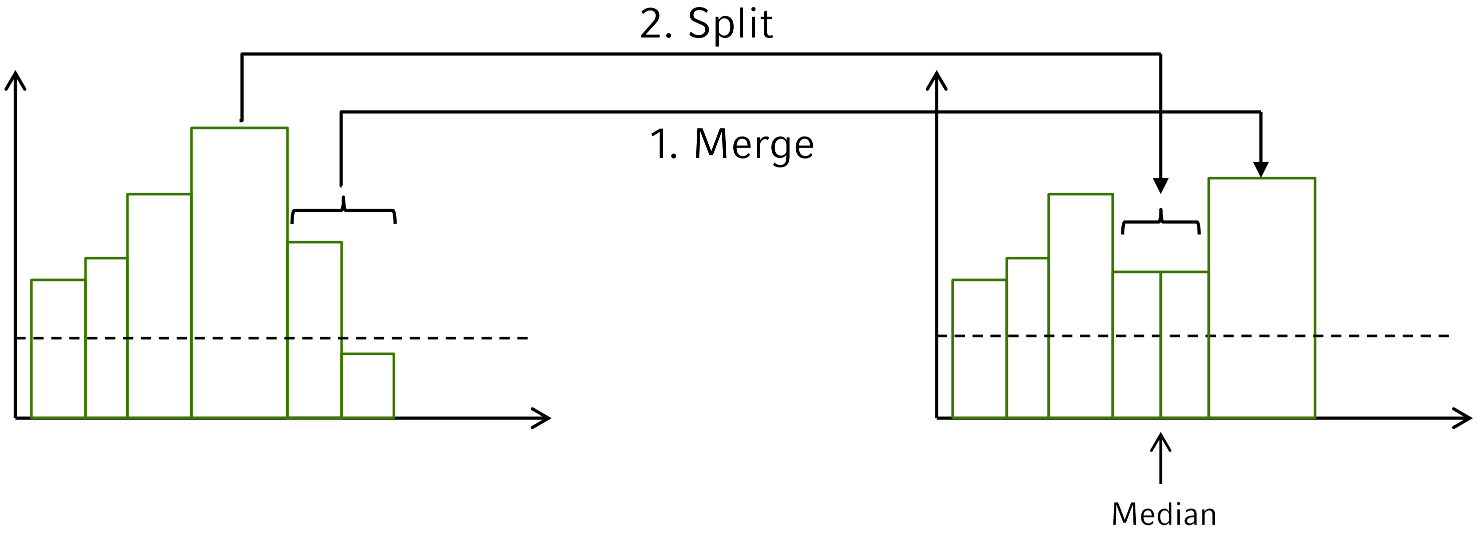
K-buckets Histograms (Gibbons et al., 1997)

1. *Merge & Split Operation*:
   - Occurs with deletes
   - Triggered whenever the count in a bucket is below a given threshold
   - Merge underflowed bucket with a neighbor bucket and split the bucket with the highest count

Maintaining Histograms

K-buckets Histograms (Gibbons et al., 1997)

1. *Merge & Split Operation*:



Maintaining Histograms

Exponential Histograms (Datar et al., 2002)

- Used to solve counting problems
- Considers simplified data streams that consist of 0 and 1 elements
- Aims at counting the number of 1's (like interesting events) within a sliding window of size N

Maintaining Histograms

Exponential Histograms (Datar et al., 2002)

- Unequal bucket sizes and interval sizes
- Needs only $O(\log(N))$ space with N being the size of the sliding window
- Each bucket consists of $size$ and $timestamp$
- Uses two additional variables, i.e. $LAST$ and $TOTAL$, to estimate the number of elements in the sliding window

Maintaining Histograms

Exponential Histograms (Datar et al., 2002)

Algorithm Exponential Histogram Maintenance
Input: data stream S, window size N, error param. ϵ
begin
  $TOTAL \coloneqq 0$
  $LAST \coloneqq 0$
  while S **do**
    $x_i \coloneqq S.next$
    if $x_i == 1$ **do**
      create new bucket b_i with timestamp t_i
      $TOTAL \mathrel{+}= 1$
      while $t_l < t_i - N.length$ **do**
        $TOTAL \mathrel{-}= b_l.size$
        drop the oldest bucket b_l
        $b_l \coloneqq b_{l-1}$
        $LAST \coloneqq b_l.size$
      while *exist* $|1/\epsilon|/2 + 2$ *buckets of the same size* **do**
        merge the two oldest buckets of the same size with the largest timestamp of both buckets
        if *last bucket was merged* **do**
          $LAST \coloneqq$ size of the new created last bucket
end

Maintaining Histograms

Exponential Histograms (Datar et al., 2002)

Algorithm Exponential Histogram Maintenance
Input: data stream S, window size N, error param. ϵ
begin
  $TOTAL \coloneqq 0$
  $LAST \coloneqq 0$
  while S **do**
    $x_i \coloneqq S.next$
    if $x_i == 1$ **do**
      create new bucket b_i with timestamp t_i
      $TOTAL \mathrel{+}= 1$
      while $t_l < t_i - N.length$ **do**
        $TOTAL \mathrel{-}= b_l.size$
        drop the oldest bucket b_l
        $b_l \coloneqq b_{l-1}$
        $LAST \coloneqq b_l.size$
      while *exist* $|1/\epsilon|/2 + 2$ *buckets of the same size* **do**
        merge the two oldest buckets of the same size with the largest timestamp of both buckets
        if *last bucket was merged* **do**
          $LAST \coloneqq$ size of the new created last bucket
end

Algorithm Exponential Histogram Count Estimation
Input: current Exponential Histogram EH
Output: estimate number of 1's within $EH.N$
begin
  return $\mathrm{EH.TOTAL} - \mathrm{EH.LAST}/2$
end

Change Detection

General Assumptions:

- For static datasets:
  - Data generated by a fixed process
  - Data is a sample of a fixed distribution

- For data streams:
  - Additional temporal dimension
  - Underlying process can change over time

→ Challenge: Detection and quantification of changes

Change Detection

Impact of changes on data processing algorithms:

- Data Mining:
  Data that arrived before a change can bias the model due to characteristics that no longer hold after the change

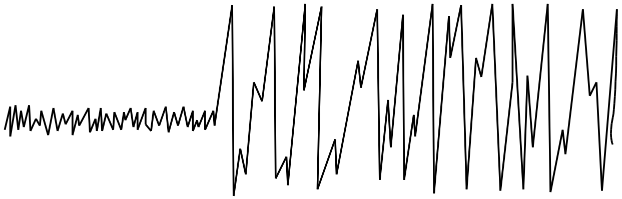
- Query processing:
  Query answers for time intervals with stable underlying data distributions might be more meaningful

Change Detection

The nature of changes

- Concept Drifts:
  Gradual change in target concept

- Concept Shifts:
  Abrupt change in target concept



Change Detection

Two general approaches

- Monitoring the evolution of performance indicators (Klinkenberg et al., 1998), e.g.
  - Accuracy of the current classifier
  - Attribute value distribution
  - Monitoring *top* attributes (according to any ranking)
- Monitoring distribution on two different time-windows

Change Detection

CUSUM Algorithm (Page, 1954)

- Monitors the **cu**mulative **sum** of instances of a random variable

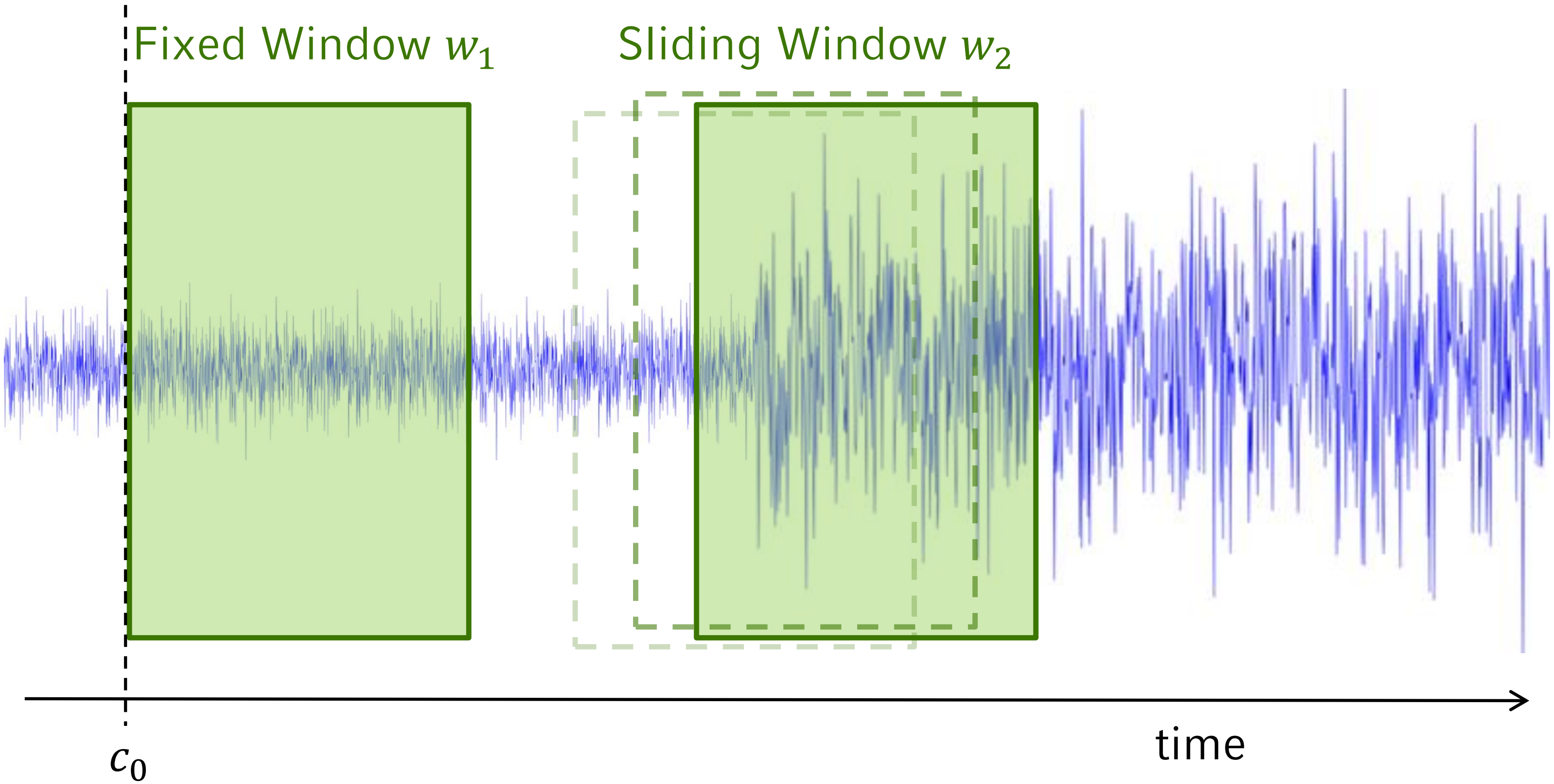
- Detects a change if the (normalized) mean of the input data is significantly different to zero, resp. to the estimated mean

Algorithm CUSUM
Input: data stream S, threshold param. α
begin
  $G_0 := 0$
  while S **do**
  $x_t :=$ next instance of S
  compute estimated mean ω_t
  $G_t := \max(0, G_{t-1} - \omega_t + x_t)$
  if $G_t > \alpha$ **then**
    report change at time t
    $G_t := 0$
end

- ω_t commonly represents the likelihood function

Change Detection

Two Windows Approach (Kifer et al., 2004)



Change Detection

Two Windows Approach (Kifer et al., 2004)

Algorithm Two Windows Approach
Input: data stream S, window sizes m_1 and m_2, distance func. $d: D \times D \rightarrow R$, threshold param. α
begin
  $c_0 \coloneqq 0$
  $W_1 \coloneqq$ first m_1 points from time c_0
  $W_2 \coloneqq$ most recent m_2 points from S
  while S **do**
    slide W_2 by 1 point
    if $d(W_1, W_2) > \alpha$ **then**
      $c_0 \coloneqq$ current time
      report change at time c_0
      $W_1 \coloneqq$ first m_1 points from time c_0
      $W_2 \coloneqq$ most recent m_2 points from S
end

d measures the distance between two probability distributions

Clustering from Data Streams

Clustering is the process of grouping objects into different groups, such that the similarity of data in each subset is high, and between different subsets is low.

Clustering from data streams aims at maintaining a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time.

Clustering from Data Streams

General approaches to clustering

- *Partitioning*: Fixed number of clusters, new object is assigned to closest cluster center (k-means/k-medoid)
- *Density-based*: Take connectivity and density functions into account (DBSCAN)
- *Hierarchical*: Find a tree-like structure representing the hierarchy of the cluster model (Single Link/Complete Link)
- *Grid-based*: Partition the space into grid cells (STING)
- *Model-based*: Take a model and find the best fit clustering (COBWEB)

Clustering from Data Streams

Requirements for stream clustering algorithms

- Compactness of representation
- Fast, incremental processing (one-pass)
- Tracking cluster changes (as clusters might (dis-)appear over time)
- Clear and fast identification of outliers

Clustering from Data Streams

LEADER algorithm (Spath, 1980)

- Simplest form of partitioning based clustering applicable to data streams
- Depends on the order of incoming objects
- Depends on a good choice of the threshold parameter δ

```
Algorithm LEADER
Input: data stream S, threshold param. δ
begin
  while S do
    x_i := next object from S
    find closest cluster c_clos to x_i
    if d(c_clos, x_i) < δ then
      assign x_i to c_clos
    else
      create new cluster with x_i
end
```

Clustering from Data Streams

Stream K-means (O'Callaghan et al., 2002)

- Partition data stream S into chunks $X_1, \dots, X_n, \dots$ so that each chunk fits in memory
- Apply k-means for each chunk X_i and retrieve k cluster centers each weighted with the number of points it compresses
- Apply k-means on the cluster centers to get an overall k-means clustering when demanded

Clustering from Data Streams

Microcluster-based Clustering

- Common approach to capture temporal information for being able to deal with cluster evolution

- A *microcluster* (or *cluster feature CF*) is a triple (N, LS, SS) that stores the sufficient information of a set of points
  - N is the number of points
  - LS is the linear sum of the N points, i.e. $\sum_{i=1}^{N} \overrightarrow{x_i}$
  - SS is the square sum of the N points, i.e. $\sum_{i=1}^{N} \overrightarrow{x_i}^2$

Clustering from Data Streams

Microcluster-based Clustering

- The properties of cluster features are:
  - Incrementality:
    $\mathrm{N_i} = \mathrm{N_i} + 1, \quad LS_i = LS_i + \vec{x}, \quad SS_i = SS_i + \vec{x}^2$
  - Additivity:
    $\mathrm{N_k} = \mathrm{N_i} + \mathrm{N_j}, \quad LS_k = LS_i + LS_j, \quad SS_k = SS_i + SS_j$
  - Centroid: $\overrightarrow{X_c} = \frac{LS_i}{N}$
  - Radius: $r = \sqrt{\frac{SS_i}{N_i} - \left(\frac{LS_i}{N_i}\right)^2}$

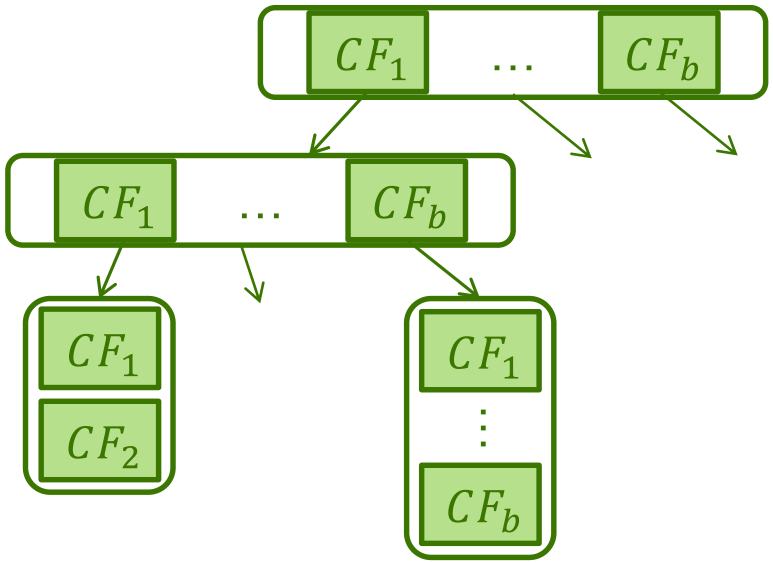
Clustering from Data Streams

BIRCH (Zhang et al., 1996)

- Usage of Microclusters within CF-Tree
  - B^+-Tree like structure
  - Two user specified parameters:
    - Branching factor B
    - Maximum diameter (or radius) T of a CF
  - Each non-leaf node contains at most B entries of the form $[CF_i, child_i]$ where
    - CF_i is the CF representing the subcluster that child forms
    - $child_i$ is a pointer to the i-th child node
  - Each leaf node contains entries of the form $[CF_i, prev, next]$

Clustering from Data Streams

BIRCH (Zhang et al., 1996)



- Inserts into CF-Tree
  - At each non-leaf node, the new object follows the *clo-sest-CF* path
  - At leaf node level, the *clo-sest-CF* tries to *absorb* the object (which depends on diameter threshold T and the page size)
    - If possible: update *closest-CF*
    - If not possible: make a new CF entry in the leaf node (split the parent node if there is no space)

Clustering from Data Streams

BIRCH (Zhang et al., 1996)

- Two step algorithm:

1. Online component:
   - Microclusters are kept locally
   - Maintenance of the hierarchical structure
   - Optional: Condense by building smaller CF-Tree (requires scan over leaf entries)

2. Offline component:
   - Apply global clustering to all leaf entries
   - Optional: Cluster refinement to the cost of additional passes (use centroids retrieved by global clustering and re-assign data points)

Clustering from Data Streams

CluStream (Aggarwal et al., 2003)

- Extension to BIRCH by incorporating temporal information
  → Consideration of cluster evolution over time

- Cluster Features:
  $CFT = (CF_2^x, CF_1^x, CF_2^t, CF_1^t, n)$
  $CF_2^x = \sum_{i=1}^{n} \overrightarrow{x_i}^2$ squared sum of data points
  $CF_1^x = \sum_{i=1}^{n} \overrightarrow{x_i}$ linear sum of data points
  $CF_2^t = \sum_{i=1}^{n} {t_i}^2$ squared sum of timestamps
  $CF_1^t = \sum_{i=1}^{n} t_i$ linear sum of timestamps

Clustering from Data Streams

CluStream (Aggarwal et al., 2003)

- Initialize: apply q-means over *initPoints*, built a summary for each cluster ($k \ll q \ll initPoints$)
- Online: microcluster maintenance
  - Find closest cluster clu of new point p
    if (p is within *max-boundary* of clu) p is absorbed by clu
    else create new cluster with p
  - If the number of clusters exceeds q, delete the oldest microcluster or merge the two closest ones

Clustering from Data Streams

CluStream (Aggarwal et al., 2003)

- Periodic storage of microcluster snapshots to disk
- Offline: on demand macro-clustering
  - User defines time horizon h and number of clusters k
  - Determine set of microclusters M within current timestamp t_c and $t_c - h$ $\quad (M(t_c) - M(\lfloor t_c - h \rfloor)$ with $M(\lfloor t_c - h \rfloor)$ being the snapshot just before $t_c - h)$
  - Apply k-means on M

Frequent Itemset Mining

- Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of *items* (e.g. products)
- Any subset $I \subseteq A$ is called an *itemset*
- Let $T = (t_1, t_2, \ldots, t_m)$ be a set of *transactions* with t_i being a pair $\langle TID_i, I_i \rangle$ where $I_i \subseteq A$ is a set of items (e.g. the set of products bought by a customer within a certain period in time)
- The *support* σ_{min} of an itemset $I \subseteq A$ is the number/fraction of transactions $t_i \in T$ that contain I

Frequent Itemset Mining

Example:
Given the set of items $A = \{a, b, c, d, e\}$, the set of transactions T, and a relative support $\sigma_{min} = 0.3$, determine the set of frequent item sets that is $\{I \subseteq A | \sigma_T(I) \geq \sigma_{min}\}$.

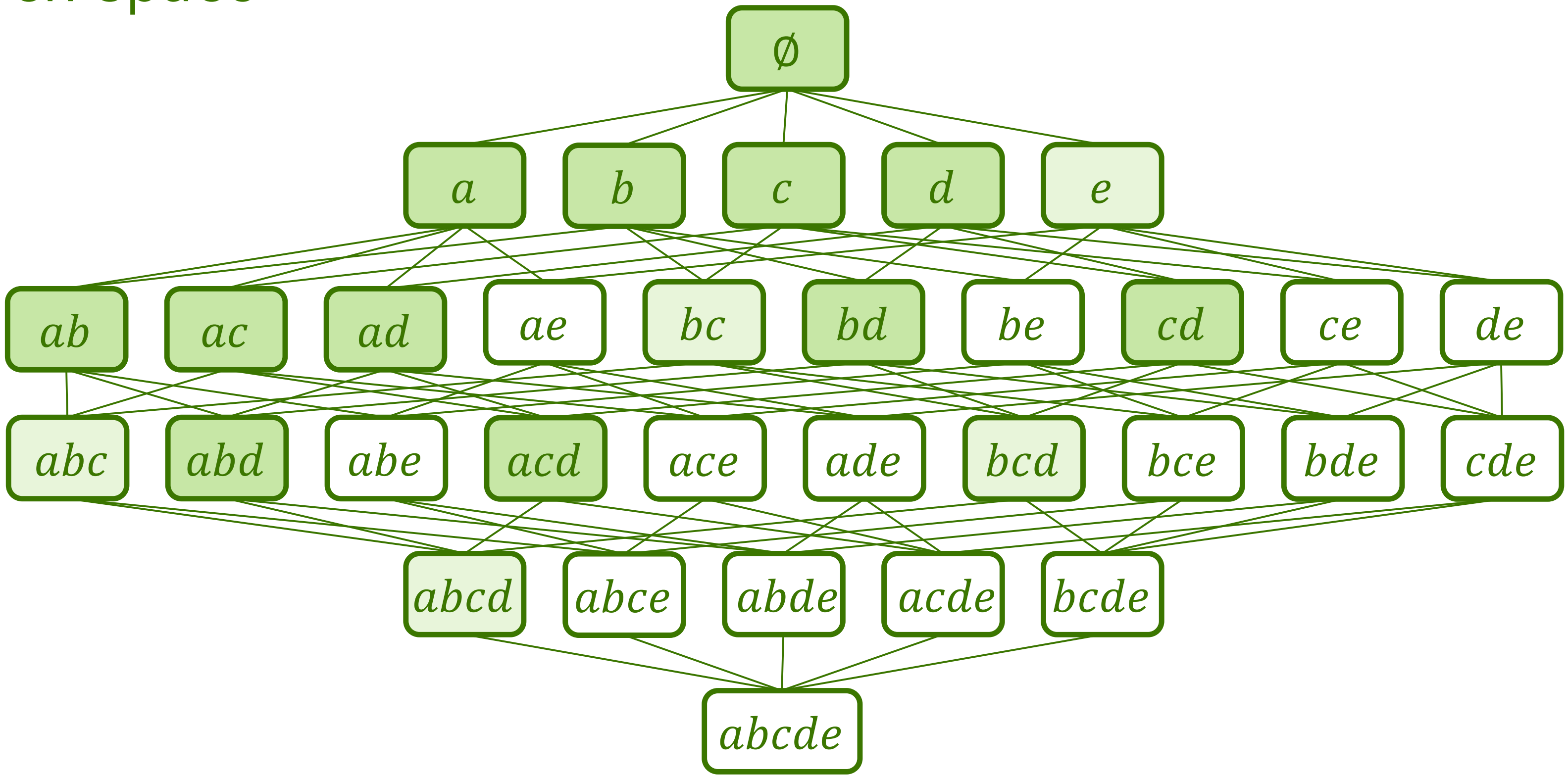
T:

TID_i	I_i
1	$\{a, b, c, d\}$
2	$\{b, d, e\}$
3	$\{a, b, d\}$
4	$\{a, b, c, d, e\}$
5	$\{a, c\}$
6	$\{c, d\}$
7	$\{a, c, d\}$

0 items	1 item	2 items	3 items
$\emptyset$: 7	$\{a\}$: 5	$\{a, b\}$: 3	$\{a, c, d\}$: 3
	$\{b\}$: 5	$\{a, c\}$: 4	$\{a, b, d\}$: 3
	$\{c\}$: 5	$\{a, d\}$: 4	
	$\{d\}$: 6	$\{b, d\}$: 4	
		$\{c, d\}$: 4	

Frequent Itemset Mining

Search space



Frequent Itemset Mining

LossyCounting Algorithm (Manku et al., 2002)

- One-pass algorithm for computing frequency counts that exceed a user-specified threshold

- Approximate error but guaranteed to be below a user-spedified boundary

→ Two parameters:
  - Support threshold $s \in [0,1]$
  - Error threshold $\epsilon \in [0,1]$
  - $\epsilon \ll s$

Frequent Itemset Mining

LossyCounting Algorithm (Manku et al., 2002)

- Setup:
  - Stream S is divided into buckets of width $\omega = \left\lceil \frac{1}{\epsilon} \right\rceil$
  - The current bucket id $b_{curr} = \left\lceil \frac{N}{\omega} \right\rceil$
  - For element e, the true frequency seen so far is f_e
  - The data structure D is a set of entries of the form (e, f, Δ)
    - e is the element
    - f is the frequency seen since e is in D
    - Δ is the maximum possible error, resp. the estimated frequency of e in buckets $b = 1$ to b_{curr}-1

Frequent Itemset Mining

LossyCounting Algorithm (Manku et al., 2002)

Algorithm LossyCounting
Input: data stream S, error threshold ϵ
begin
  $D = \emptyset$, $N = 0$, $\omega = \left\lceil \frac{1}{\epsilon} \right\rceil$
  while S **do**
    e_i ≔ next object from S
    $N \mathrel{+}= 1$
    $b_{curr} = \left\lceil \frac{N}{\omega} \right\rceil$
    if $e_i \in D$ **then**
      increment e_i's frequency by 1
    else
      $D.add((e_i, 1, b_{curr} - 1))$
    whenever $N \equiv 0 \bmod \omega$ **do**
      foreach entry (e, f, Δ) in D **do**
        if $f + \Delta \leq b_{curr}$ **then**
          delete (e, f, Δ)
end

Algorithm LossyCounting – User request
Input: lookup table D, support threshold s
begin
  $S = \emptyset$
  foreach entry (e, f, Δ) in D **do**
    if $f \geq (s - \epsilon)N$ **then**
      add (e, f, Δ) to S
  return S
end

f is the exact frequency count of e since the entry was inserted into D

Δ is the maximum number of times e could have occurred in the first $b_{curr} - 1$ buckets

Further Reading

- Joao Gama: *Knowledge Discovery from Data Streams* (http://www.liaad.up.pt/area/jgama/DataStreamsCRC.pdf)
- Gibbons, Phillip B., Yossi Matias, and Viswanath Poosala. *Fast incremental maintenance of approximate histograms*. VLDB. Vol. 97 (1997)
- Datar, Mayur, et al. *Maintaining stream statistics over sliding windows*. SIAM Journal on Computing 31.6 (2002)
- Klinkenberg, R., and Renz I. *Adaptive information filtering: Learning drifting concepts*. Proc. of AAAI-98/ICML-98 workshop Learning for Text Categorization (1998)
- Page, E. S. *Continuous Inspection Scheme*. Biometrika 41 (1954)
- Kifer, Daniel, Shai Ben-David, and Johannes Gehrke. *Detecting change in data streams*. VLDB. (2004)

Further Reading

- Spath, H. *Cluster Analysis Algorithms for Data Reduction and Classification*. Ellis Horwood (1980)
- L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, R. Motwani: *Streaming-Data Algorithms for High-Quality Clustering*. ICDE. (2002)
- Zhang, Tian, Raghu Ramakrishnan, and Miron Livny. *BIRCH: an efficient data clustering method for very large databases*. ACM SIGMOD (1996)
- Aggarwal, Charu C., et al. *A framework for clustering evolving data streams*. Proc. VLDB (2003)
- Manku, Gurmeet Singh, and Rajeev Motwani. *Approximate frequency counts over data streams*. Proc. VLDB. (2002)