Data Mining: Concepts and Techniques

— Chapter 8 —

8.1. Mining data streams

Jiawei Han and Micheline Kamber
Department of Computer Science
University of Illinois at Urbana-Champaign
www.cs.uiuc.edu/~hanj

Chapter 8. Mining Stream, Time-Series, and Sequence Data

- **Mining data streams**
- **Mining time-series data**
- **Mining sequence patterns in transactional databases**
- **Mining sequence patterns in biological data**

Mining Data Streams

- What is stream data? Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

Characteristics of Data Streams

- Data Streams
  - Data streams—continuous, ordered, changing, fast, huge amount
  - Traditional DBMS—data stored in finite, persistent data sets
- Characteristics
  - Huge volumes of continuous data, possibly infinite
  - Fast changing and requires fast, real-time response
  - Data stream captures nicely our data processing needs of today
  - Random access is expensive—single scan algorithm (*can only have one look*)
  - Store only the summary of the data seen thus far
  - Most stream data are at pretty low-level or multi-dimensional in nature, needs multi-level and multi-dimensional processing

Stream Data Applications

- Telecommunication calling records
- Business: credit card transaction flows
- Network monitoring and traffic engineering
- Financial market: stock exchange
- Engineering & industrial processes: power supply & manufacturing
- Sensor, monitoring & surveillance: video streams, RFIDs
- Security monitoring
- Web logs and Web page click streams
- Massive data sets (even saved but random access is too expensive): digital earth, Hadron collider, SETI, etc.

DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- "Unbounded" disk store
- Only current state matters
- No real-time services
- Relatively low update rate
- Data at any granularity
- Assume precise data
- Access plan determined by query processor, physical DB design

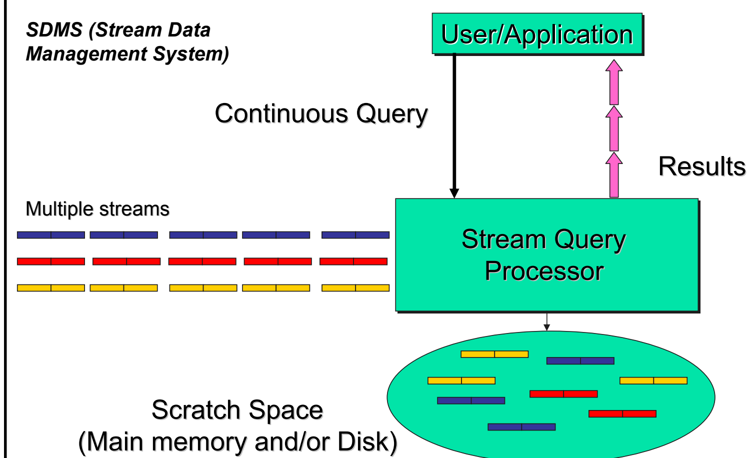
- Transient streams
- Continuous queries
- Sequential access
- Bounded main memory
- Historical data is important
- Real-time requirements
- Possibly multi-GB arrival rate
- Data at fine granularity
- Data stale/imprecise
- Unpredictable/variable data arrival and characteristics

Ack. From Motwani's PODS tutorial slides

Mining Data Streams

- What is stream data? Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

Architecture: Stream Query Processing



Challenges of Stream Data Processing

- Multiple, continuous, rapid, time-varying, ordered streams
- Main memory computations
- Queries are often continuous
  - Evaluated continuously as stream data arrives
  - Answer updated over time
- Queries are often complex
  - Beyond element-at-a-time processing
  - Beyond stream-at-a-time processing
  - Beyond relational queries (scientific, data mining, OLAP)
- Multi-level/multi-dimensional processing and data mining
  - Most stream data are at low-level or multi-dimensional in nature

Processing Stream Queries

- Query types
  - One-time query vs. continuous query (being evaluated continuously as stream continues to arrive)
  - Predefined query vs. ad-hoc query (issued on-line)
- Unbounded memory requirements
  - For real-time response, main memory algorithm should be used
  - Memory requirement is unbounded if one will join future tuples
- Approximate query answering
  - With bounded memory, it is not always possible to produce exact answers
  - High-quality approximate answers are desired
  - Data reduction and synopsis construction methods
    - Sketches, random sampling, histograms, wavelets, etc.

Methodologies for Stream Data Processing

- Major challenges
  - Keep track of a large universe, e.g., pairs of IP address, not ages
- Methodology
  - Synopses (trade-off between accuracy and storage)
  - Use *synopsis data structure*, much smaller ($O(\log^k N)$ space) than their base data set ($O(N)$ space)
  - Compute an *approximate answer* within a *small error range* (factor ε of the actual answer)
- Major methods
  - Random sampling
  - Histograms
  - Sliding windows
  - Multi-resolution model
  - Sketches
  - Radomized algorithms

Stream Data Processing Methods (1)

- **Random sampling** (but without knowing the total length in advance)
  - *Reservoir sampling*: maintain a set of *s* candidates in the reservoir, which form a true random sample of the elements seen so far in the stream. As the data stream flows, every new element has a certain probability (s/N) of replacing an old element in the reservoir.
- **Sliding windows**
  - Make decisions based only on *recent data* of sliding window size w
  - An element arriving at time t expires at time $t + w$
- **Histograms**
  - Approximate the frequency distribution of element values in a stream
  - Partition data into a set of contiguous buckets
  - Equal-width (equal value range for buckets) vs. V-optimal (minimizing frequency variance within each bucket)
- **Multi-resolution models**
  - Popular models: balanced binary trees, micro-clusters, and wavelets

Stream Data Processing Methods (2)

- Sketches
  - Histograms and wavelets require multi-passes over the data but sketches can operate in a single pass
  - Frequency moments F_k of a stream A = {a_1, …, a_N}: $F_k = \sum_{i=1}^{v} m_i^k$

    where v: the universe or domain size, m_i: the frequency of i in the sequence
    - Given N elements and v values, sketches can approximate F_0, F_1, F_2 in O(log v + log N) space
- Randomized algorithms
  - Monte Carlo algorithm: bound on running time but may not return correct result (Note: Las Vegas algorithm correct but unbounded.)
  - Chebyshev's inequality: $P(|X - \mu| > k) \le \frac{\sigma^2}{k^2}$
    - Where X a random variable with mean μ and standard deviation σ
  - Chernoff bound: $P[X < (1+\delta)\mu|] < e^{-\mu\delta^2/4}$
    - Where X the sum of independent Poisson trials X_1, …, X_n, δ in (0, 1]
    - The probability decreases exponentially as we move from the mean

Approximate Query Answering in Streams

- Sliding windows
  - Only over sliding windows of *recent stream data*
  - Approximation but often more desirable in applications
- Batched processing, sampling and synopses
  - Batched if update is fast but computing is slow
    - Compute periodically, not very timely
  - Sampling if update is slow but computing is fast
    - Compute using sample data, but not good for joins, etc.
  - Synopsis data structures
    - Maintain a small *synopsis* or *sketch* of data
    - Good for querying historical data
- Blocking operators, e.g., sorting, avg, min, etc.
  - Blocking if unable to produce the first output until seeing the entire input

Projects on DSMS (Data Stream Management System)

- Research projects and system prototypes
  - STREAM (Stanford): A general-purpose DSMS
  - Cougar (Cornell): sensors
  - Aurora (Brown/MIT): sensor monitoring, dataflow
  - Hancock (AT&T): telecom streams
  - Niagara (OGI/Wisconsin): Internet XML databases
  - OpenCQ (Georgia Tech): triggers, incr. view maintenance
  - Tapestry (Xerox): pub/sub content-based filtering
  - Telegraph (Berkeley): adaptive engine for sensors
  - Tradebot (www.tradebot.com): stock tickers & streams
  - Tribeca (Bellcore): network monitoring
  - MAIDS (UIUC/NCSA): Mining Alarming Incidents in Data Streams

Stream Data Mining vs. Stream Querying

- Stream mining — A more challenging task in many cases
  - It shares most of the difficulties with stream querying
    - But often requires less "precision", e.g., no join, grouping, sorting
  - Patterns are hidden and more general than querying
  - It may require exploratory analysis
    - Not necessarily continuous queries
- Stream data mining tasks
  - Multi-dimensional on-line analysis of streams
  - Mining outliers and unusual patterns in stream data
  - Clustering data streams
  - Classification of stream data

Mining Data Streams

- What is stream data? Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

Challenges for Mining Dynamics in Data Streams

- Most stream data are at pretty low-level or multi-dimensional in nature: needs ML/MD processing
- Analysis requirements
  - Multi-dimensional trends and unusual patterns
  - Capturing important changes at multi-dimensions/levels
  - Fast, real-time detection and response
  - Comparing with data cube: Similarity and differences
- Stream (data) cube or stream OLAP: Is this feasible?
  - Can we implement it efficiently?

Multi-Dimensional Stream Analysis: Examples

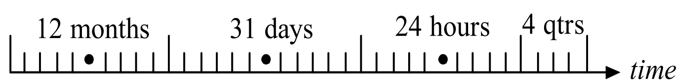
- Analysis of Web click streams
  - Raw data at low levels: seconds, web page addresses, user IP addresses, …
  - Analysts want: changes, trends, unusual patterns, at reasonable levels of details
  - E.g., *Average clicking traffic in North America on sports in the last 15 minutes is 40% higher than that in the last 24 hours."*
- Analysis of power consumption streams
  - Raw data: power consumption flow for every household, every minute
  - Patterns one may find: *average hourly power consumption surges up 30% for manufacturing companies in Chicago in the last 2 hours today than that of the same day a week ago*

A Stream Cube Architecture

- A tilted time frame
  - Different time granularities
    - second, minute, quarter, hour, day, week, …
- Critical layers
  - Minimum interest layer (m-layer)
  - Observation layer (o-layer)
  - User: watches at o-layer and occasionally needs to drill-down down to m-layer
- Partial materialization of stream cubes
  - Full materialization: too space and time consuming
  - No materialization: slow response at query time
  - Partial materialization: what do we mean "partial"?

A Tilted Time Model

- Natural tilted time frame:
  - Example: Minimal: quarter, then 4 quarters → 1 hour, 24 hours → day, …



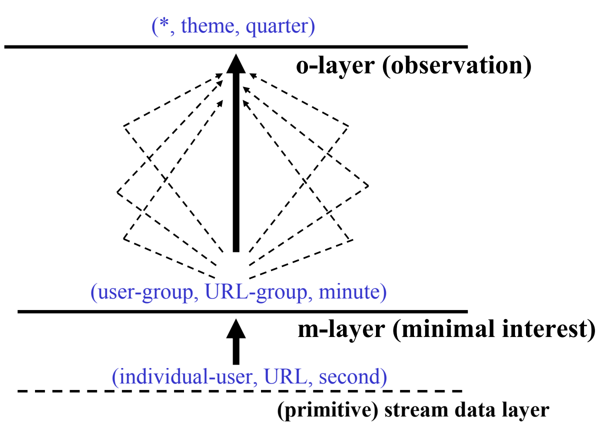
- Logarithmic tilted time frame:
  - Example: Minimal: 1 minute, then 1, 2, 4, 8, 16, 32, …

A Tilted Time Model (2)

- Pyramidal tilted time frame:
  - Example: Suppose there are 6 frames and each takes maximal 3 snapshots
  - Given a snapshot number N, if N mod $2^d = 0$, insert into the frame number d (=0, …, 5). If there are more than 3 snapshots, "kick out" the oldest one.

Frame no.	Snapshots (by clock time)
0	69 67 65
1	70 66 62
2	68 60 52
3	56 40 24
4	48 16
5	64 32

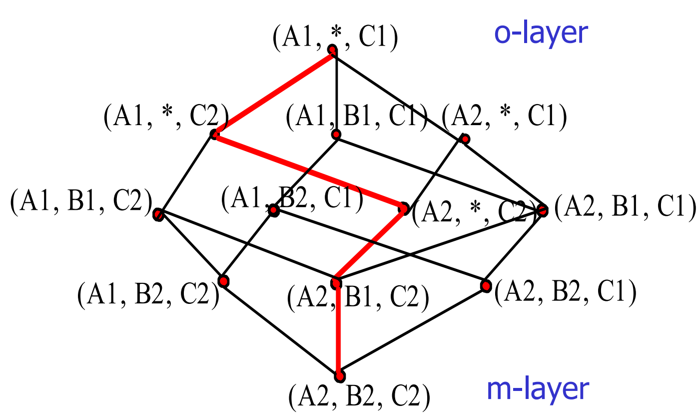
Two Critical Layers in the Stream Cube



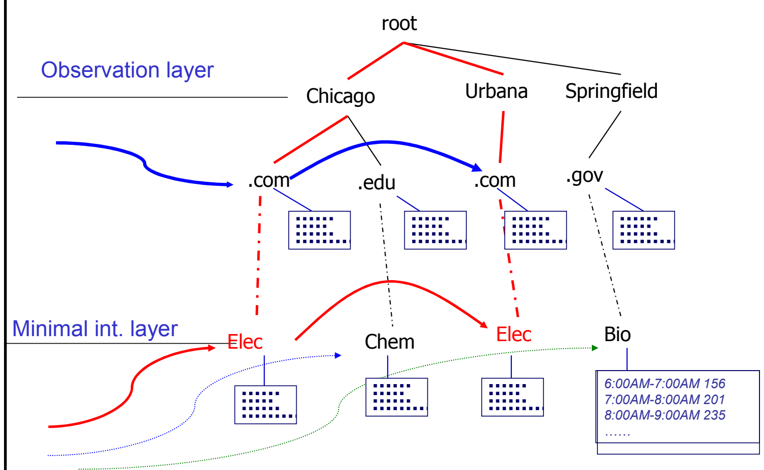
On-Line Partial Materialization vs. OLAP Processing

- On-line materialization
  - Materialization takes precious space and time
    - Only incremental materialization (with tilted time frame)
  - Only materialize "cuboids" of the critical layers?
    - Online computation may take too much time
  - Preferred solution:
    - *popular-path* approach: Materializing cuboids along the popular drilling paths
    - *H-tree structure*: Such cuboids can be computed and stored efficiently using the H-tree structure
- Online aggregation vs. query-based computation
  - Online computing while streaming: aggregating stream cubes
  - Query-based computation: using computed cuboids

Stream Cube Structure: From m-layer to o-layer



An H-Tree Cubing Structure



Benefits of H-Tree and H-Cubing

- H-tree and H-cubing
  - Developed for computing data cubes and ice-berg cubes
    - J. Han, J. Pei, G. Dong, and K. Wang, "*Efficient Computation of Iceberg Cubes with Complex Measures*", SIGMOD'01
  - Fast cubing, space preserving in cube computation
- Using H-tree for stream cubing
  - Space preserving
    - Intermediate aggregates can be computed incrementally and saved in tree nodes
  - Facilitate computing other cells and multi-dimensional analysis
  - H-tree with computed cells can be viewed as *stream cube*

Mining Data Streams

- What is stream data? Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

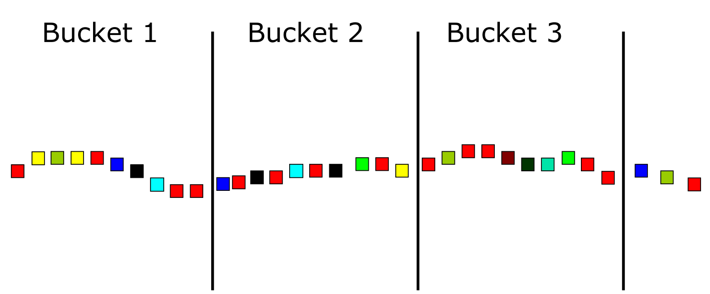
Frequent Patterns for Stream Data

- Frequent pattern mining is valuable in stream applications
  - e.g., network intrusion mining (Dokas, et al'02)
- Mining precise freq. patterns in stream data: unrealistic
  - Even when stored in a compressed form, such as FPtree
- How to mine frequent patterns with good approximation?
  - Approximate frequent patterns (Manku & Motwani VLDB'02)
  - Keep only current frequent patterns? No changes can be detected
- Mining evolution freq. patterns (C. Giannella, J. Han, X. Yan, P.S. Yu, 2003)
  - Use tilted time window frame
  - Mining evolution and dramatic changes of frequent patterns
- Space-saving computation of frequent and top-k elements (Metwally, Agrawal, and El Abbadi, ICDT'05)

Mining Approximate Frequent Patterns

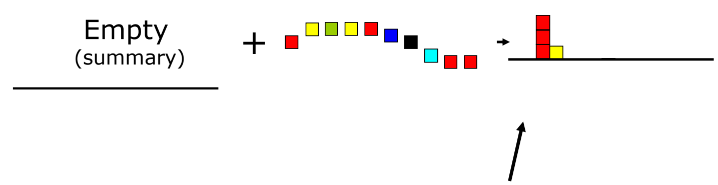
- Mining precise freq. patterns in stream data: unrealistic
  - Even when stored in a compressed form, such as FPtree
- Approximate answers are often sufficient (e.g., trend/pattern analysis)
  - **Example:** a router is interested in all flows:
    - whose frequency is at least 1% (σ) of the entire traffic stream seen so far
    - and feels that a 1/10 of σ (ε = 0.1%) error is comfortable
- How to mine frequent patterns with good approximation?
  - Lossy Counting Algorithm (Manku & Motwani, VLDB'02)
  - Major ideas: not tracing items until they become frequent
  - Advantage: guaranteed error bound
  - Disadvantage: keep a large set of traces

Lossy Counting for Frequent Items



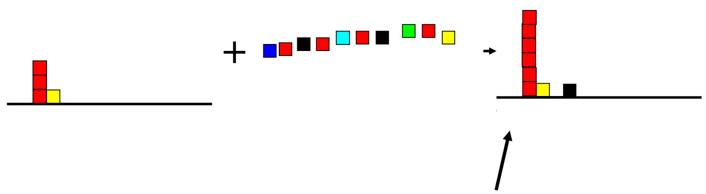
Divide Stream into 'Buckets' (bucket size is 1/ ε = 1000)

First Bucket of Stream



At bucket boundary, decrease all counters by 1 to get the new updated summary.

Next Bucket of Stream



At bucket boundary, decrease all counters by 1

Approximation Guarantee

- Given:
  - support threshold σ
  - error threshold ε
  - stream length N
- Output: items that remain with frequency counts exceeding $(\sigma - \varepsilon)$ N
- How much do we undercount?

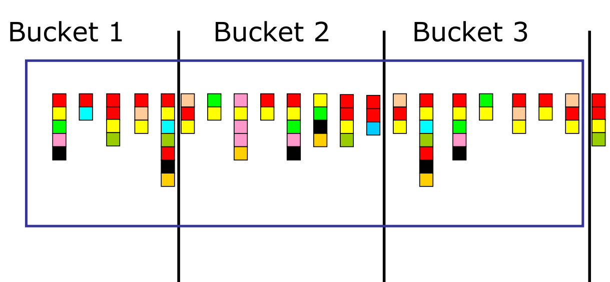
  If stream length seen so far = N

  and bucket-size = $1/\varepsilon$

  then frequency count error $\leq$ #buckets passed thus far = εN
- Approximation guarantee
  - No false negatives (discarded items have frequency < $(\sigma-\varepsilon)$N)
  - False positives have true frequency count at least $(\sigma-\varepsilon)$N
  - Frequency count underestimated by at most εN

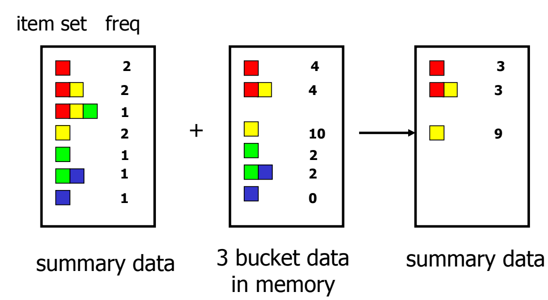
Lossy Counting For Frequent Itemsets

Divide Stream into 'Buckets' as for frequent items
But fill as many buckets as possible in main memory one time



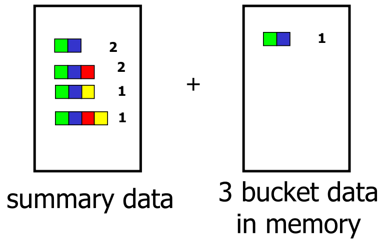
If we put 3 buckets of data into main memory one time,
Then decrease each frequency count by 3

Update of Summary Data Structure



Itemset ( ) is deleted.
That's why we choose a large number of buckets
=> delete more

Pruning Itemsets – Apriori Rule



If we find itemset ( ) is not a frequent itemset, then we do not need to consider its superset.

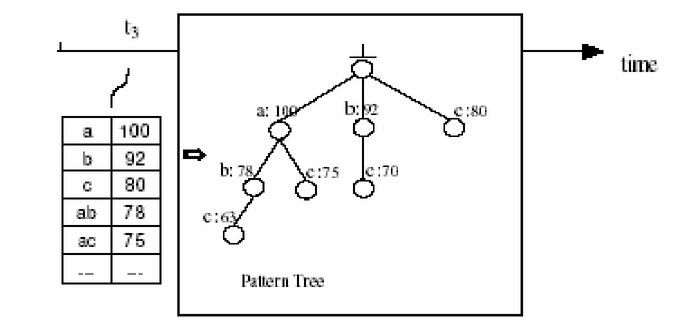
Summary of Lossy Counting

- Strength
  - A simple idea
  - Can be extended to frequent itemsets
- Weakness:
  - Space Bound is not good
  - For frequent itemsets, they do scan each record many times
  - The output is based on all previous data. But sometimes, we are only interested in recent data
- A space-saving method for stream frequent item mining
  - Metwally, Agrawal and El Abbadi, ICDT'05

Mining Evolution of Frequent Patterns for Stream Data

- Approximate frequent patterns (Manku & Motwani VLDB'02)
  - Keep only current frequent patterns—No changes can be detected
- Mining evolution and dramatic changes of frequent patterns (Giannella, Han, Yan, Yu, 2003)
  - Use tilted time window frame
  - Use compressed form to store significant (approximate) frequent patterns and their time-dependent traces
- Note: To mine precise counts, one has to trace/keep a fixed (and small) set of items

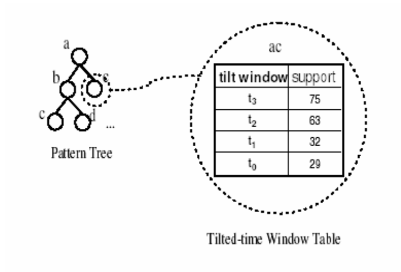
Two Structures for Mining Frequent Patterns with Tilted-Time Window



- FP-Trees store Frequent Patterns, rather than Transactions
- Tilted-time major: An FP-tree for each tilted time frame

Frequent Pattern & Tilted-Time Window (2)

- The second data structure:
  - Observation: FP-Trees of different time units are similar
  - Pattern-tree major:
    - each node is associated with a tilted-time window
    - on insert all the tilt windows can be updated



Mining Data Streams

- What is stream data? Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

Classification for Dynamic Data Streams

- Decision tree induction for stream data classification
  - VFDT (Very Fast Decision Tree)/CVFDT (Domingos, Hulten, Spencer, KDD00/KDD01)
- Is decision-tree good for modeling fast changing data, e.g., stock market analysis?
- Other stream classification methods
  - Instead of decision-trees, consider other models
    - Naïve Bayesian
    - Ensemble (Wang, Fan, Yu, Han. KDD'03)
    - K-nearest neighbors (Aggarwal, Han, Wang, Yu. KDD'04)
  - Tilted time framework, incremental updating, dynamic maintenance, and model construction
  - Comparing of models to find changes

Hoeffding Tree

- With high probability, classifies tuples the same
- Only uses small sample
  - Based on Hoeffding Bound principle
- Hoeffding Bound (Additive Chernoff Bound)

  r a random variable

  R the range of r

  n the # independent observations

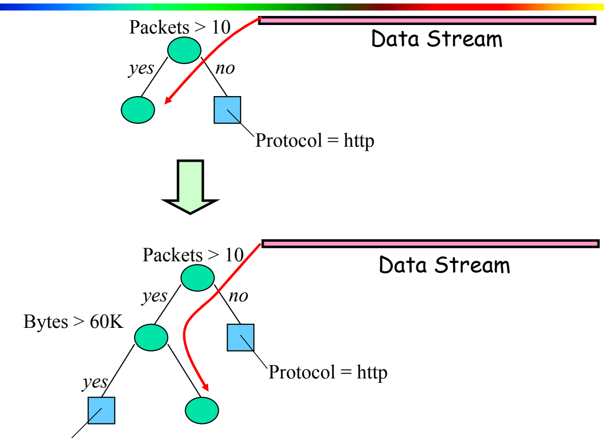
  Mean of r is at least $r_{avg} - \varepsilon$, with probability $1 - \delta$

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

Hoeffding Tree Algorithm

- Hoeffding Tree Input
  S the sequence of examples
  X the attributes
  G( ) the evaluation function (for determining similarity)
  D the desired accuracy
- Hoeffding Tree Algorithm
  for each example in S
  retrieve $G(X_a)$ and $G(X_b)$ //two highest $G(X_i)$
  if ($G(X_a) - G(X_b) > \varepsilon$)
  split on X_a
  recurse to next node
  break

Decision-Tree Induction with Data Streams



Ack. From Gehrke's SIGMOD tutorial slides

Hoeffding Tree: Strengths and Weaknesses

- Strengths
  - Scales better than traditional methods
    - Sublinear with sampling
    - Very small memory utilization
  - Incremental
    - Make class predictions in parallel
    - New examples are added as they come
- Weakness
  - Could spend a lot of time with ties
  - Memory used with tree expansion
  - Number of candidate attributes

VFDT (Very Fast Decision Tree)

- Modifications to Hoeffding Tree
  - Near-ties broken more aggressively
  - G computed every n_{min}
  - Deactivates certain leaves to save memory
  - Poor attributes dropped
  - Initialize with traditional learner (helps learning curve)
- Compare to Hoeffding Tree: Better time and memory
- Compare to traditional decision tree
  - Similar accuracy
  - Better runtime with 1.61 million examples
    - 21 minutes for VFDT
    - 24 hours for C4.5
- Still does not handle concept drift

CVFDT (Concept-adapting VFDT)

- Concept Drift
  - Time-changing data streams
  - Incorporate new and eliminate old
- CVFDT
  - Increments count with new example
  - Decrement old example
    - Sliding window
    - Nodes assigned monotonically increasing IDs
  - Grows alternate subtrees
  - When alternate more accurate => replace old
  - O(w) better runtime than VFDT-window

Ensemble of Classifiers Algorithm

- H. Wang, W. Fan, P. S. Yu, and J. Han, "Mining Concept-Drifting Data Streams using Ensemble Classifiers", KDD'03.
- Method (derived from the ensemble idea in classification)
  - train K classifiers from K chunks
  - for each subsequent chunk
    - train a new classifier
    - test other classifiers against the chunk
    - assign weight to each classifier
    - select top K classifiers

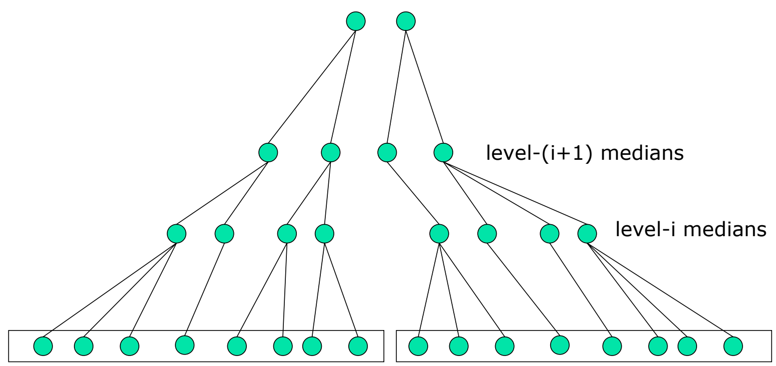
Mining Data Streams

- What is stream data? Why Stream Data Systems?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

Clustering Data Streams [GMMO01]

- Based on the k-median method
  - Data stream points from metric space
  - Find k clusters in the stream s.t. the sum of distances from data points to their closest center is minimized
- Constant factor approximation algorithm
  - In small space, a simple two step algorithm:
    1. For each set of M records, S_i, find O(k) centers in $S_1, \ldots, S_l$
       - Local clustering: Assign each point in S_i to its closest center
    2. Let S' be centers for $S_1, \ldots, S_l$ with each center weighted by number of points assigned to it
       - Cluster S' to find k centers

Hierarchical Clustering Tree



Hierarchical Tree and Drawbacks

- Method:
  - maintain at most m level-i medians
  - On seeing m of them, generate O(k) level-(i+1) medians of weight equal to the sum of the weights of the intermediate medians assigned to them
- Drawbacks:
  - Low quality for evolving data streams (register only k centers)
  - Limited functionality in discovering and exploring clusters over different portions of the stream over time

Clustering for Mining Stream Dynamics

- Network intrusion detection: one example
  - Detect bursts of activities or abrupt changes in real time—by on-line clustering
- Our methodology (C. Agarwal, J. Han, J. Wang, P.S. Yu, VLDB'03)
  - Tilted time frame work: o.w. dynamic changes cannot be found
  - Micro-clustering: better quality than k-means/k-median
    - incremental, online processing and maintenance)
  - Two stages: micro-clustering and macro-clustering
  - With limited "overhead" to achieve high efficiency, scalability, quality of results and power of evolution/change detection

CluStream: A Framework for Clustering Evolving Data Streams

- Design goal
  - High quality for clustering evolving data streams with greater functionality
  - While keep the stream mining requirement in mind
    - One-pass over the original stream data
    - Limited space usage and high efficiency
- CluStream: A framework for clustering evolving data streams
  - Divide the clustering process into online and offline components
    - Online component: periodically stores summary statistics about the stream data
    - Offline component: answers various user questions based on the stored summary statistics

The CluStream Framework

- Micro-cluster
  - Statistical information about data locality
  - Temporal extension of the *cluster-feature vector*
    - Multi-dimensional points $\overline{X}_1 \ldots \overline{X}_k \ldots$ with time stamps $T_1 \ldots T_k \ldots$
    - Each point contains *d* dimensions, i.e., $\overline{X}_i = \left(x_i^1 \ldots x_i^d\right)$
    - A micro-cluster for *n* points is defined as a (2.*d* + 3) tuple

$$\left(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n\right)$$

- Pyramidal time frame
  - Decide at what moments the snapshots of the statistical information are stored away on disk

CluStream: Pyramidal Time Frame

- Pyramidal time frame
  - Snapshots of a set of micro-clusters are stored following the pyramidal pattern
    - They are stored at differing levels of granularity depending on the recency
  - Snapshots are classified into different orders varying from 1 to *log(T)*
    - The *i*-th order snapshots occur at intervals of α^i where $\alpha \geq 1$
    - Only the last $(\alpha + 1)$ snapshots are stored

CluStream: Clustering On-line Streams

- Online micro-cluster maintenance
  - Initial creation of q micro-clusters
    - q is usually significantly larger than the number of natural clusters
  - Online incremental update of micro-clusters
    - If new point is within max-boundary, insert into the micro-cluster
    - Otherwise, create a new cluster
    - May delete obsolete micro-cluster or merge two closest ones
- Query-based macro-clustering
  - Based on a user-specified time-horizon h and the number of macro-clusters K, compute macroclusters using the k-means algorithm

Mining Data Streams

- What is stream data? Why SDS?
- Stream data management systems: Issues and solutions
- Stream data cube and multidimensional OLAP analysis
- Stream frequent pattern analysis
- Stream classification
- Stream cluster analysis
- Research issues

Stream Data Mining: Research Issues

- Mining sequential patterns in data streams
- Mining partial periodicity in data streams
- Mining notable gradients in data streams
- Mining outliers and unusual patterns in data streams
- Stream clustering
  - Multi-dimensional clustering analysis?
    - Cluster not confined to 2-D metric space, how to incorporate other features, especially non-numerical properties
  - Stream clustering with other clustering approaches?
  - Constraint-based cluster analysis with data streams?

Summary: Stream Data Mining

- Stream data mining: A rich and on-going research field
  - Current research focus in database community:
    - DSMS system architecture, continuous query processing, supporting mechanisms
  - Stream data mining and stream OLAP analysis
    - Powerful tools for finding general and unusual patterns
    - Effectiveness, efficiency and scalability: lots of open problems
- Our philosophy on stream data analysis and mining
  - A multi-dimensional stream analysis framework
  - Time is a special dimension: Tilted time frame
  - What to compute and what to save?—Critical layers
  - partial materialization and precomputation
  - Mining dynamics of stream data

References on Stream Data Mining (1)

- C. Aggarwal, J. Han, J. Wang, P. S. Yu. A Framework for Clustering Data Streams, VLDB'03
- C. C. Aggarwal, J. Han, J. Wang and P. S. Yu. On-Demand Classification of Evolving Data Streams, KDD'04
- C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Projected Clustering of High Dimensional Data Streams, VLDB'04
- S. Babu and J. Widom. Continuous Queries over Data Streams. SIGMOD Record, Sept. 2001
- B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom. Models and Issues in Data Stream Systems", PODS'02. (Conference tutorial)
- Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. "Multi-Dimensional Regression Analysis of Time-Series Data Streams, VLDB'02
- P. Domingos and G. Hulten, "Mining high-speed data streams", KDD'00
- A. Dobra, M. N. Garofalakis, J. Gehrke, R. Rastogi. Processing Complex Aggregate Queries over Data Streams, SIGMOD'02
- J. Gehrke, F. Korn, D. Srivastava. On computing correlated aggregates over continuous data streams. SIGMOD'01
- C. Giannella, J. Han, J. Pei, X. Yan and P.S. Yu. Mining frequent patterns in data streams at multiple time granularities, Kargupta, et al. (eds.), Next Generation Data Mining'04

References on Stream Data Mining (2)

- S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams, FOCS'00
- G. Hulten, L. Spencer and P. Domingos: Mining time-changing data streams. KDD 2001
- S. Madden, M. Shah, J. Hellerstein, V. Raman, Continuously Adaptive Continuous Queries over Streams, SIGMOD02
- G. Manku, R. Motwani. Approximate Frequency Counts over Data Streams, VLDB'02
- A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. ICDT'05
- S. Muthukrishnan, Data streams: algorithms and applications, Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, 2003
- R. Motwani and P. Raghavan, ***Randomized Algorithms***, Cambridge Univ. Press, 1995
- S. Viglas and J. Naughton, Rate-Based Query Optimization for Streaming Information Sources, SIGMOD'02
- Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, VLDB'02
- H. Wang, W. Fan, P. S. Yu, and J. Han, Mining Concept-Drifting Data Streams using Ensemble Classifiers, KDD'03

Data Mining:
Concepts and Techniques

— Chapter 8 —

8.2 Mining time-series data

Jiawei Han and Micheline Kamber
Department of Computer Science
University of Illinois at Urbana-Champaign
www.cs.uiuc.edu/~hanj

Chapter 8. Mining Stream, Time-Series, and Sequence Data

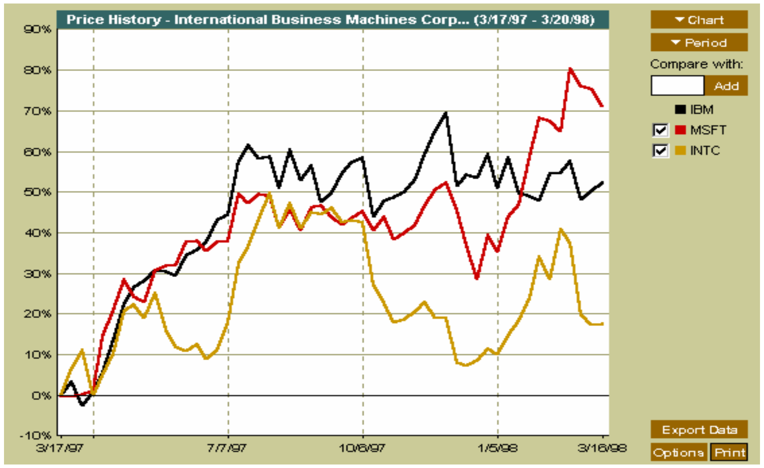
- Mining data streams
- **Mining time-series data**
- Mining sequence patterns in transactional databases
- Mining sequence patterns in biological data

Time-Series and Sequential Pattern Mining

- Regression and trend analysis—A statistical approach
- Similarity search in time-series analysis
- Sequential Pattern Mining
- Markov Chain
- Hidden Markov Model

Mining Time-Series Data

- Time-series database
  - Consists of sequences of values or events changing with time
  - Data is recorded at regular intervals
  - Characteristic time-series components
    - Trend, cycle, seasonal, irregular
- Applications
  - Financial: stock price, inflation
  - Industry: power consumption
  - Scientific: experiment results
  - Meteorological: precipitation



- A time series can be illustrated as a time-series graph which describes a point moving with the passage of time

Categories of Time-Series Movements

- Categories of Time-Series Movements
  - Long-term or trend movements (trend curve) (T): general direction in which a time series is moving over a long interval of time
  - Cyclic movements or cycle variations (C): long term oscillations about a trend line or curve
    - e.g., business cycles, may or may not be periodic
  - Seasonal movements or seasonal variations (S)
    - i.e, almost identical patterns that a time series appears to follow during corresponding months of successive years.
  - Irregular or random movements (I)
- Time series analysis: decomposition of a time series into these four basic movements
  - Additive Modal: TS = T + C + S + I
  - Multiplicative Modal: TS = T × C × S × I

Estimation of Trend Curve

- The freehand method
  - Fit the curve by looking at the graph
  - Costly and barely reliable for large-scaled data mining
- The least-square method
  - Find the curve minimizing the sum of the squares of the deviation of points on the curve from the corresponding data points
- The moving-average method

Moving Average

- Moving average of order n

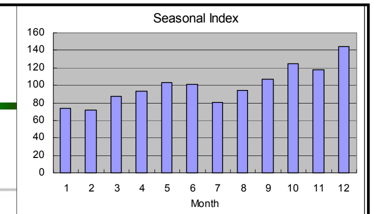
$$\frac{y_1 + y_2 + \cdots + y_n}{n}, \frac{y_2 + y_3 + \cdots + y_{n+1}}{n}, \frac{y_3 + y_4 + \cdots + y_{n+2}}{n}, \cdots$$

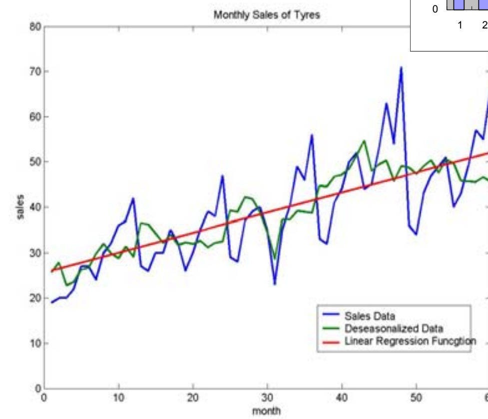
  - Smoothes the data
  - Eliminates cyclic, seasonal and irregular movements
  - Loses the data at the beginning or end of a series
  - Sensitive to outliers (can be reduced by weighted moving average)

Trend Discovery in Time-Series (1): Estimation of Seasonal Variations

- Seasonal index
  - Set of numbers showing the relative values of a variable during the months of the year
  - E.g., if the sales during October, November, and December are 80%, 120%, and 140% of the average monthly sales for the whole year, respectively, then 80, 120, and 140 are seasonal index numbers for these months
- Deseasonalized data
  - Data adjusted for seasonal variations for better trend and cyclic analysis
  - Divide the original monthly data by the seasonal index numbers for the corresponding months

Seasonal Index





Raw data from http://www.bbk.ac.uk/manop/man/docs/QII_2_2003%20Time%20series.pdf

Trend Discovery in Time-Series (2)

- Estimation of cyclic variations
  - If (approximate) periodicity of cycles occurs, cyclic index can be constructed in much the same manner as seasonal indexes
- Estimation of irregular variations
  - By adjusting the data for trend, seasonal and cyclic variations
- With the systematic analysis of the trend, cyclic, seasonal, and irregular components, it is possible to make long- or short-term predictions with reasonable quality

Time-Series & Sequential Pattern Mining

- Regression and trend analysis—A statistical approach
- Similarity search in time-series analysis
- Sequential Pattern Mining
- Markov Chain
- Hidden Markov Model

Similarity Search in Time-Series Analysis

- Normal database query finds exact match
- Similarity search finds data sequences that differ only slightly from the given query sequence
- Two categories of similarity queries
  - Whole matching: find a sequence that is similar to the query sequence
  - Subsequence matching: find all pairs of similar sequences
- Typical Applications
  - Financial market
  - Market basket data analysis
  - Scientific databases
  - Medical diagnosis

Data Transformation

- Many techniques for signal analysis require the data to be in the frequency domain
- Usually data-independent transformations are used
  - The transformation matrix is determined a priori
    - discrete Fourier transform (DFT)
    - discrete wavelet transform (DWT)
- The distance between two signals in the time domain is the same as their Euclidean distance in the frequency domain

Discrete Fourier Transform

from $\vec{x} = [x_t], t = 0, \ldots, n-1$ to $\vec{X} = [X_f], f = 0, \ldots, n-1$:

$$X_f = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} x_t \exp(-j2\pi f t/n),\ f = 0, 1, \ldots, n-1$$

- DFT does a good job of concentrating energy in the first few coefficients
- If we keep only the first few coefficients in the DFT, we can compute the lower bounds of the actual distance
- Feature extraction: keep the first few coefficients (F-index) as representative of the sequence

DFT (continued)

- Parseval's Theorem

$$\sum_{t=0}^{n-1} | x_t |^2 = \sum_{f=0}^{n-1} | X_f |^2$$

- The Euclidean distance between two signals in the time domain is the same as their distance in the frequency domain
- Keep the first few (say, 3) coefficients underestimates the distance and there will be no false dismissals!

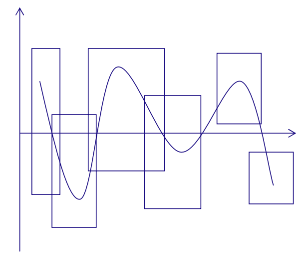
$$\sum_{t=0}^{n} | S[t] - Q[t] |^2 \leq \varepsilon \Rightarrow \sum_{f=0}^{3} | F(S)[f] - F(Q)[f] |^2 \leq \varepsilon$$

Multidimensional Indexing in Time-Series

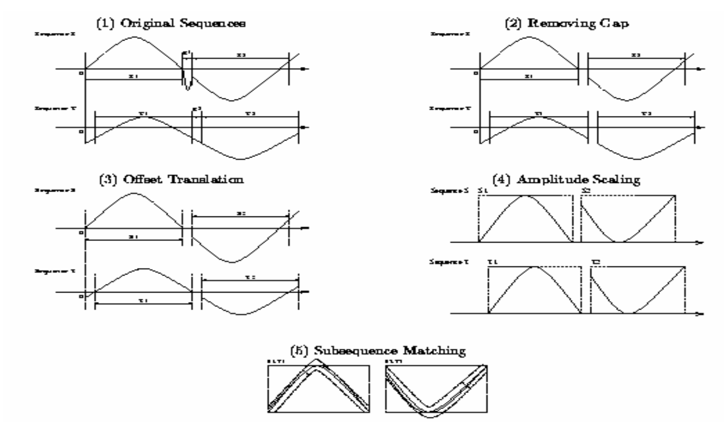
- Multidimensional index construction
  - Constructed for efficient accessing using the first few Fourier coefficients
- Similarity search
  - Use the index to retrieve the sequences that are at most a certain small distance away from the query sequence
  - Perform post-processing by computing the actual distance between sequences in the time domain and discard any false matches

Subsequence Matching

- Break each sequence into a set of pieces of window with length *w*
- Extract the features of the subsequence inside the window
- Map each sequence to a "trail" in the feature space
- Divide the trail of each sequence into "subtrails" and represent each of them with minimum bounding rectangle
- Use a multi-piece assembly algorithm to search for longer sequence matches



Analysis of Similar Time Series



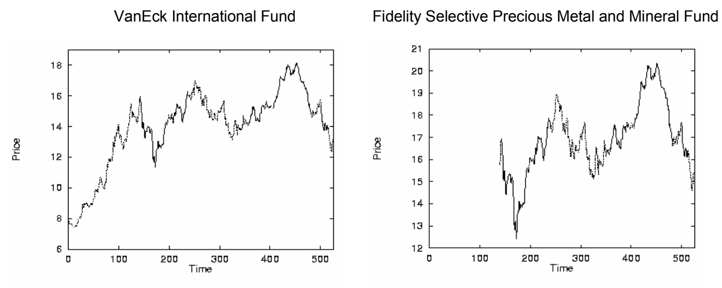
Enhanced Similarity Search Methods

- Allow for gaps within a sequence or differences in offsets or amplitudes
- Normalize sequences with amplitude scaling and offset translation
- Two subsequences are considered similar if one lies within an envelope of ε width around the other, ignoring outliers
- Two sequences are said to be similar if they have enough non-overlapping time-ordered pairs of similar subsequences
- Parameters specified by a user or expert: sliding window size, width of an envelope for similarity, maximum gap, and matching fraction

Steps for Performing a Similarity Search

- Atomic matching
  - Find all pairs of gap-free windows of a small length that are similar
- Window stitching
  - Stitch similar windows to form pairs of large similar subsequences allowing gaps between atomic matches
- Subsequence Ordering
  - Linearly order the subsequence matches to determine whether enough similar pieces exist

Similar Time Series Analysis



Two similar mutual funds in different fund groups

Query Languages for Time Sequences

- Time-sequence query language
  - Should be able to specify sophisticated queries like

  Find all of the sequences that are similar to some sequence in class *A*, but not similar to any sequence in class *B*
  - Should be able to support various kinds of queries: range queries, all-pair queries, and nearest neighbor queries
- Shape definition language
  - Allows users to define and query the overall shape of time sequences
  - Uses human readable series of sequence transitions or macros
  - Ignores the specific details
    - E.g., the pattern up, Up, UP can be used to describe increasing degrees of rising slopes (~ Paerson's code)
    - Macros: spike, valley, etc.

References on Time-Series & Similarity Search

- R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. FODO'93 (Foundations of Data Organization and Algorithms).
- R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. VLDB'95.
- R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. VLDB'95.
- C. Chatfield. The Analysis of Time Series: An Introduction, 3rd ed. Chapman & Hall, 1984.
- C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. SIGMOD'94.
- D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. SIGMOD'97.
- Y. Moon, K. Whang, W. Loh. Duality Based Subsequence Matching in Time-Series Databases, ICDE'02
- B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. ICDE'98.
- B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. ICDE'00.
- Dennis Shasha and Yunyue Zhu. **High Performance Discovery in Time Series: Techniques and Case Studies**, SPRINGER, 2004