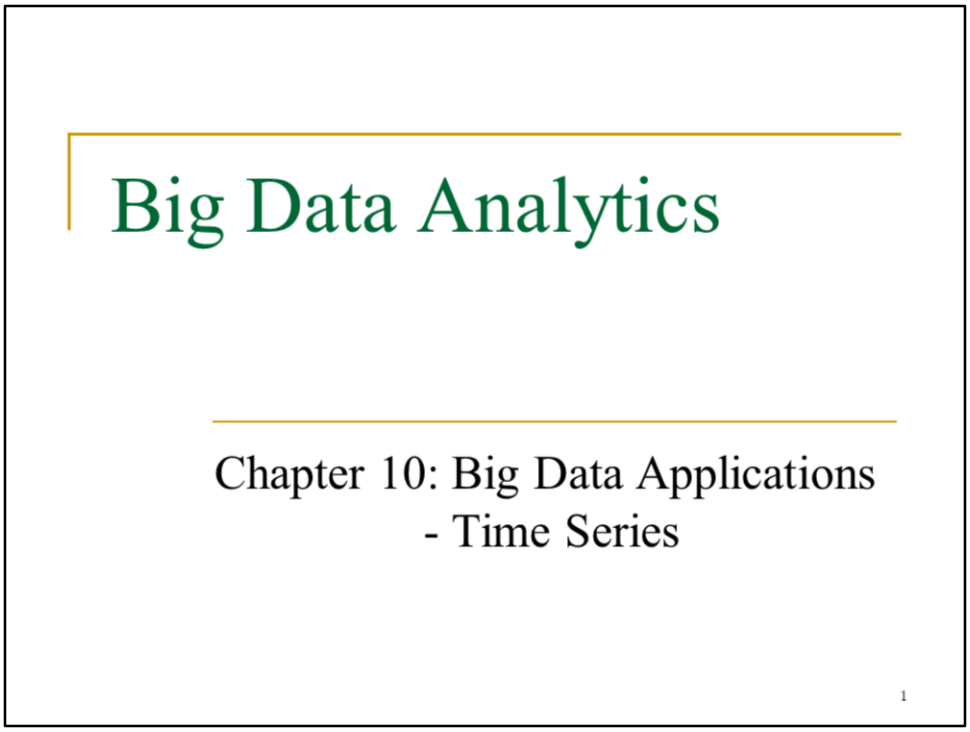
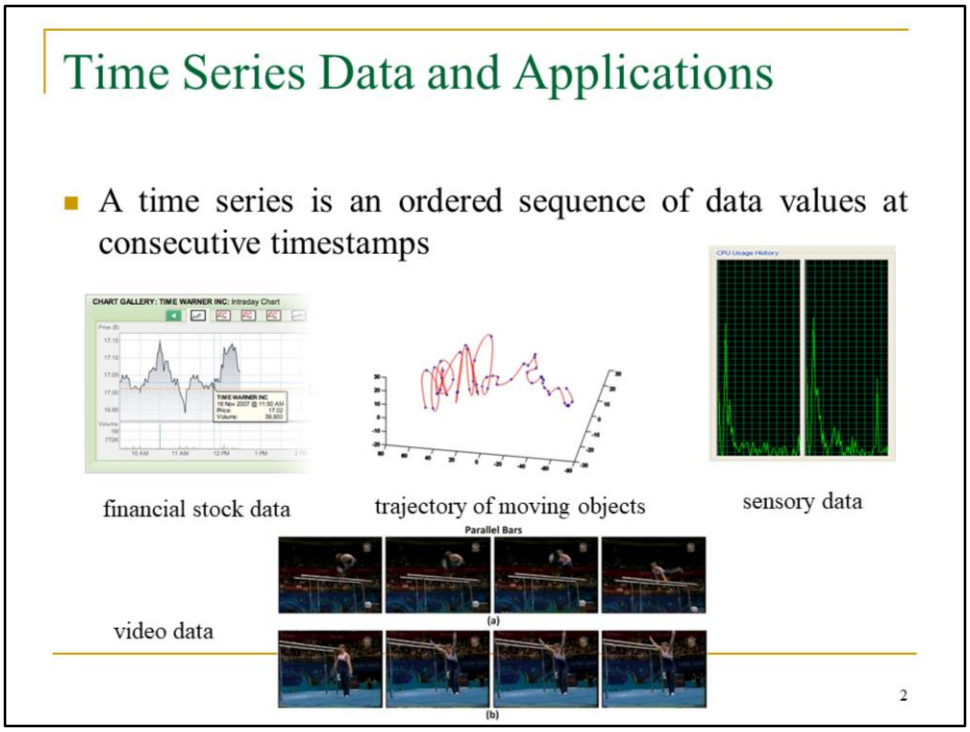


Chapter 10: Big Data Applications - Time Series

In this chapter, we will discuss one of big data applications with time series.



References

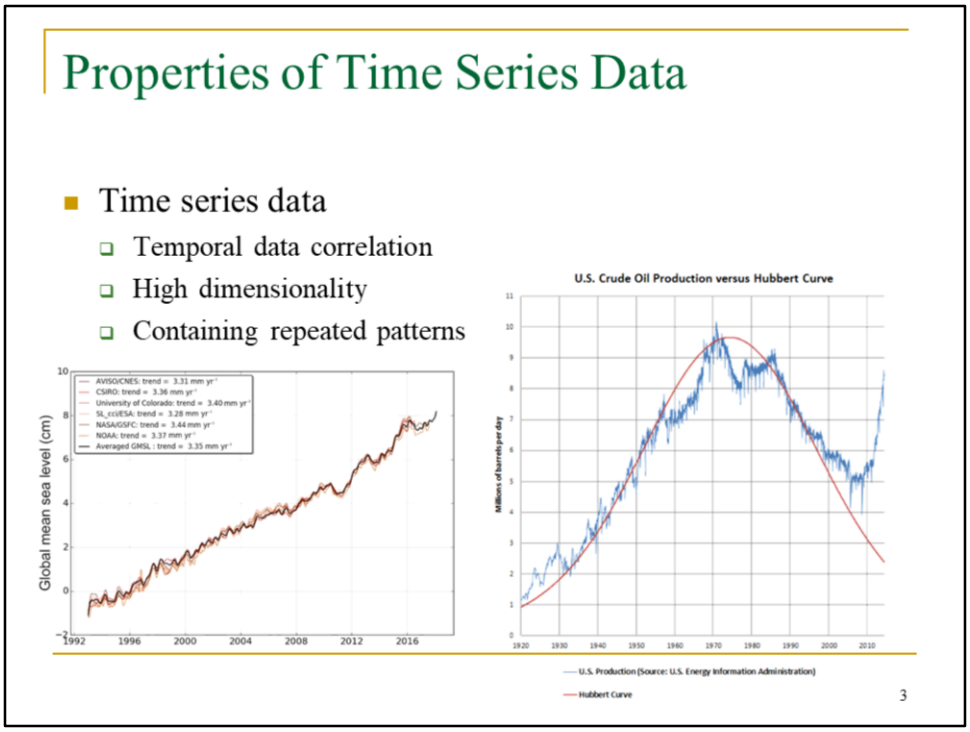
Matrix Investment Group, LLC - Making Real Estate Investing Accessible. http://www.matrixinvestmentgroup.com/. Accessed 18 Mar. 2019. Changes were not made.

Robert Pless: Image Spaces and Video Trajectories: Using Isomap to Explore Video Sequences. ICCV 2003: 1433-1440

Bikingdog. *English: Video Frames of the Parallel Bars Action Category in the UCF-101 Dataset[1] (a) The Highest Ranking Four Frames in Video Temporal Attention Weights, in Which the Athlete Is Performing on the Parallel Bars; (b) The Lowest Ranking Four Frames in Video Temporal Attention Weights, in Which the Athlete Is Standing on the Ground. All Weights Are Predicted by the ATW CNN Algorithm[2]. The Highly Weighted Video Frames Generally Captures the Most Distinctive Movements Relevant to the Action Category.* 12 Sept. 2018. Own work, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:Video_frames_of_the_Parallel_Bars_action_category.png. Changes were not made.

A time series is an ordered sequence of data values or objects at consecutive timestamps. Examples of time series include financial stock data over time, trajectories of moving objects, sensory data collected in a streaming fashion, and

video data containing streams of frames.

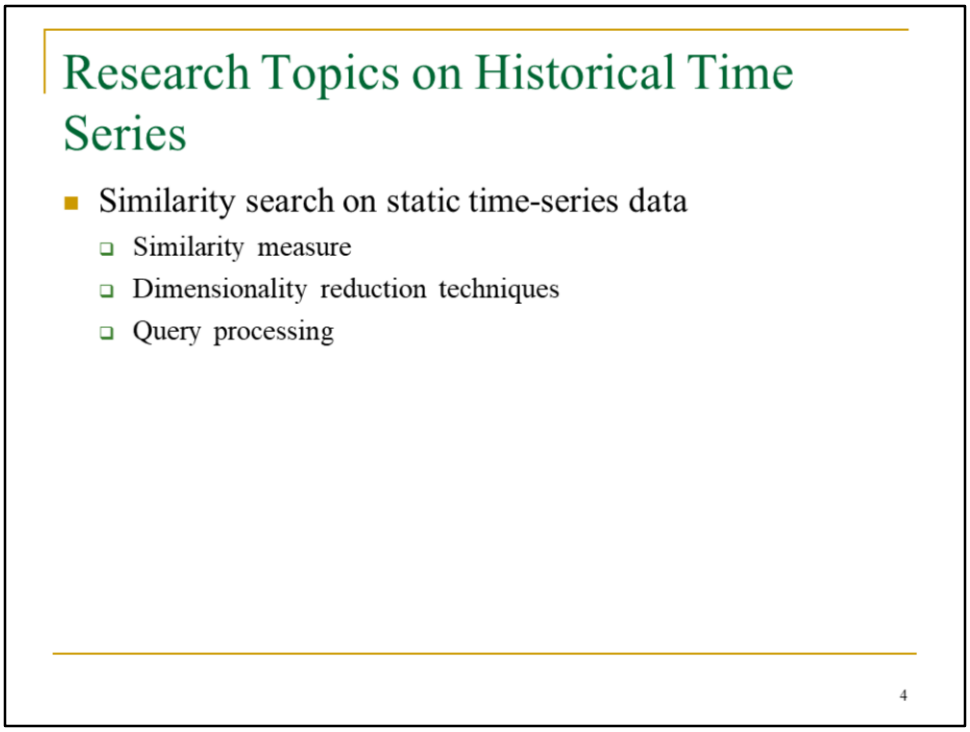


References

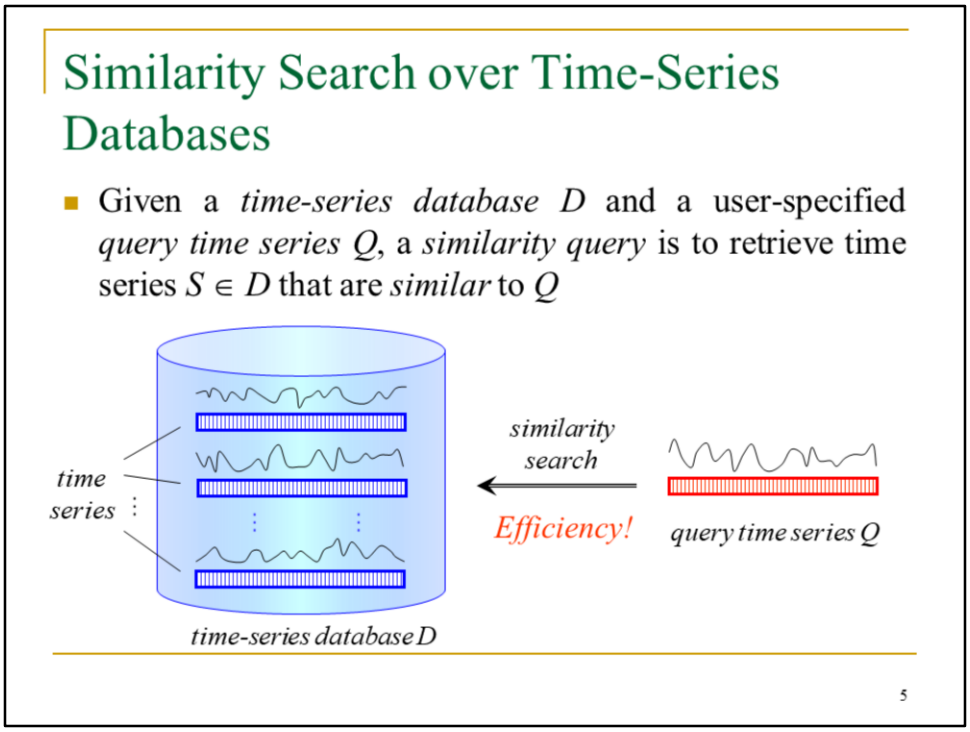
Group, WCRP Global Sea Level Budget. *English: Evolution of GMSL Time Series from Six Different Groups' (AVISO/CNES, SL_cci/ESA, University of Colorado, CSIRO, NASA/GSFC, NOAA) Products. Annual Signals Are Removed and 6-Month Smoothing Applied. All GMSL Time Series Are Centered in 1993 with Zero Mean.*2018. https://www.earth-syst-sci-data.net/10/1551/2018/essd-10-1551-2018.html, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:Evolution-of-GMSL-time-series-from-six-different-groups%E2%80%99.jpg. Changes were not made.

RockyMtnGuy. *English: US Crude Oil Production 1900 to 2010 Matched against a Typical Hubbert Curve. Data from the US Energy Information Administration.* 13 Feb. 2012. Own work, *Wikimedia Commons*, https://commons.wikimedia.org/wiki/File:US_Crude_Oil_Production_versus_Hubbert_Curve.png. Changes were not made.

Time series data often exhibit properties such as temporal correlations (i.e., data at two consecutive or close timestamps have similar values), high dimensionality (typically, 128, 256, or even 1024), and with repeated patterns (e.g., daily, weekly, or seasonally repeated patterns).



For the existing research on historical static time series data, there are some research topics, such as the selection/design of good similarity measure, dimensionality reduction for time series, and improving query efficiency over the time series data.



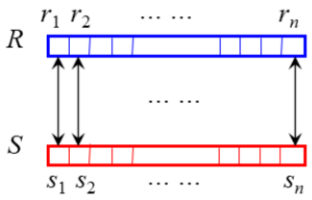
We first discuss the fundamental problem of similarity search in the static time-series database. Assume that we have a time-series database D containing many series. Given a query time series Q, we want to find those time series that are similar to the query series Q.

Since there are a lot of time series in the database D, it is very important to enhance the efficiency of the search.

Similarity Measures Without Warping

- Euclidean distance
  - Given $R = <r_1, r_2, \ldots, r_n>$ and $S = <s_1, s_2, \ldots, s_n>$

$$dist(R, S) = \sqrt{\sum_{i=1}^{n} (r_i - s_i)^2}$$



- L_p-norm distance

$$dist(R, S) = \sqrt[p]{\sum_{i=1}^{n} |r_i - s_i|^p} \quad (1 \leq p < \infty)$$

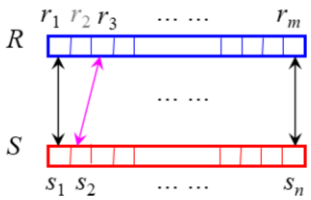
$$dist(R, S) = max_{i=1}^{n} |r_i - s_i| \quad (p = \infty)$$

6

In the literature, we have different measures to define the similarity between two time series. For similarity measure without warping, we can define the similarity functions such as Euclidean distance or L_p-norm distance between two time series of the same length, where p is within interval [1, \infinity]. When p = 2, L_p-norm is the Euclidean distance. Here, “without warping” means that the measure considers the distances between the same positions from two series, for example, the first positions r1 and s1 from time series R and S in the figure, respectively.

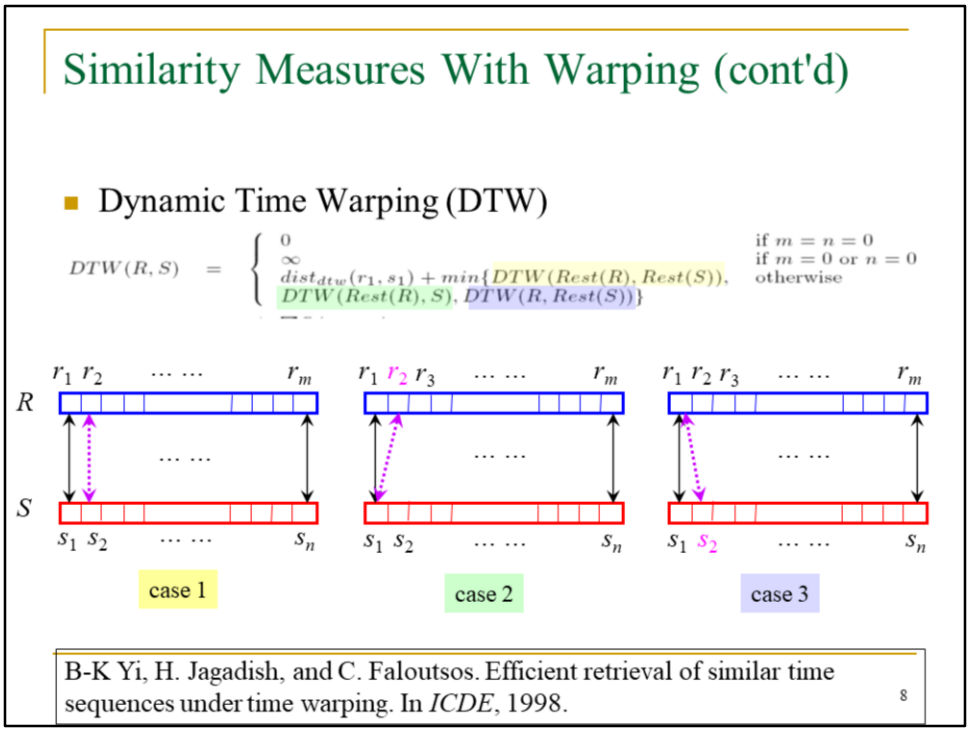
Similarity Measures With Warping

- The distance function allows the matching between two time series on warping positions



7

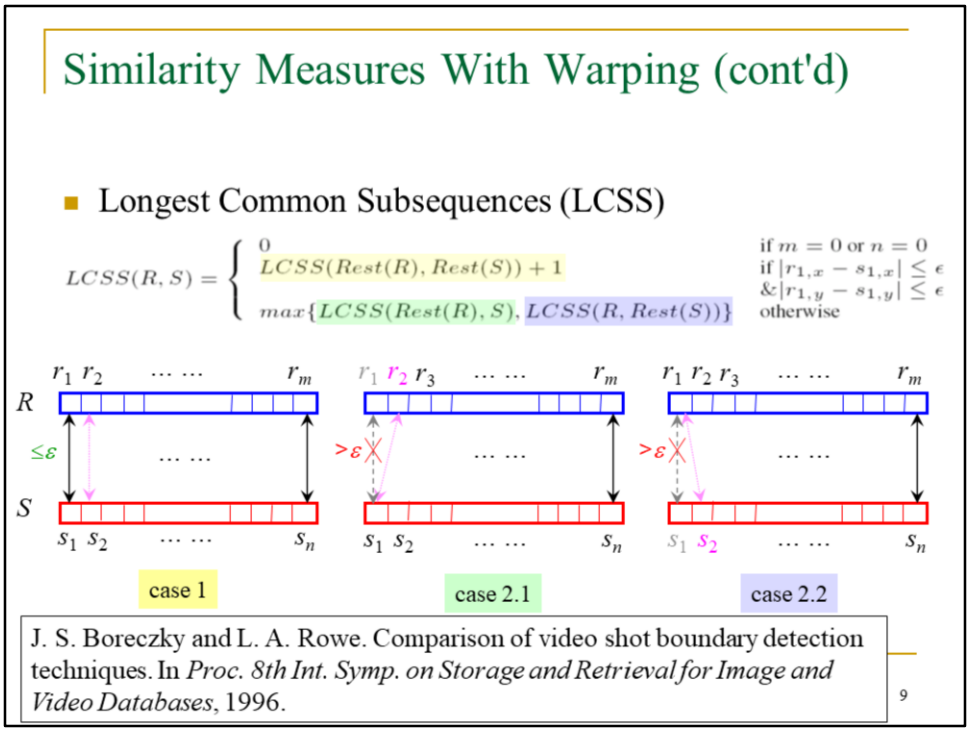
For similarity measure with warping, we can define the similarity function between two time series of the same or different lengths. In this case, the similarity measure can be defined with respect to warping positions from two time series. As an example in the figure, the distance between positions r3 and s2 (from series R and S, respectively) is considered (skipping position r2 in time series R).



The dynamic time warping (DTW) is one of the similarity measures between two time series with warping. The DTW is defined recursively. Specifically, for base cases, when the length of both series is 0, their similarity is equal to 0; when one of series has length 0, it will return positive infinity (since the first and last positions of both series are required to be aligned in DTW).

When the base cases do not hold, the DTW distance is given by the distance between the first positions s1 and r1 from two series R and S, respectively, plus the minimum possible distance between the remaining series. Here, due to the warping, s1 and r1 may still match with other positions in the remaining series.

Therefore, for the remaining series, there are 3 possible cases. The first case corresponds to the case where position r2 is aligned with s2; the second case is that r2 is aligned with s1, and; Case 3 is that position r1 is aligned with s2. Thus, DTW recursively considers the minimum possible distance for the remaining series in these 3 cases.



Another warping distance function is the longest common subsequences (LCSS), which is often used for the distance between 2D trajectories. Note that, for DTW, smaller DTW distance indicates high similarity; but for LCSS, large LCSS value indicates high similarity.

LCSS is also defined by a recursive function. For the base case, when one of series has length 0, we return 0.

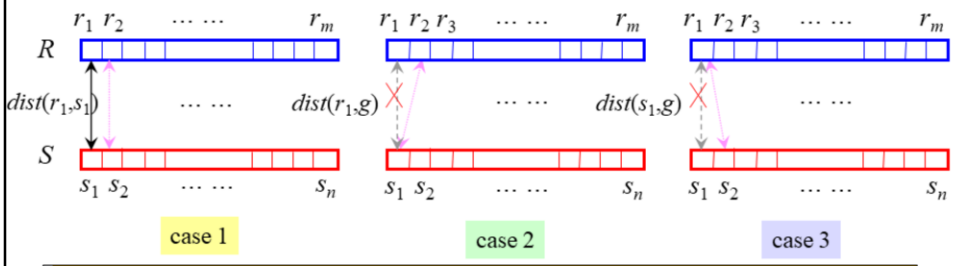
When the first positions (2D locations) r1 and s1 from two series are within \epsilon distance away on each dimension, we count them as matching and return 1 + LCSS of the remaining series Rest(R) and Rest(S) (excluding r1 and s1, respectively).

Otherwise (i.e., the distance between r1 and s1 is greater than \epsilon), we consider the matching between r2 and s1 (Case 2.1), or that between r1 and s2 (Case 2.2). So in the figure, we consider taking the maximum of the distances between the remaining series in Cases 2.1 and 2.2.

Similarity Measures With Warping (cont'd)

- Edit distance with Real Penalty (ERP)

$$ERP(R,S) = \begin{cases} \sum_1^n dist(s_i, g) & \text{if } m = 0 \\ \sum_1^m dist(r_i, g) & \text{if } n = 0 \\ min\{ERP(Rest(R), Rest(S)) + dist(t_1, s_1), & \\ ERP(Rest(R), S) + dist(r_1, g), & \text{otherwise} \\ ERP(R, Rest(S)) + dist(s_1, g)\} & \end{cases}$$



Lei Chen and Raymond Ng. On the marriage of Lp-norms and edit distance. In *VLDB*, 2004.

10

Similarly, there are some other warping distances such as edit distance with real penalty (ERP). ERP is a recursive function for the distance between two trajectories (one application of time series). In the base case, if one of series has length 0, we return the sum of distances between the other series (with nonzero length) and a constant g (which can be set in real applications).

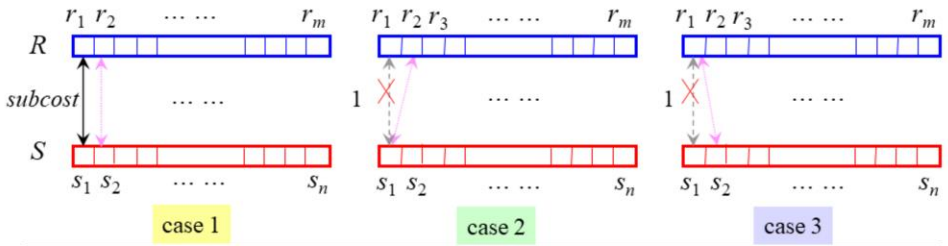
Otherwise, we also consider 3 warping cases as shown in the figure, and take the minimum distance among these 3 cases. In Case 1, we directly consider the distance between r1 and s1, plus the ERP distance for Rest(R) and Rest(S). For Case 2, s1 is not matching with r1, but matching with r2. So for r1, we consider the distance from r1 to constant g. The ERP is given by dist(r1, g) + ERP distance between Rest(R) and S. Similarly, for Case 3, we consider dist(s1, g) + ERP distance between R and Rest(S).

Similarity Measures With Warping (cont'd)

- Edit Distance on Real sequence (EDR)

$$EDR(R,S) = \begin{cases} n & \text{if } m = 0 \\ m & \text{if } n = 0 \\ min\{EDR(Rest(R), Rest(S)) + subcost, & \\ \quad EDR(Rest(R), S) + 1, EDR(R, Rest(S)) + 1\} & \text{otherwise} \end{cases}$$

where $subcost = 0$ if $match(r_1, s_1) = true$ and $subcost = 1$ otherwise.



Lei Chen, M. Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, 2005.

11

Another similarity measure is edit distance on real sequence (EDR), which is also defined by the recursive formula in the figure. You can check the details in the reference paper.

GEMINI Framework for Similarity Search on Time-Series Data

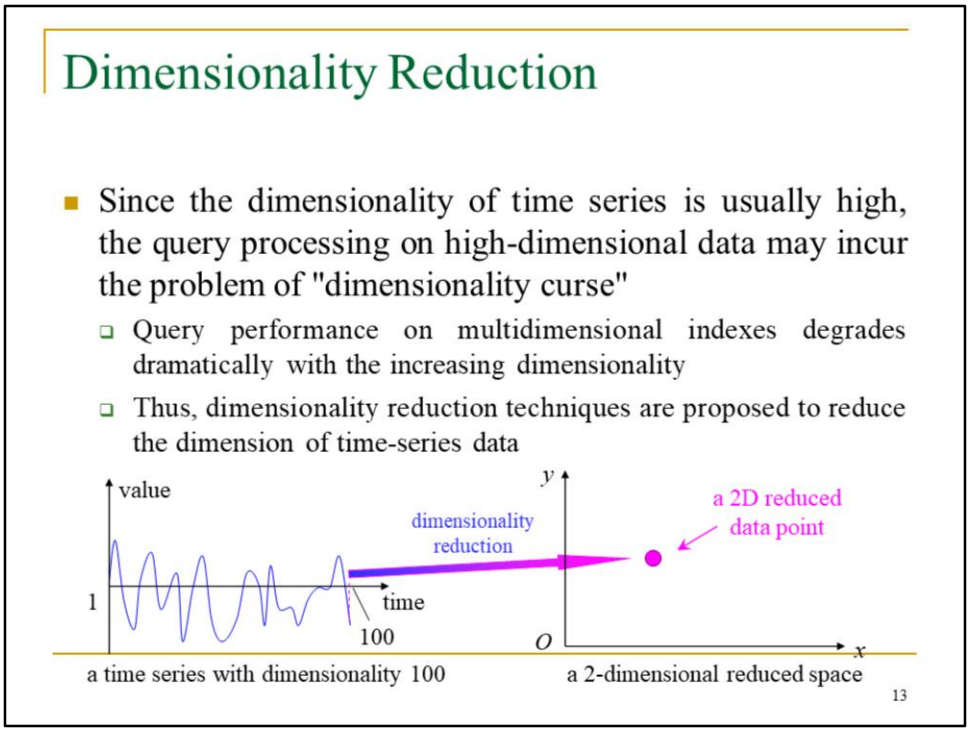
- Similarity search over time-series databases
  - The whole matching [FODO, 1993]
  - The subsequence matching [SIGMOD, 1994]
- GEMINI
  - Convert each time series S of length n into a lower dimensional space
  - Insert the reduced data into a multidimensional index, on which the similarity search is processed
    - The reduction method should follow the *lower bounding lemma* to guarantee no false dismissals

12

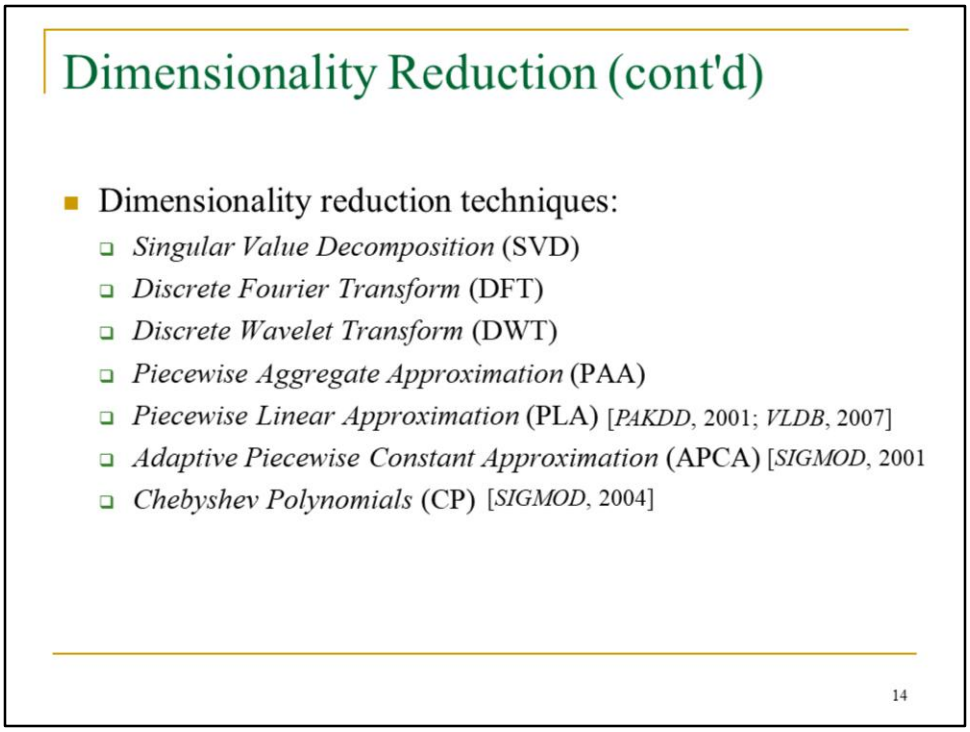
In the literature, the similarity search over time series data can be classified into two categories, whole matching and subsequence matching. The whole matching is to find the whole data sequences (i.e., the whole time series) from the database that match with the query sequence; the subsequence matching is to consider the subsequences in time-series databases that are similar to the query time series.

To enable the similarity search, we usually follow the GEMINI framework, which first reduces the high dimensionality of time series to a lower dimensional point, and then inserts the reduced data into a multidimensional index (e.g., R*-tree). Note that, here the dimensionality reduction method must satisfy the lower bounding lemma to guarantee no false dismissals of query answers (i.e., the distance between two reduced points must be smaller than or equal to that between two time series in the original space).

The similarity search is conducted on the resulting index, by first transforming the query time series into a reduced lower-dimensional point, and then searching similar (reduced) data points through the spatial index.



Due to the high dimensionality of time series and “dimensionality curse”, we need to reduce the dimensionality before indexing the time series. As shown in the figure, we can reduce a time series of length (dimensionality) 100 to a 2D reduced data point.



References

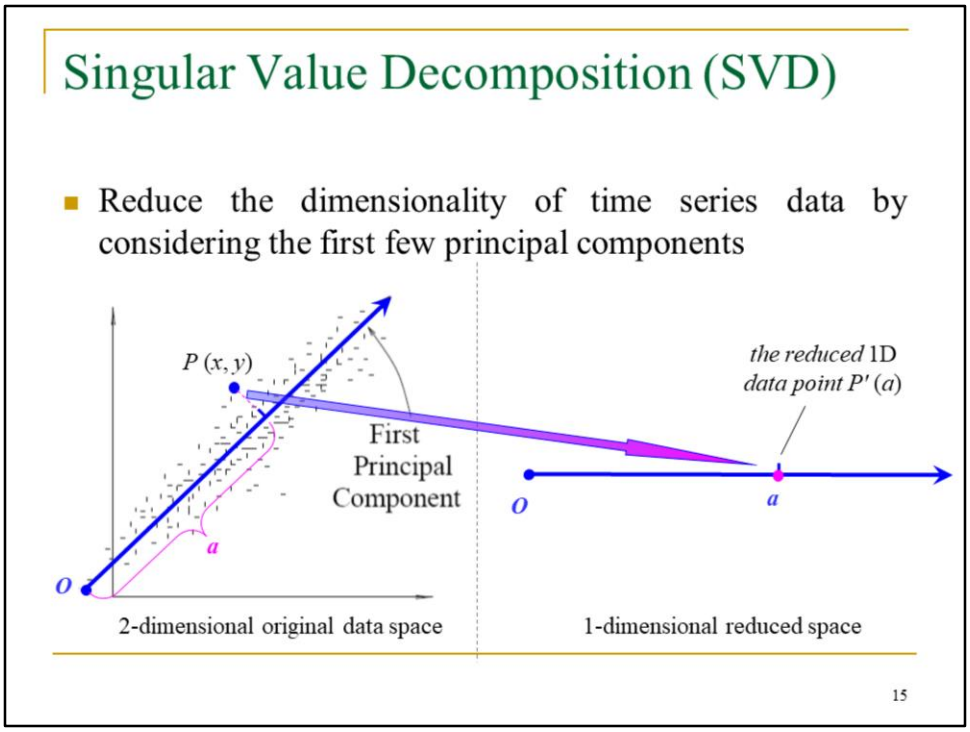
[*PAKDD,* 2001] Y. Morinaka, M. Yoshikawa, T. Amagasa, and S. Uemura. The L-index: An indexing structure for efficient subsequence matching in time sequence databases. In *PAKDD*, 2001.

[*VLDB*, 2007] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. Indexable PLA for Efficient Similarity Search. In *VLDB*, 2007.

[*SIGMOD*, 2001] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2001.

[SIGMOD, 2004] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD*, 2004.

Previous dimensionality reduction techniques include SVD, DFT, DWT, PAA, PLA, APCA, CP, etc.



References

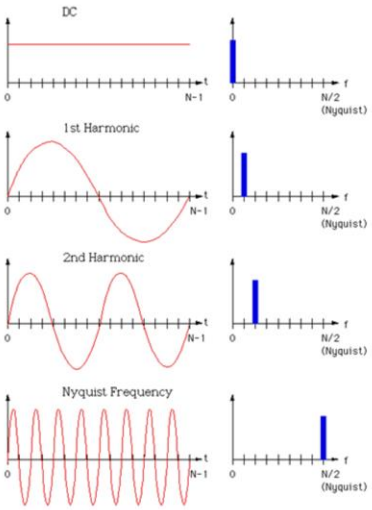
Kaushik Chakrabarti, Sharad Mehrotra: Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces. VLDB 2000: 89-100

For singular value decomposition (SVD), it reduces the dimensionality of time series by considering only the first few principal components. As shown in the figure, objects in the original 2D data space are clustered along a line. In this case, we can rotate the coordinate system, and only consider the first principal component of each object as the reduced 1D data point (i.e., each object has a projected value on the new rotated axis).

Discrete Fourier Transform (DFT)

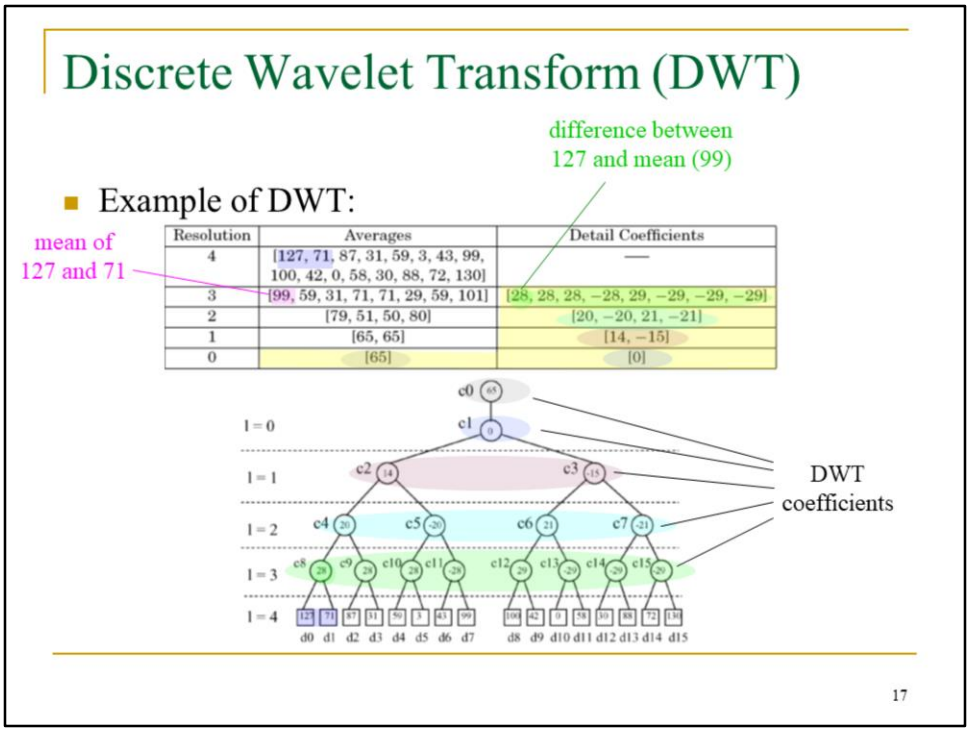
- Transform time series data from the time domain to frequency domain
  - x_t – time series data at timestamp t ($0 \le t \le n$ - 1)
  - X_f – transformed data
  - DFT:

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



16

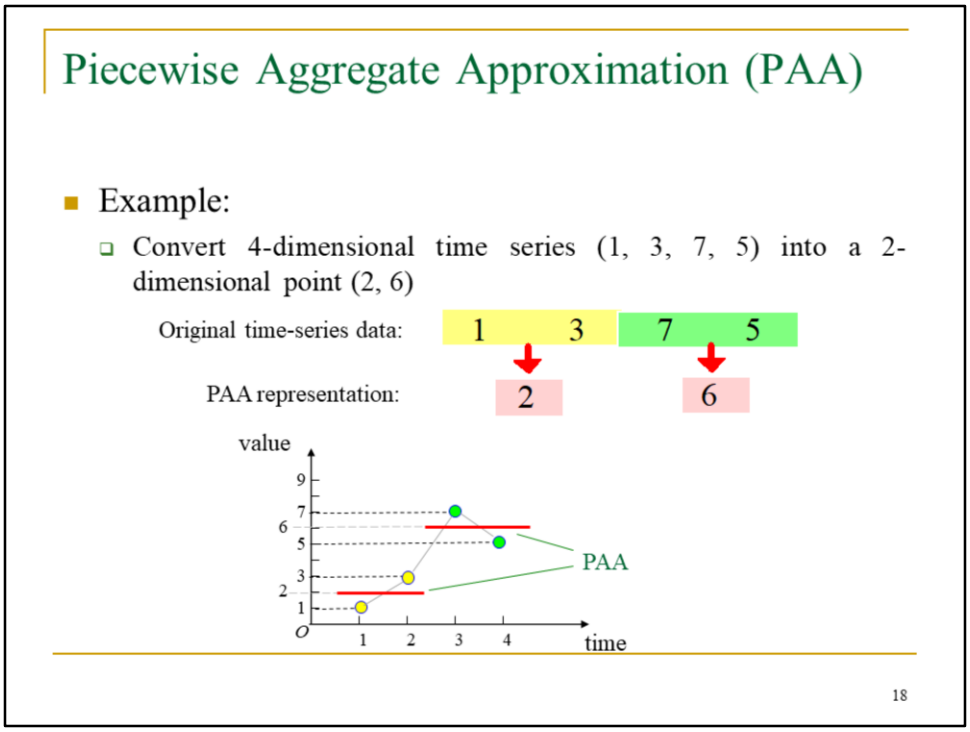
Discrete Fourier Transform (DFT) transforms the time series from the time domain to the frequency domain, and only takes a few DFT coefficients as the reduced dimensions.



References

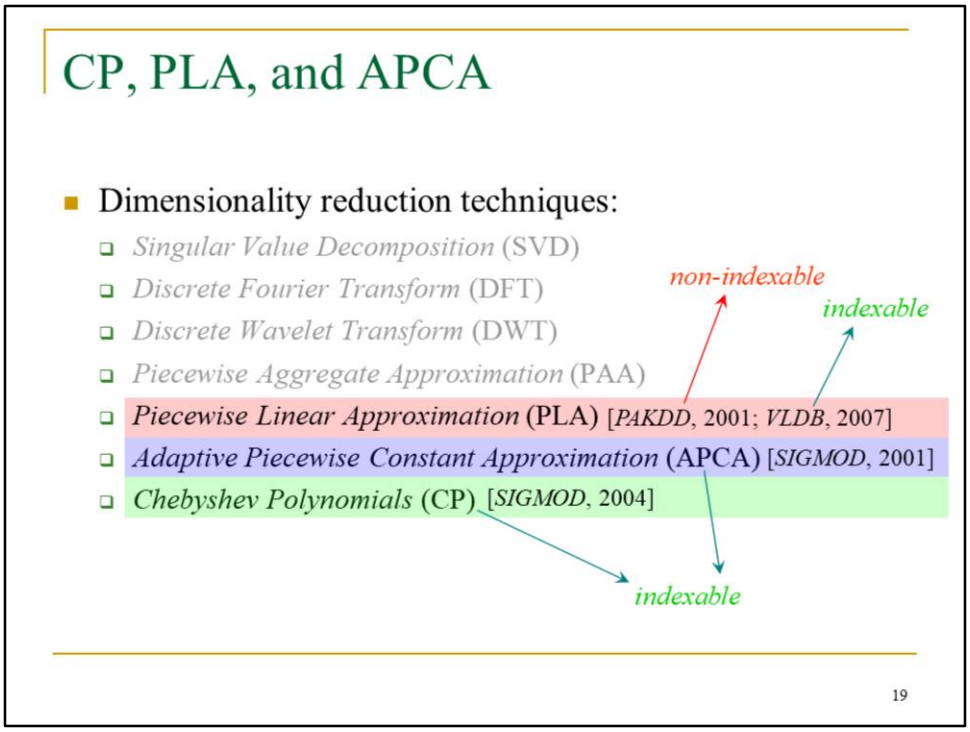
Minos Garofalakis and Phillip B. Gibbons. 2004. Probabilistic wavelet synopses. ACM Trans. Database Syst. 29, 1 (March 2004), 43-90.

Discrete Wavelet Transform (DWT) also transforms each time series to a few DWT coefficients.



The Piecewise Aggregate Approximation (PAA) divides the time series into multiple segments of the same length, and takes the average on each segment. As a result, the average values from segments form a reduced data point.

As shown in the figure, for a time series (1, 3, 7, 5) with dimensionality 4, we divide it into two segments, each of length 2, and take the average on each segment. Then, the reduced data is given by (2, 6) in a 2D reduced space.



References

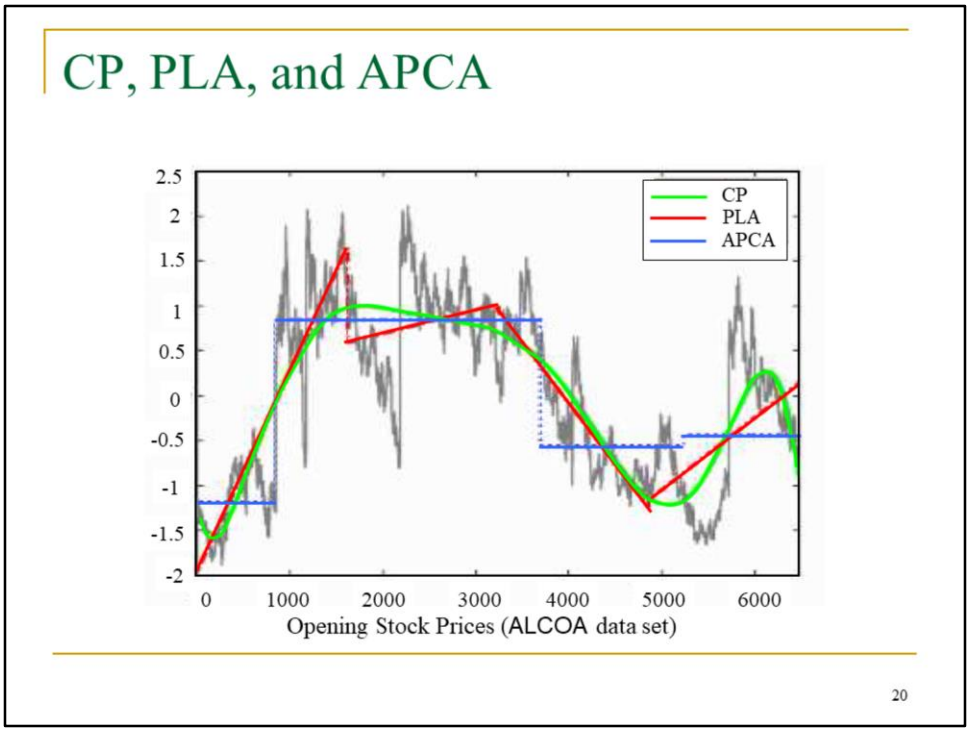
[*PAKDD,* 2001] Y. Morinaka, M. Yoshikawa, T. Amagasa, and S. Uemura. The L-index: An indexing structure for efficient subsequence matching in time sequence databases. In *PAKDD*, 2001.

[*VLDB*, 2007] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. Indexable PLA for Efficient Similarity Search. In *VLDB*, 2007.

[*SIGMOD*, 2001] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2001.

[SIGMOD, 2004] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD*, 2004.

APCA, CP, and PLA are another 3 recently proposed techniques.



Here is an example of the 3 dimensionality reduction methods on a time series.

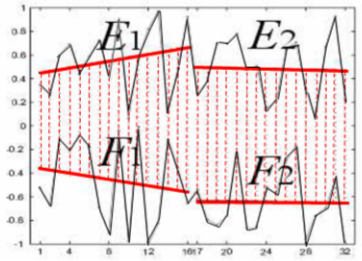
APCA adaptively divides the time series into segments of different lengths (different from PAA with the same segment length).

CP approximates time series data by Chebyshev Polynomial curves, and uses the coefficients to represent the curve. APCA and CP are indexable and we can use indexes to index the reduced data.

In contrast, PLA is to use line segments to approximate segments from time series. There is one previous work from a PAKDD 2001 paper on PLA, called L-index, which divides a time series into different lengths and is however non-indexable for the similarity search. Thus, another work on PLA in VLDB 2007 was proposed, which is indexable.

Indexable Piecewise Linear Approximation

- Divide each time series $S = <s_1, s_2, \ldots, s_n>$ into m segments of *equal length* l (i.e. $n = l \cdot m$)
- For the i-th segment, we approximate it with a line segment in the form $a_i \cdot t + b_i$ ($t \in [1, l]$), such that the *reconstruction error* is minimized
- PLA representation:
  - $S_{PLA} = <a_1, b_1; a_2, b_2; \ldots; a_m, b_m>$



$m = 2$

21

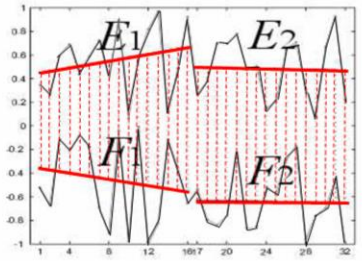
Specifically, the indexable PLA representation of time series is to divide the time series into segments of equal lengths. As shown in the figure, the two segments E1 and F1 from two time series, respectively, are of the same length. Similarly, E2 and F2 are of the same length. Note that, segments of equal lengths inherently make the PLA reduction method indexable, whereas the previous L-index method are non-indexable due to different lengths of the corresponding segments (e.g. E1 and F1).

Each segment of the time series can be approximated by a line segment that minimizes the reconstruction error. For example, the line of the i-th segment is defined in the form a_i * t + b_i. The reduced PLA data can be represented by coefficients of line segments $<a_1, b_1; a_2, b_2; \ldots; a_m, b_m>$

Lower Bounding Lemma

- PLA distance $dist_{PLA}(S, Q)$
  - $S_{PLA} = \langle a_{11}, b_{11}; \ldots; a_{1m}, b_{1m} \rangle$
  - $Q_{PLA} = \langle a_{21}, b_{21}; \ldots; a_{2m}, b_{2m} \rangle$
  - $$dist_{PLA}(S,Q) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{l}(a_{3i} \cdot j + b_{3i})^2}$$

    where $a_{3i}=a_{1i}-a_{2i}$ and $b_{3i}=b_{1i}-b_{2i}$
- *Lower bounding lemma*
  - $dist_{PLA}(S, Q) \le dist(S, Q)$
  - This lemma can guarantee the similarity search with *no-false-dismissals* over PLA index



m = 2

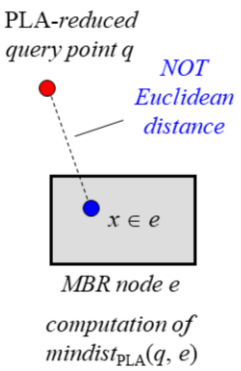
22

Given two reduced PLA data of S and Q, we define the distance between two reduced data by summing up the distance between the corresponding line segments for all segments. For example, the distance between E1 and F1 plus that between E2 and F2.

We have proved that this distance between two reduced data is a lower bound of that between original time series, which can guarantee that *no-false-dismissals* are introduced for the similarity search over PLA index.

PLA Index

- We use R-tree to index the reduced PLA data
- In order to perform the similarity search, we need to compute the minimum possible distance $mindist_{PLA}(q, e)$ from a PLA-reduced query point q to an MBR node e
  - $q = <q_{a1}, q_{b1}; \ldots; q_{am}, q_{bm}>$ and $x = <x_{a1}, x_{b1}; \ldots; x_{am}, x_{bm}>$
  - $$dist^2_{PLA}(q,x) = \sum_{i=1}^{m} \left(\frac{l(l+1)(2l+1)}{6}(q_{a_i} - x_{a_i})^2 + l(l+1)(q_{a_i} - x_{a_i})(q_{b_i} - x_{b_i}) + l(q_{b_i} - x_{b_i})^2 \right).$$



$dist^2_{PLA}(q^{(i)}, x^{(i)})$

23

In order to speed up the similarity search, we index the reduced PLA data with an R-tree. Since the distance between any two reduced PLA data is not Euclidean distance any more, the minimum distance from any PLA query point to an intermediate node of R-tree is not Euclidean distance either. Thus, we have to re-define this distance.

As shown in the figure, given a PLA query point q and an MBR node e, the minimum distance from q to e is the minimum possible distance from q to any point x in the node e.

The distance from q to x can be formalized in this complex formula, which is a summation of distances from each segment.

Calculating Minimum $dist^2_{PLA}(q^{(i)}, x^{(i)})$

- Without loss of generality, we consider the reduced data from the i-th segment, $q^{(i)}$ and $x^{(i)}$

$$dist^2_{PLA}(q^{(i)}, x^{(i)}) = \frac{l(l+1)(2l+1)}{6}(q_{a_i} - x_{a_i})^2 + l(l+1)(q_{a_i} - x_{a_i})(q_{b_i} - x_{b_i}) + l(q_b - x_b)^2$$

Goal: obtain minimum $dist^2_{PLA}(q^{(i)}, x^{(i)})$

- Interestingly, we can simplify $dist^2_{PLA}(q^{(i)}, x^{(i)})$ as:

$$dist^2_{PLA}(q^{(i)}, x^{(i)}) = (u_A - u_B)^2 + (v_A - v_B)^2$$

where

$$u_A = \sqrt{l} \cdot \frac{l+1}{2} \cdot (x_a - q_a),$$
$$u_B = \sqrt{l} \cdot (-x_b + q_b),$$
$$v_A = \sqrt{\frac{l^3 - l}{12}} \cdot (x_a - q_a), \text{ and}$$
$$v_B = 0.$$

Euclidean distance between points $A(u_A, v_A)$ *and* $B(u_B, v_B)$ *in a 2D u-v space*

constraints of points A and B

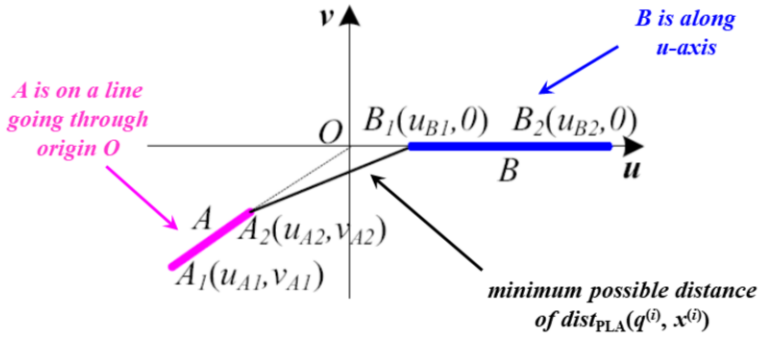
24

Without loss of generality, we consider each segment, on which the distance $dist_{PLA}^2(q^{(i)}, x^{(i)})$ for the i-th segment is given in the first formula. Our goal is to obtain the minimum possible distance $dist_{PLA}^2(q^{(i)}, x^{(i)})$, which is not trivial.

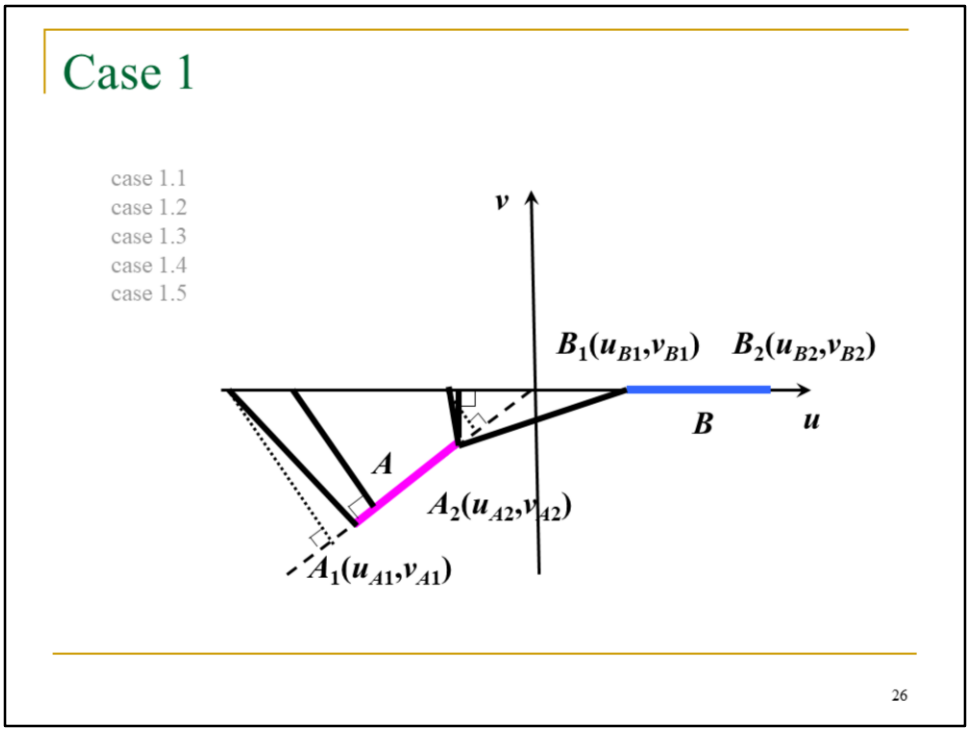
Fortunately, after simplifying $dist_{PLA}^2(q^{(i)}, x^{(i)})$, we find that it can be viewed as the Euclidean distance between 2 points A and B in a 2D u-v space, where the constraints of A and B are given in the four formulae at the bottom.

Calculating Minimum $dist^2_{PLA}(q^{(i)}, x^{(i)})$ (cont.)

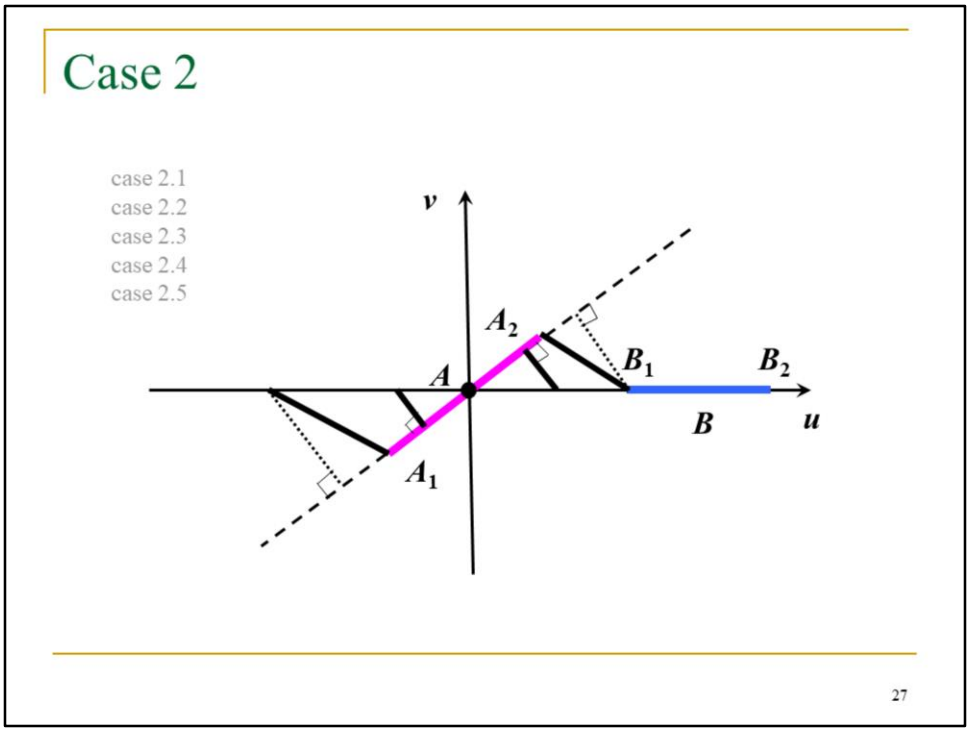
- In the *u*-*v* space, *A* and *B* are represented by two line segments, respectively



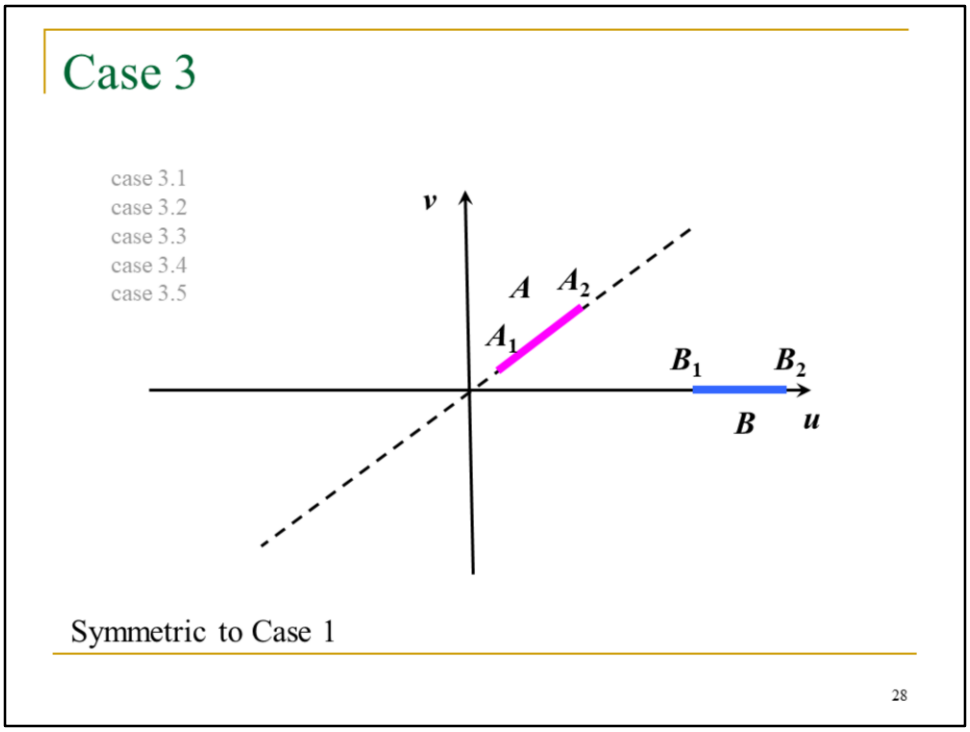
As indicated by the constraints of A and B, A is a line going through origin O, and B is along u-axis. Both A and B are line segments. Therefore, the problem of finding minimum possible distance of $dist_{PLA}^2(q^{(i)}, x^{(i)})$ is now reduced to the problem of obtaining the minimum squared distance between two line segments.



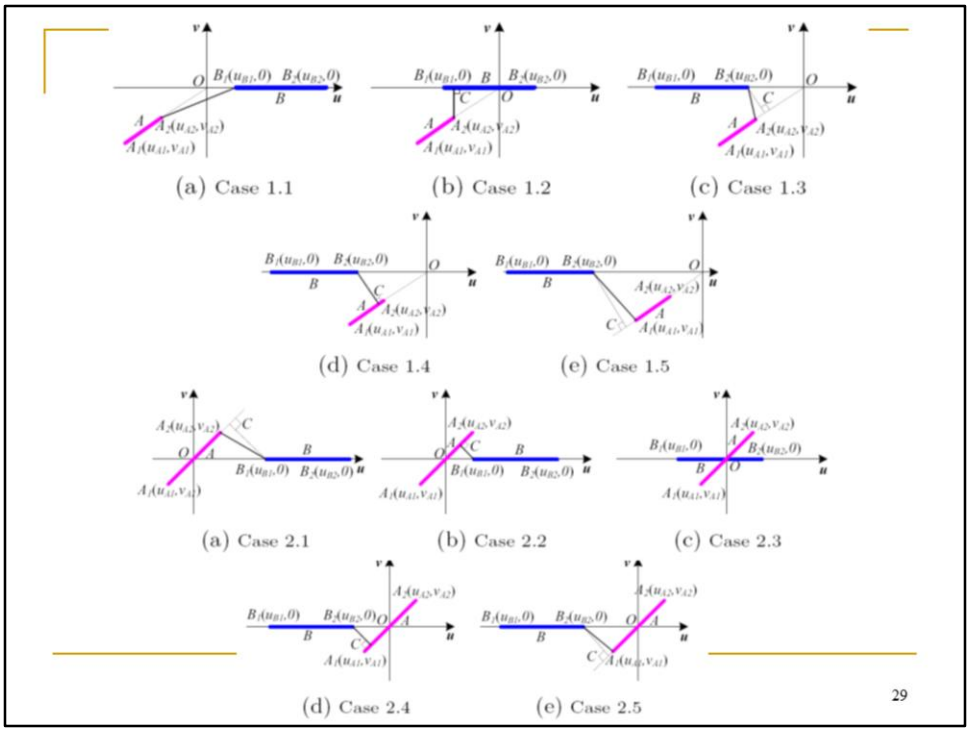
According to different relative positions of the two line segments A and B in the u-v space, we can obtain the minimum distance between two line segments with respect to different cases. For Case 1, when line segment A is in the third quadrant, line segment B can be located anywhere along u-axis.



For Case 2, line segment A spans across the first and third quadrants.



Case 3 is symmetric to Case 1. Line segment A is in the first quadrant.



In summary, we can have 10 subcases from these 3 cases.

A Summary of Different Cases

Cases	Switching Conditions	$mindist^2_{PLA}(q^{(i)}, e^{(i)})$
1.1	$u_{A2} < 0, u_{B1} > u_{A2}$	$\|A_2B_1\|^2$
1.2	$u_{A2} < 0, u_{B1} \leq u_{A2}, u_{B2} > u_{A2}$	$\|A_2C\|^2$
1.3	$u_{A2} < 0, u_{A1} \leq u_{B2} \leq u_{A2}, u_C \geq u_{A2}$	$\|A_2B_2\|^2$
1.4	$u_{A2} < 0, u_{B2} \leq u_{A2}, u_{A1} < u_C < u_{A2}$	$\|B_2C\|^2$
1.5	$u_{A2} < 0, u_{B2} < u_{A1}, u_C \leq u_{A1}$	$\|A_1B_2\|^2$
2.1	$u_{A1} \leq 0, u_{A2} \geq 0, u_{B1} > 0, u_C \geq u_{A2}$	$\|A_2B_1\|^2$
2.2	$u_{A1} \leq 0, u_{A2} \geq 0, u_{B1} > 0, u_C < u_{A2}$	$\|B_1C\|^2$
2.3	$u_{A1} \leq 0, u_{A2} \geq 0, u_{B1} \leq 0, u_{B2} \geq 0$	0
2.4	$u_{A1} \leq 0, u_{A2} \geq 0, u_{B2} < 0, u_C > u_{A1}$	$\|B_2C\|^2$
2.5	$u_{A1} \leq 0, u_{A2} \geq 0, u_{B2} < 0, u_C \leq u_{A1}$	$\|A_1B_2\|^2$

30

These cases can be summarized in this table. In this way, we can obtain the minimum distance from any query point to MBR nodes by checking a few switch cases.

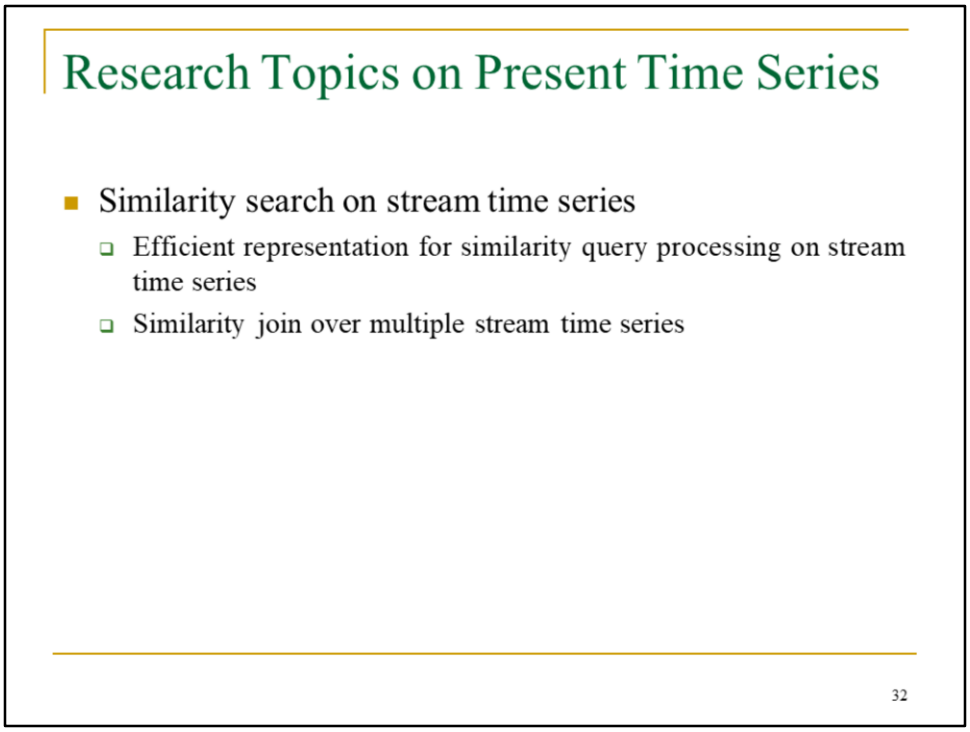
Query Processing Over PLA Index

- Now we can compute $mindist_{PLA}(q, e)$ between any PLA-reduced query point q and an MBR node e
- We answer a *k nearest neighbor* (*k*NN) query, by traversing the R-tree index in a *best-first* manner
  - Maintain a min-heap with entries in the form (e, *key*), where e is an MBR (or point) and *key* is either $mindist_{PLA}(q, e)$ (or $dist_{PLA}(q, e)$)

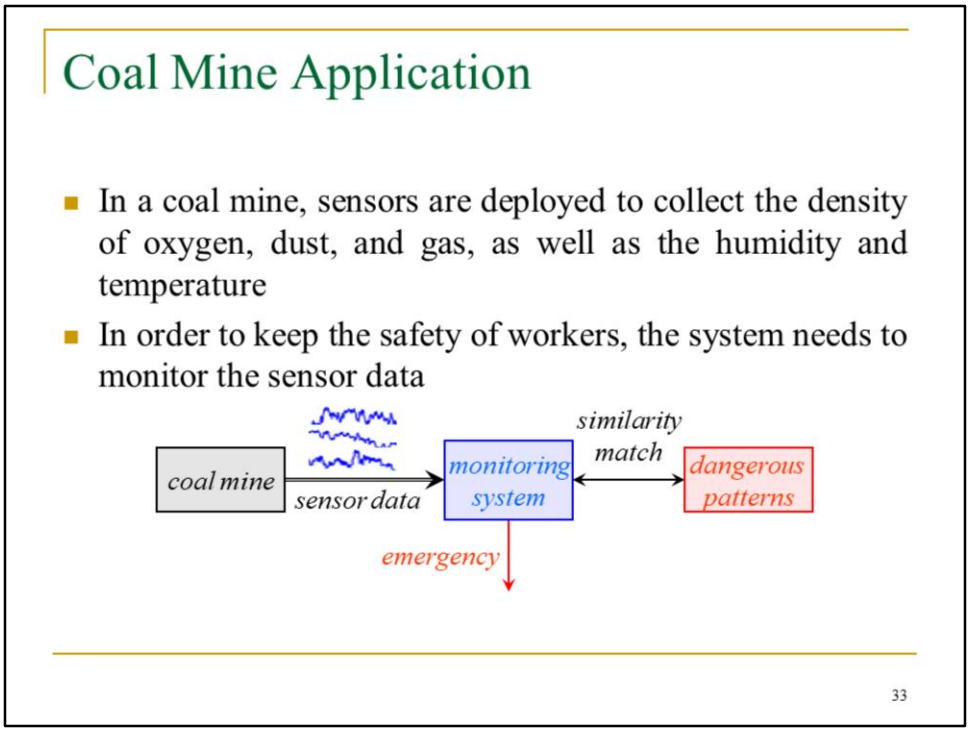
31

With this minimum distance between point and MBR node, we can answer the kNN query by traversing the R-tree index.

In particular, we maintain a minimum heap containing entries with key defined as the minimum distance from a point to node/point in the reduced PLA space and search the index in a traditional best-first manner.



So far we have talked about the static time series. There is another research direction for the similarity search on stream time series, including efficient representation of stream time series, similarity queries, and join operator over multiple time series.



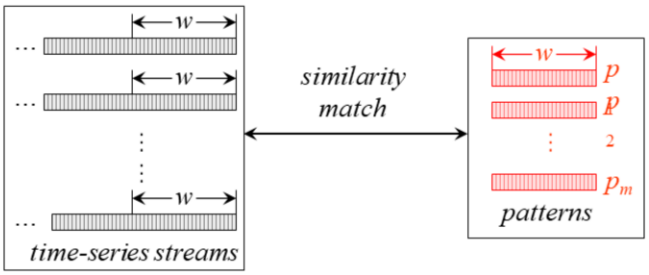
References

Xiang Lian, Lei Chen, Jeffrey Xu Yu, Jinsong Han, and Jian Ma. Multi-Scale Representations for Fast Pattern Matching in Stream Time Series. In IEEE Transactions on Knowledge and Data Engineering (TKDE), 21(4), pages 568-581, 2009.

There are many real applications for streaming time series. For example, in coal mine surveillance application, we deploy many sensors in the tunnel of the coal mine to collect time series data such as the densities of oxygen, dust, and gas, as well as temperature and humidity. The sensory data are transmitted back to the sink in a fashion of stream time series. The monitoring system needs to monitor these stream time series data by detecting dangerous events such as fire or explosion, and reporting the potential danger.

Problem Statement

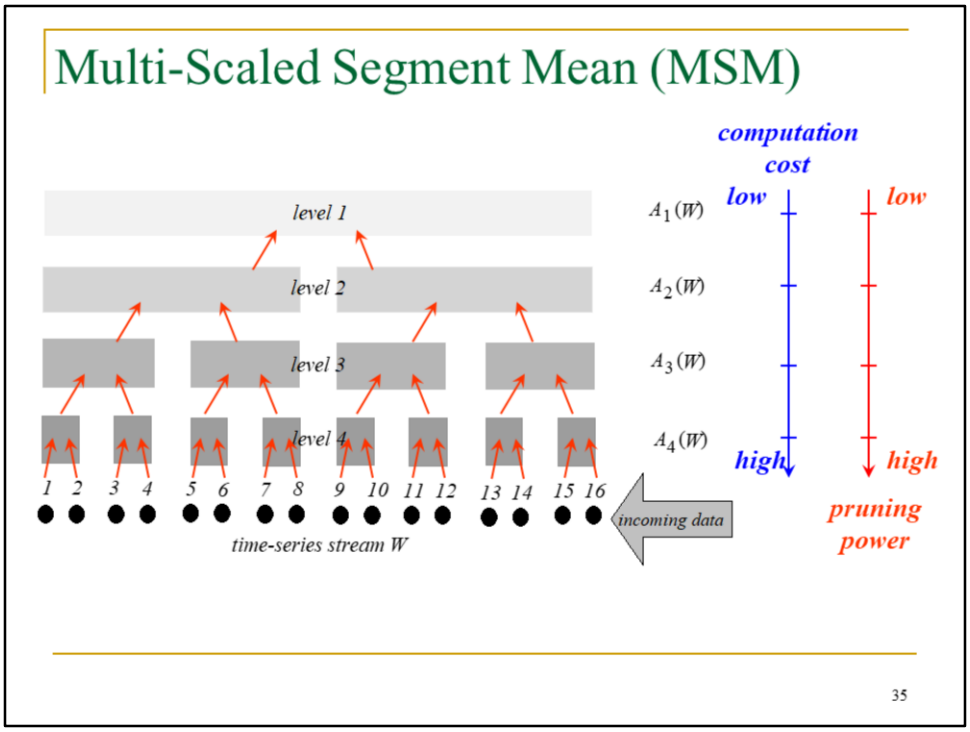
- Assume we have *time-series streams* in the form $S = (s_1, s_2, \ldots, s_i, \ldots)$, in which we consider *sliding windows* of size w, that is, $W_i = (s_i, s_{i+1}, \ldots, s_{i+w-1})$
- Given a set of pre-defined time-series patterns, $P = \{p_1, p_2, \ldots, p_m\}$, we study the problem of retrieving the *similar match* between every window W_i, for $i = 1, \ldots$, in *high speed* time-series stream S and patterns $p_i \in P$



34

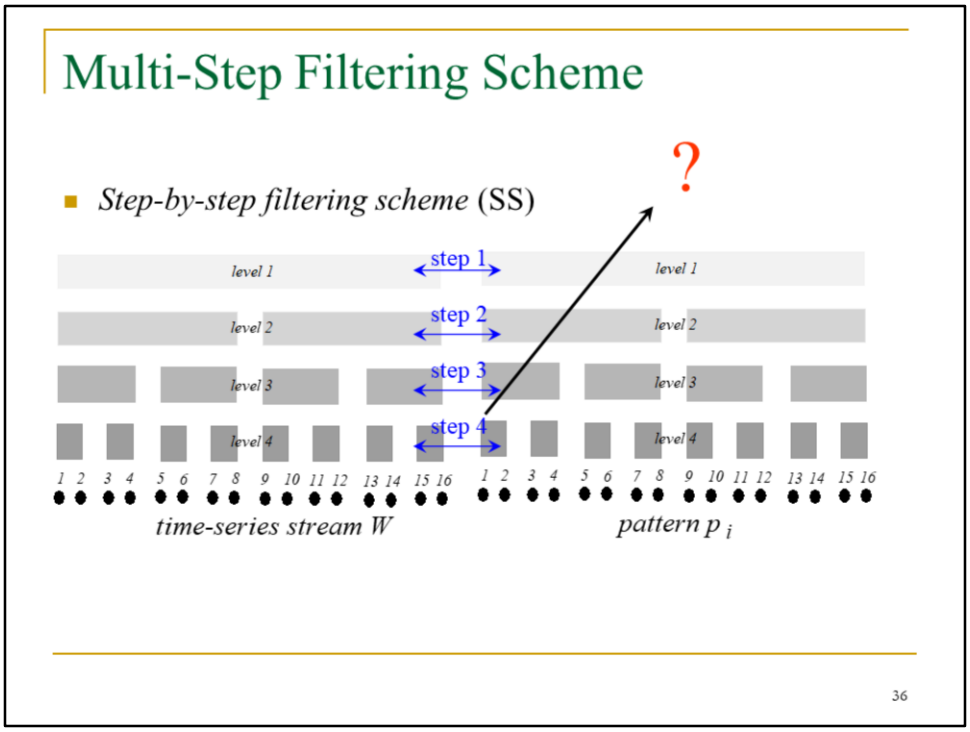
Formally, given a number of stream time series, we consider the sliding window of size w for each time series, which contains the most recent w items from the stream.

For detecting the patterns, we have *m* pre-defined query time-series patterns of size w, and the similarity match problem is to find those sliding windows from the streams that are similar to query patterns.



To summarize time series data, we can use multi-level representation. As shown in the figure, the bottom level contains 16 original values in the time series. Then, for level 4, we take the average of data at two consecutive timestamps, and obtain 8 average values. Similarly, for level 3, we can take the average on two consecutive values on level 4, and obtain 4 values, and so on. Finally, at level 1, we obtain just one value which is the average of all the 16 values in the time series.

We can see that from bottom up, there are fewer values to summarize the time series, thus, with lower computation cost. On the other hand, the pruning power becomes lower (due to the coarser resolution).



Therefore, during the similarity query processing, one straightforward method is the step-by-step filtering scheme (SS), which starts from level 1 to level 4 to enable the filtering. One interesting problem is that which level we should stop for the similarity matching (since the more levels we consider, the higher computation cost we need to consume).

Stopping Conditions of SS

- Denote $cost_j$ as the total cost of pruning from level (l_{min}+1) to level j
- Stopping conditions

  (1) We reach the last approximation level, or

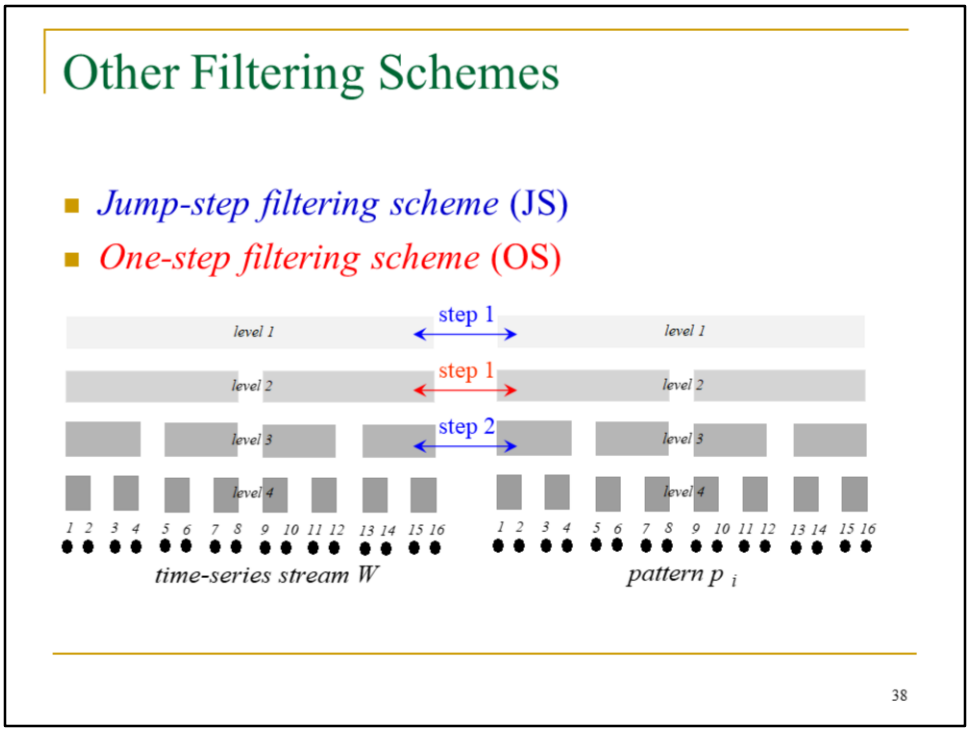
  (2) All patterns are pruned, or

  (3) $cost_j > cost_{j-1}$, where $cost_j = \sum_{i=l_{min}}^{j-1} (N \cdot P_i \cdot |P| \cdot 2^i \cdot C_d) + N \cdot P_j \cdot |P| \cdot w \cdot C_d$

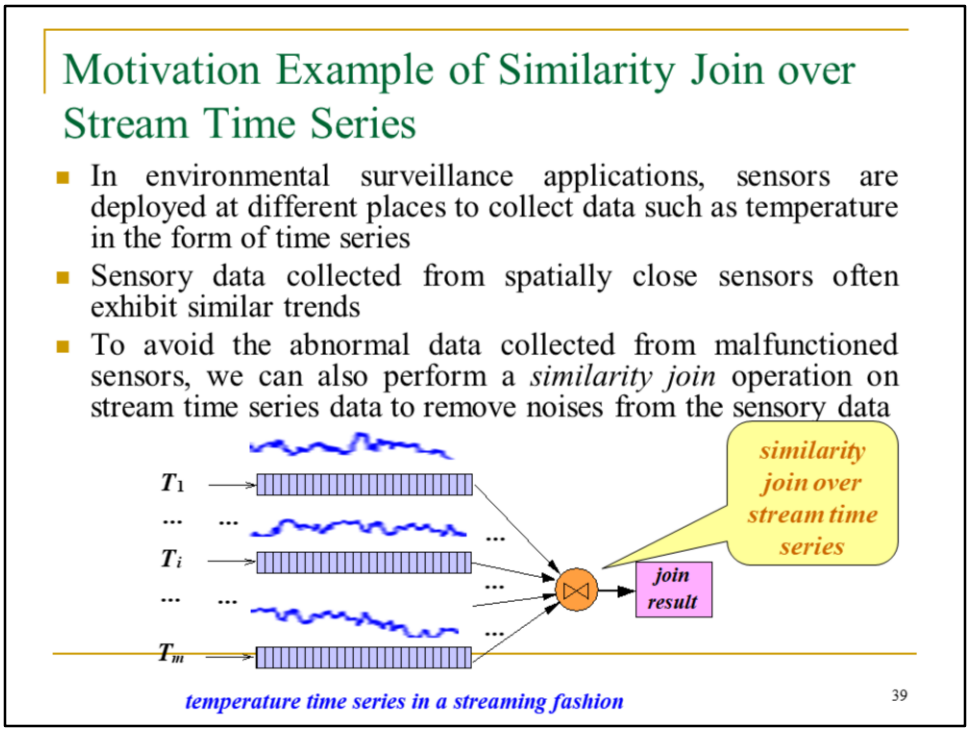
$$\log \frac{P_{j-1} - P_j}{P_{j-1}} < (j - 1 - \log(w))$$

37

To decide the stopping condition, we need to derive a cost model about the searching cost, cost_j, from level (l_min+1) to level j. Then, if cost_j on level j is greater than that on level j-1, then we should stop searching for the next level j. The stopping condition can be given by the inequality in the slide.



There are some other possible schemes such as jump-step filtering (JS) or one-step filtering scheme (OS). For JS, since the first few levels definitely have low pruning power, the JS problem is that we may directly jump to an intermediate level to check the matching. The OS scheme is to directly estimate the appropriate level to check the similarity.



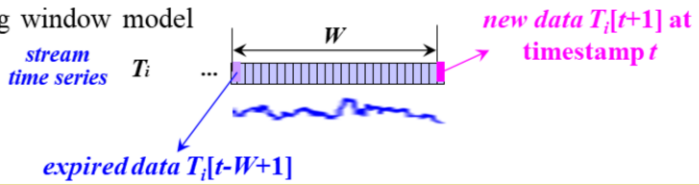
References

Xiang Lian and Lei Chen. Efficient Similarity Join over Multiple Stream Time Series. In IEEE Transactions on Knowledge and Data Engineering (TKDE), 21(11), pages 1544-1558, 2009.

So far we have talked the similarity matching over a single stream time series. It is also important to study the join operator over multiple time series. This join operator has the real applications such as sensor data analysis from the coal mine. We may deploy multiple sensors in spatially close locations, whose collected sensory data should exhibit similar and correlated trends. In order to avoid the abnormal data from malfunctioned sensors or avoid false alarms for sensor data monitoring, we may conduct a similarity join operation to find similar patterns from multiple stream time series, which can be used for cleaning data.

Similarity Join on Stream Time Series

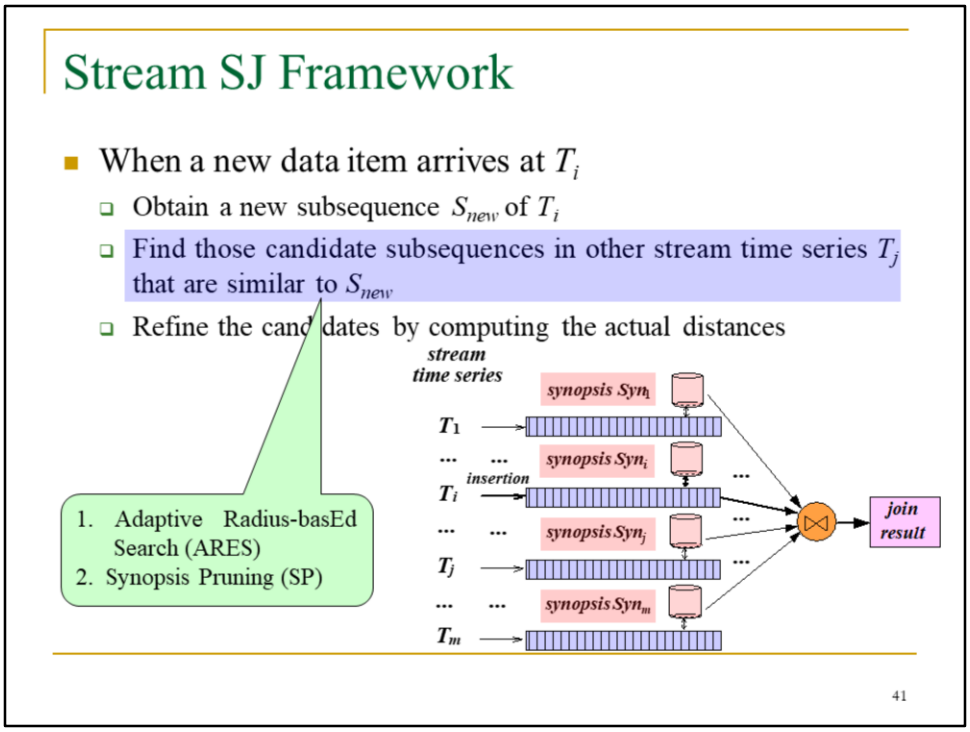
- Similarity Join (SJ)
  - Given two time-series databases R and S containing (sub)sequences of length n, an SJ query outputs all pairs $<r, s>$, such that $dist(r, s) \leq \varepsilon$, where $r \in R$, $s \in S$, $dist(\cdot, \cdot)$ is a distance function between two series and ε is a similarity threshold
- Stream Time Series
  - The time series data continuously arrive at the system
  - Sliding window model



40

The similarity join has been studied in static time-series databases, which retrieves pairs (r, s) from two time-series databases, such that r and s are similar to each other, that is their distance is within \epsilon threshold.

In the scenario of stream time series where time series data continuously arrive at the system, we will consider the sliding window model, which always consider the most recent W data items from stream time series. To dynamically maintain the sliding window, as shown in the figure, at a new timestamp when a new data item T_i[t+1] arrives, we insert it into the sliding window and evict the oldest one T_i[t-W+1] from the sliding window.

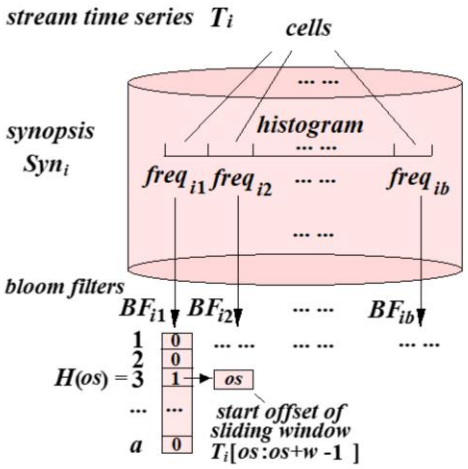


Here is a framework for stream similarity join.

When a new data item arrives at a stream time series T_i, we will first obtain the new subsequence S_{new} from the series, and find all candidate subsequences from other stream time series that may match with S_{new}. We will discuss later the search strategy and pruning methods for the join operator. Finally, we will refine candidate pairs by computing their actual distances and obtain actual matching pairs from streams.

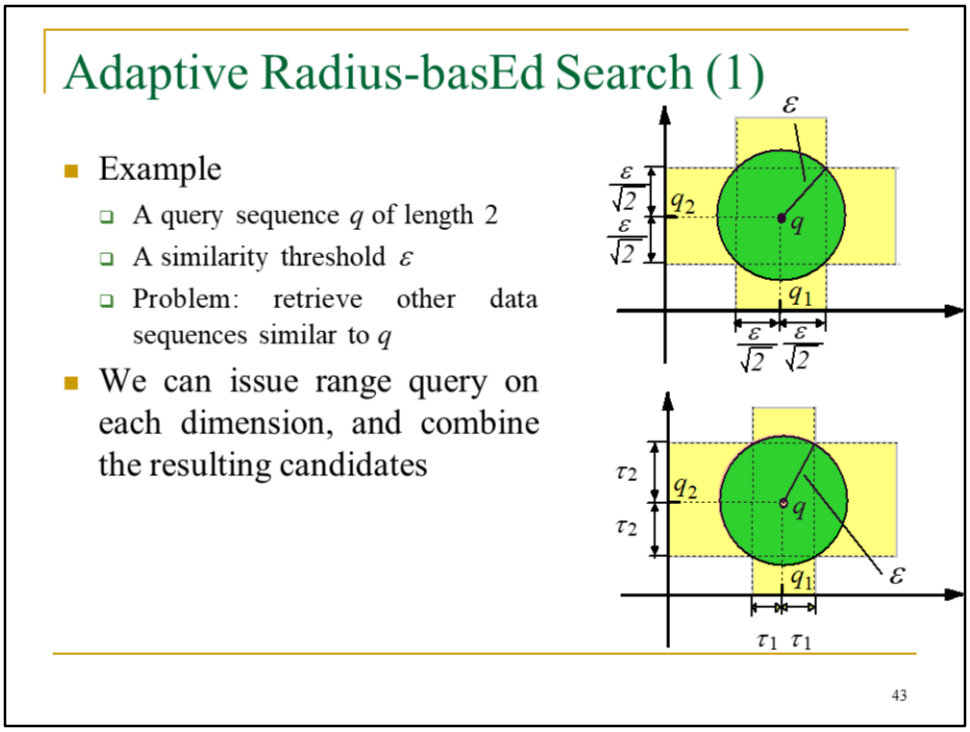
Synopsis

- Equi-width histogram
  - Each bin stores the frequency of data reduced from subsequences
- Bloom filter
  - Each bin also contains a *bloom filter* with start offsets of sliding windows



42

In order to speed up the search in the stream environment, we can design a space-efficient synopsis to facilitate the join operator. Here, we use an equi-width histogram, where each bin stores the frequency of data reduced from subsequences via dimensionality reduction techniques. We also store a bloom filter in each bin, which records the start offsets of those sliding windows.



To search for the candidate pairs that have distances within \epsilon, instead of directly finding matching pairs in the multidimensional space, we can search for candidate pairs by considering each dimension separately, and then combine the resulting candidate pairs.

As shown in the top figure, instead of search for all subsequences within \epsilon distance away from a query sequence q, we can search for candidates within \epsilon/sqrt(2) away from q_1 on the x-axis and q_2 on the y-axis, where q_1 and q_2 are the projected values of query point q on the 2 dimensions.

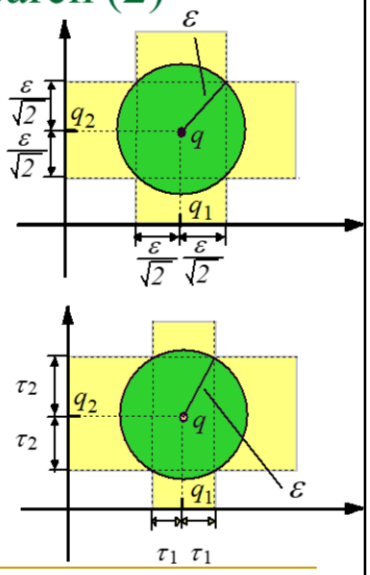
From the top figure, we can see that subsequences within the green circle are our actual answers, and that in yellow part are false alarms. Here, we use the same search radius \epsilon/sqrt(2) on both dimensions. However, if the data distribution is not uniform, it is possible to include many false alarms in the candidate set. Inspired by this, we can propose an adaptive radius-based search approach (called ARES). As shown in the bottom figure, we use radius \tau_1 for x-axis, and \tau_2 for y-axis.

Adaptive Radius-basEd Search (2)

- To guarantee the property of *no-false-dismissals*, the selected radii, τ_i, of range queries should satisfy the condition:

$$\boxed{\sum_{i=1}^{s} \tau_i^2 \geq \varepsilon^2}$$

- Since stream data are changing all the time, we propose a cost model to guide the selection of radius on each dimension *adaptively* in order to retrieve small number of candidates



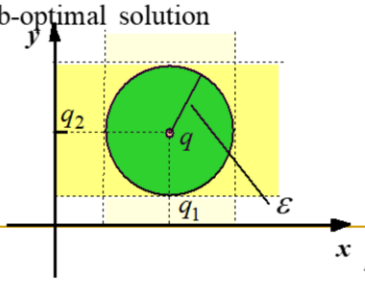
44

We can prove that as long as the search radius on each dimension can collaboratively cover the search circle, then there is no false dismissal for the search results. In the general case of s-dimensional space, we have the formula of no-false-dismissal condition as shown in the slide.

Since stream time series data are dynamically changing, we need to design a cost model to guide the selection of adaptive radii on each dimension, such that the total number of the retrieved candidates is minimized.

Adaptive Selection of Radii

- We provide a cost model for ARES, and prove that:
  - Assume data points are uniformly distributed within ε distance from each query dimension q_i with density d_i, where $1 \leq i \leq s$
  - In order to obtain the *minimum number of candidates*, we always find a query dimension q_j with the minimum density d_j and set τ_i to 0, if $i \neq j$; ε, otherwise
  - For non-uniform data, we give sub-optimal solution
- Example
  - If the density of x-axis is smaller than y-axis, then we set $\tau_1 = \varepsilon$ and $\tau_2 = 0$



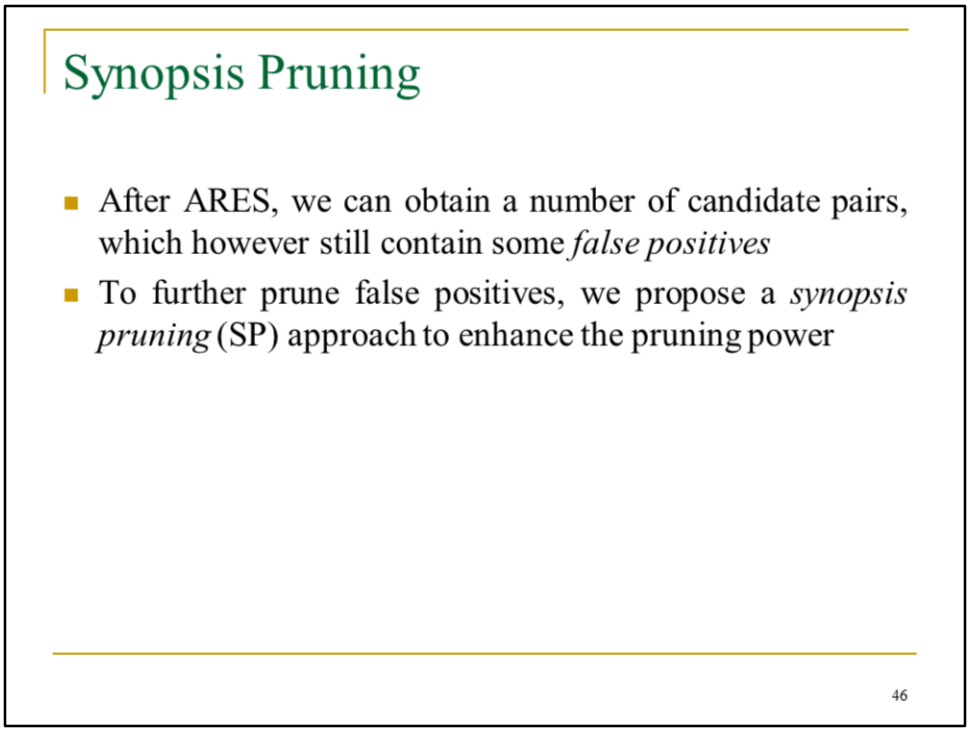
45

References

Xiang Lian and Lei Chen. Efficient Similarity Join over Multiple Stream Time Series. In IEEE Transactions on Knowledge and Data Engineering (TKDE), 21(11), pages 1544-1558, 2009.

In the reference paper (TKDE 2009), we can estimate the density, d_i ,of data within \epsilon distance away from q_i on each dimension i. It is proved that if data are uniformly distributed within the search region on each dimension I, then, we should always set a radius \tau_j of a query dimension j with the minimum density d_j to \epsilon, and set other radii \tau_i to 0 for other dimensions i.

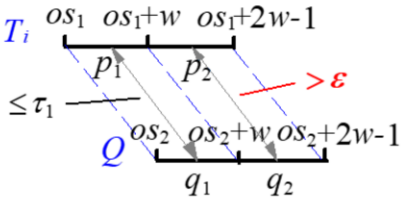
For non-uniform data, we can give sub-optimal solution.



After ARES, we can obtain a number of candidate pairs, which however still contain some *false positives*. To further prune false positives, we can utilize a *synopsis pruning* (SP) approach to enhance the pruning power.

Synopsis Pruning (cont'd)

- Basic Heuristics
  - Assume we have two time series T_i and Q,
  - For each series, we divide it into two segments, in which data is reduced
  - We can see that if the distance between the second segment of two series is already greater than ε, then we can safely discard this pair

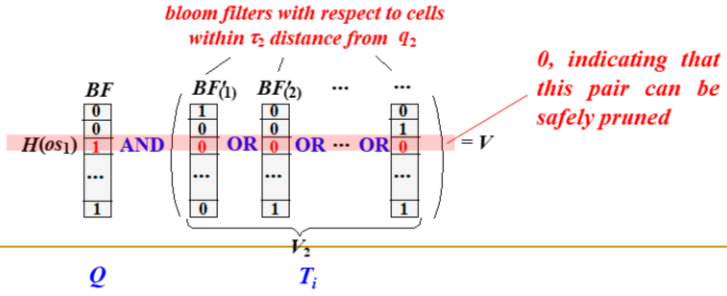


47

The basic idea of the synopsis pruning is to divide the time series into multiple segments, and we can safely prune a pair of time series, if at least one pair of segments has the distance greater than \epsilon.

Synopsis Pruning (cont'd)

- To enable such pruning, we utilize the bloom filters in synopsis Syn_i
- We choose hash functions such that start offsets of consecutive segments are mapped to the same position
- This way, we can perform the bit operation



48

To enable the pruning, we utilize bloom filters to check if candidate pairs from segments have consecutive offsets (i.e., whether they can form a matching pair of complete subsequences). We choose has functions such that start offsets of consecutive segments should be mapped to the same position. They way, we can use bit operations to quickly prune false alarms.

In the figure, for the first segment of Q, the start offset os_1 is mapped to position H(os_1) of bloom filter BF. However, for the same location, the candidates of the second segment (within \tau_2 distance from q_2) do not have start offset os_1+w. In this case, we can safely prune this pair.

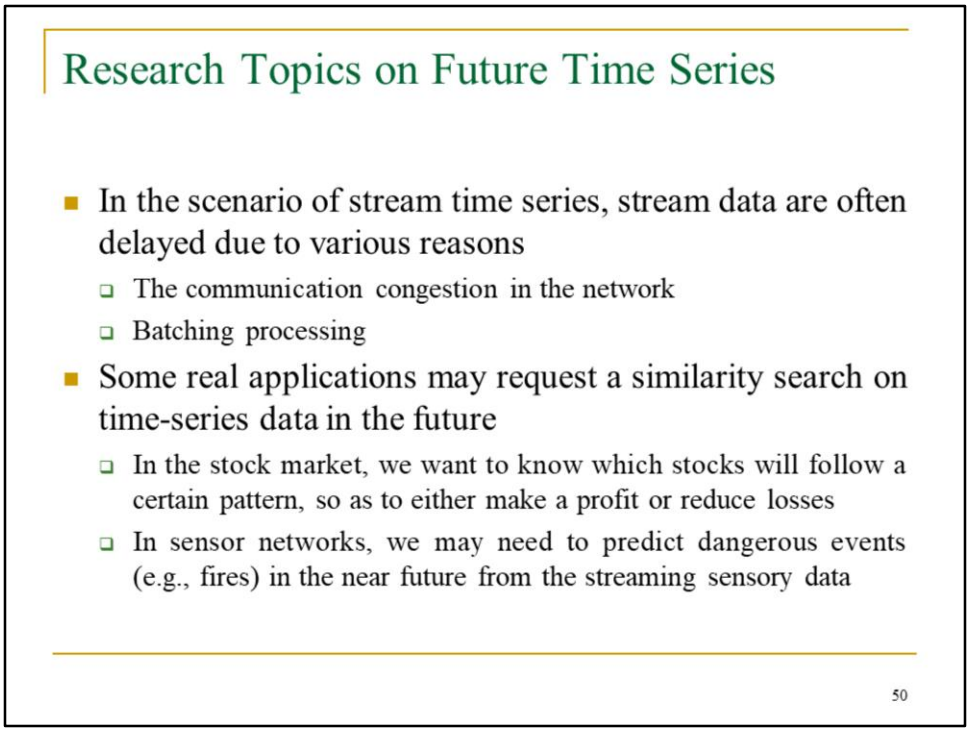
SJ over Multiple Stream Time Series

- When a new data item arrives at T_i
  - Obtain a new subsequence S_{new} of T_i
  - Update the synopsis Syn_i
  - for each stream time series T_j $(1 \le j \le m)$
    - $cand$ = **ARES**(S_{new}, T_j , ε);
    - $cand''$ = **Synopsis_Pruning** ($cand$, Syn_j);
  - Refine the candidates in set $cand''$ by computing the actual distances

1. Adaptive Radius-basEd Search (ARES)
2. Synopsis Pruning (SP)

49

Here is the entire process of similarity join over multiple stream time series. For a newly arriving data item of series T_i, we obtain the new subsequence S_{new}, update the synopsis of T_i. Then, for each stream time series T_j, we use ARES to find all candidates that may match with S_{new}, and apply the synopsis pruning to rule out false alarms. Finally, we refine the remaining candidate pairs and return the actual join results.



References

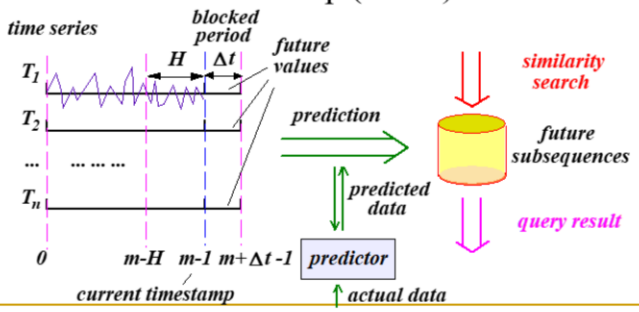
Xiang Lian and Lei Chen. Efficient Similarity Search over Future Stream Time Series. In IEEE Transactions on Knowledge and Data Engineering (TKDE), 20(1), pages 40-54, 2008.

Next, we will talk about research topics for the future time series.

In the context of stream time series, stream data are often delayed for reasons such as network congestion or batch processing. So the time series data are sometimes missing and not available in this case. The similarity search on such incomplete series data can give inaccurate results. Therefore, we may need to design some approaches to predict the missing or even future time series.

Problem Definition and Framework

- Assume we have n stream time series $T_1, T_2, \ldots, T_n$
- For each stream time series T_i, at the current timestamp $(m\text{-}1)$, we know the most recent H data $t_0, t_1, \ldots$, and t_{m-1}
- Data $t_m, t_{m+1}, \ldots, t_{m+\Delta t-1}$ from each stream time series will arrive at the future timestamp $(m+\Delta t)$ in a batch



51

Motivated by this, we have the framework for the similarity search over future time series.

As shown in the figure, we have *n* stream time series. Assume at the current timestamp, we know the most recent *H* values of each stream time series. Because of the delay, the future Δt values will not arrive until Δt timestamps later.

Our similarity search problem is to find future subsequences that are similar to a query sequence. Here, future subsequences are those that contain future values that have not arrived.

Therefore, we have two goals:
(1) predict future values with a predictor, and
(2) perform similarity search on future subsequences

After the actual Δt data arrive, we may need to train the predictor with as low cost as possible.

Problem Definition and Framework (cont'd)

- Our goal:
  - The prediction problem:
    - Given H consecutive data x_1, x_2, …, and x_H, predict the subsequent Δt values x_{H+1}, x_{H+2}, …, and $x_{H+\Delta t}$
  - Similarity search over $n \cdot \Delta t$ *future subsequences*
- Solutions:
  - Polynomial predictions and probabilistic prediction
  - Index the predicted future time series and perform the search on future time series

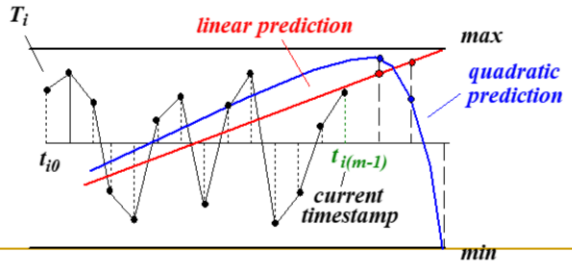
52

For the prediction problem, there are some solutions to predict future Δt values based on the previous H consecutive data. The prediction can be based on polynomial curve or probabilistic prediction.

For the search over the predicted time series, we can build indexes for over the predicted future time series and speed up the search efficiency.

Data Predictions

- Polynomial Predictions
  - Basic idea:
    - Use a curve to approximate the most recent data with the smallest *approximation error*, and
    - Predict future values with the curve
    - Linear prediction: $x = a \cdot t + b$
    - Quadratic prediction: $x = a \cdot t^2 + b \cdot t + c$



53

For the polynomial prediction, we can use a polynomial curve to approximate the trend of recent data with the smallest approximation errors. This curve can be either a linear or a quadratic curve, as shown in the example of the figure.

Discussion on Polynomial Predictions

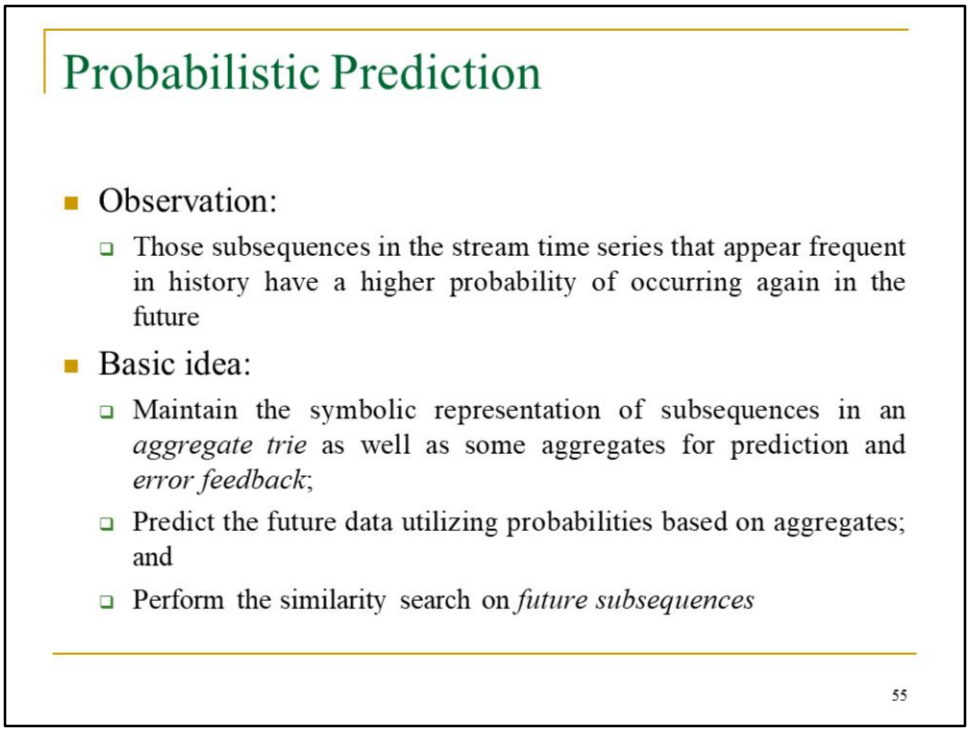
- Advantages:
  - Efficient for online processing
  - They can achieve acceptable prediction accuracy
- Drawbacks:
  - They can only predict short-term values with small error, but not long-term ones
  - They make use only H most recent data, but not all of the historical data. Therefore, values are predicted locally, but not from the global point of view

54

The polynomial approach is efficient for online processing and can achieve acceptable accuracy for short-term prediction.

However, it is not suitable for long-term values. For example, in the previous slide, the predicted values of the line may go out of the value range [min, max].

Furthermore, it makes use of the most recent data. So values are predicted locally. Next, we will discuss a probabilistic approach which can predict from the global point of view.



The probabilistic prediction is based on the observation that, subsequences that appear frequent in history have higher chance to appear again in the future.

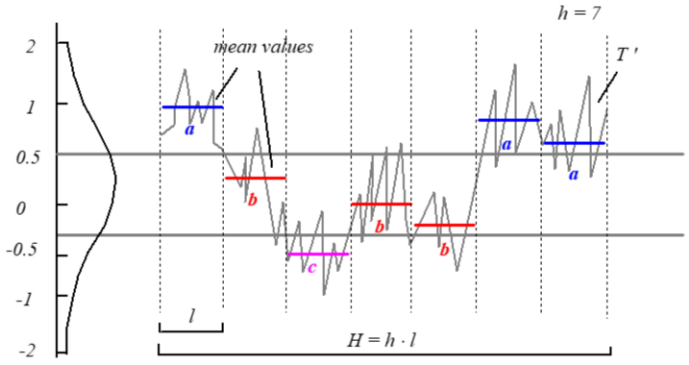
Therefore, in the probabilistic approach, we maintain the symbolic representation of historical subsequences in a special index called *aggregate trie*, which contains some aggregates for prediction and its feedback.

As a second step, we predict the future data from the aggregate trie, and

The predicted future subsequences are used for similarity search.

Symbolic Representation

- Symbolic representation of time series:
  - *abcbbaa*



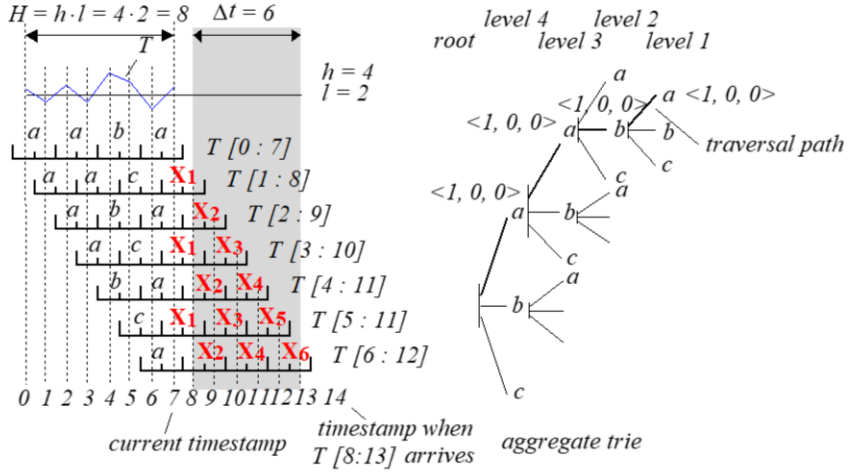
56

Before we continue to discuss details of the aggregate trie, we first introduce the symbolic representation of a sequence.

As an example in the figure, we divide a time series into 7 segments of equal length. Within each segment, we take the average and map the mean value to a unique symbol. For example, in the first segment of the figure, the mean value 1 is mapped to symbol *a*. Therefore, the entire time series is discretized to a string with 7 symbols abcbbaa.

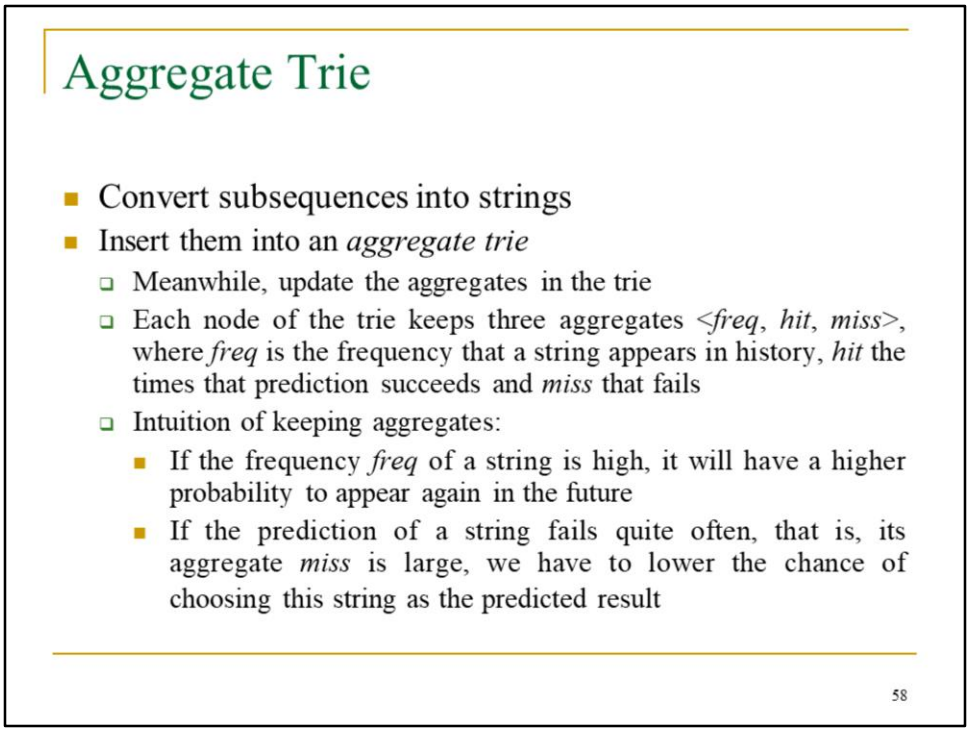
The symbolic representation is able to approximate a time series with small space cost, since each symbol can be represented by only a few bits.

Illustration of Predictions



57

This is an example of the symbolic representation of sliding windows in the time series.



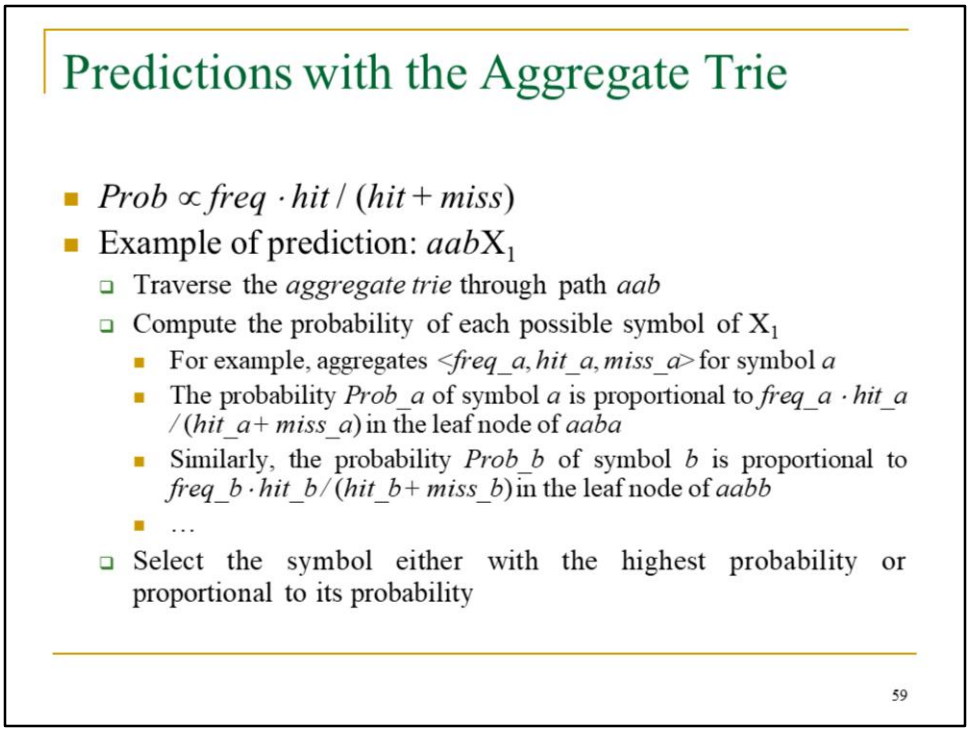
We convert subsequences in the stream time series into their symbolic representations, i.e. strings.

Then, we insert them into an aggregate trie.

Here, the aggregate trie is different from the normal one, in the sense that it contains an aggregate triple in each node, that is, <*freq*, *hit*, *miss*>. In particular, *freq* is the frequency that a string appears in history, *hit* the times that prediction succeeds and *miss* that fails.

Intuitively, if the frequency of a string is high it will have a higher probability to appear again in the future.

Furthermore, if our prediction always fails for a string, which means its *miss* aggregate is large, then we are likely to predict this string wrongly again next time.



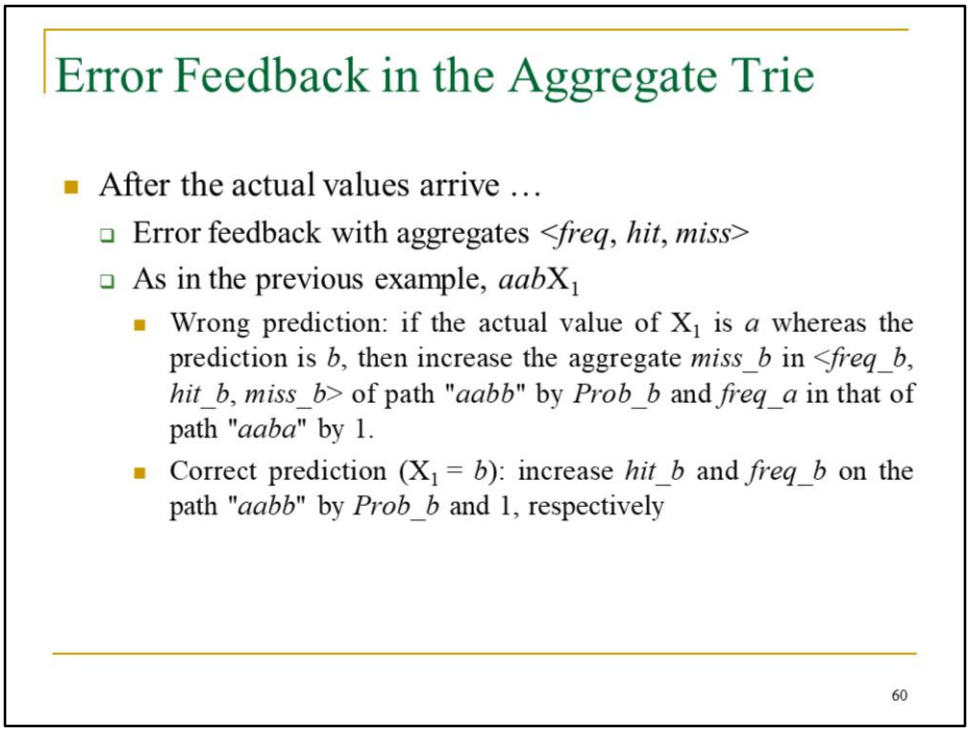
So we predict the future data according to the probability that it is likely to be.

Specifically, we define this probability by $freq \cdot hit / (hit + miss)$

As an example (in the figure of the next slide), assume currently we know symbols *aab*, and want to predict the next symbol X_1.

We traverse the aggregate trie through path *aab*. Then, we calculate the probability of each possible value of X_1.
For example, we obtain aggregates <*freq_a*, *hit_a*, *miss_a*> for symbol *a*, and compute the probability proportional to $freq\_a \cdot hit\_a / (hit\_a + miss\_a)$. Symbol *b* and *c* are similar.

Finally, we select the symbol as X_1 with the highest probability or proportional to its probability



After the actual values arrive, we need to feed back the prediction error.

Recall that if the prediction of a string fails quite often, we increase the *miss* aggregate so as to lower the chance of choosing this string as the predicted result.

Therefore, for a wrong prediction in the trie, e.g. *b*, we increase the aggregate *miss_b* in <*freq_b*, *hit_b*, *miss_b*> by *Prob_b*, and for the actual symbol *a*, increase *freq_a* in <*freq_a*, *hit_a*, *miss_a*> by 1.

For the correct prediction, simply increase *hit_b* and *freq_b* by *Prob_b* and 1, respectively

Alternative Index for the Aggregate Trie

- In the trie, each path from the root to leaf is unique, and thus can be mapped to a unique *multidimensional* key
  - For example, if there are totally 10 symbols $a_0 \sim a_9$, we map them to integers $0 \sim 9$, respectively
  - String $a_2a_3a_9$ corresponds to a unique key <2, 3, 9> of a time series
  - Therefore, any entry such as <$a_2a_3a_9$X, *freq_X*, *hit_X*, *miss_X*> in the trie can be represented by a key <2, 3, 9, *tid*> and aggregates <*freq_X*, *hit_X*, *miss_X*>, where *tid* is the series id
- Instead of tries, we index strings in an extended hashing file **HI** [6] to facilitate the prediction
- Furthermore, a B$^+$-tree **FI** is used to index *future subsequences* and help the similarity search as well as updating aggregates

61

Next, we discuss the alternative index for the aggregate trie.

Note that, the aggregate trie is capable of predicting one stream time series only. In the case of multiple time series, one possible way is to maintain multiple separate tries, however, considering thousands of stream time series, it is very costly. Furthermore, the height of tries is another problem which may incur high I/O cost. Motivated by this, we aim at build up efficient indexes to reduce the I/O cost.

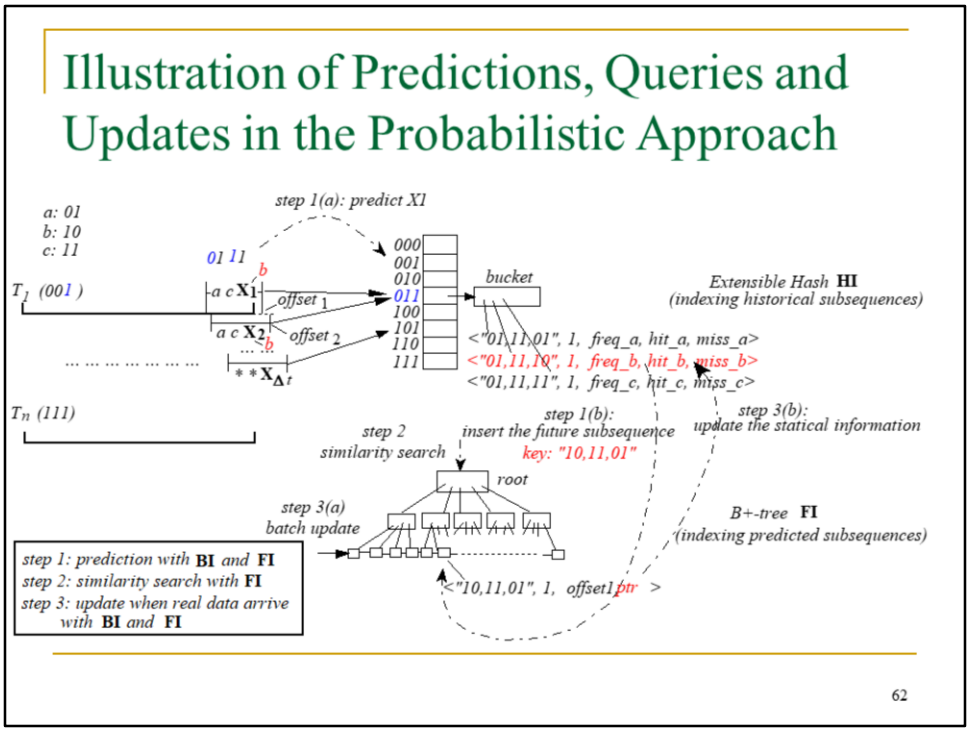
Observe that, each path of the trie can be uniquely represented by a sequence of symbols. We can map each path to a multidimensional key.

As an example, 10 symbols $a_0 \sim a_9$ can correspond to integers $0 \sim 9$. So a string $a_2a_3a_9$ can be represented by a key <2, 3, 9>. Together with an id of the stream time series *tid*, each path of a series can be uniquely identified by a key, for example <2, 3, 9, *tid*> .

Therefore, instead of tries, we index strings in an extended hashing file with their corresponding keys and aggregates mentioned in the trie. Prediction procedure is similar to that in the aggregate trie.

Another index we use is a B$^+$-tree which stores future subsequences for the similarity search and error feedback.

Ps: keys of **BI** are from



This figure illustrates the three steps of the probabilistic approach in the next slide. The top index **HI** is the extended hash index used for prediction, and the bottom one **BI** is a B^+-tree with future subsequences.

Step 1(a). At the current timestamp, for each stream time series, predict *future subsequences* with the extended hash index **HI**

Step 1(b). Insert the predicted strings of *future subsequences* into the B^+-tree **FI**

Step 2. During the *blocked period*, perform similarity search over **FI**

Step 3. Scan all leaf nodes of the B^+-tree **FI** and batch update aggregates in **HI** when the actual data arrive (note that, there is a pointer *ptr* pointing to the corresponding entry in **HI**)

Three Steps of the Probabilistic Approach

1(a). At the current timestamp, for each stream time series, predict *future subsequences* with the extended hash index **HI**

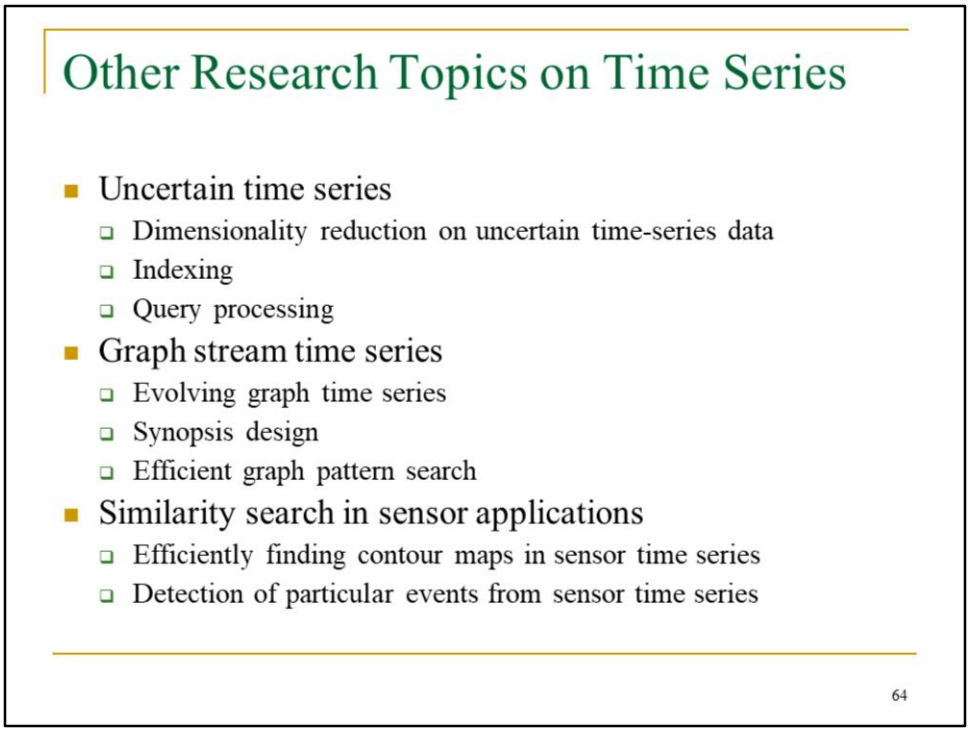
1(b). Insert the predicted strings of *future subsequences* into a B^+-tree **FI**

2. During the *blocked period*, perform similarity search over **FI**

3. Batch update aggregates in **HI** when the actual data arrive

63

Here is detailed pseudo code of the 3 steps mentioned in the previous slide.



There are some other research topics for time series, such as uncertain time series, graph stream time series, similarity search in real applications of time series (e.g., sensor data monitoring).

Motivation of Uncertain Time-Series Databases

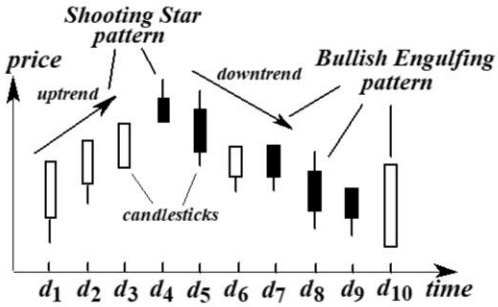
- Previous works on the *similarity search* over time-series databases are usually based on the assumption that the search procedure is performed on ideally *clean* data
- In reality, however, application data are often imprecise and uncertain
  - Sensor data collected from different sites may be distorted for various reasons such as the packet loss, environmental factors, or even systematic errors of devices themselves

65

Previous works on the similarity search over time-series databases usually assume that the queries are conducted over clean data. In reality, however, time-series data such as sensory data, trajectories, and so on are often imprecise and uncertain.

Motivation of Uncertain Time-Series Databases (cont'd)

- Candlestick Model

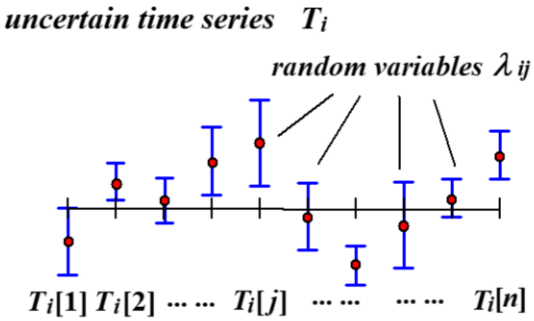


66

For stock price data, some data model like candlestick model inherently model the stock price at each day as an uncertain interval.

Data Model for Uncertain Time Series

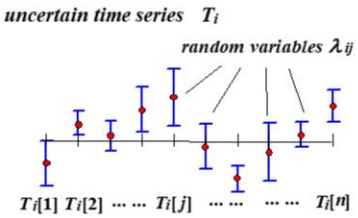
- Uncertain time-series databases



67

Formally, we can model uncertain time series as an ordered sequence of variables, where each variable is represented by an uncertain interval.

Problem Definition



- Uncertain Range Query
  - Uncertain time series data, T_i
  - Query time series, Q
  - A distance threshold r
  - A probabilistic threshold $\alpha \in (0, 1]$
  - To find uncertain time series T_i such that:
    - $Pr\{dist(Q, T_i) \leq r\} \geq \alpha$
- Challenges
  - High query cost involving the probability computation
  - High dimensional uncertain data (dimensionality curse)

68

Range queries can be issued over uncertain time series to retrieve those data series T_i from the databases that are similar to the given query series Q (i.e., within r-distance) with high confidence (i.e., with probability greater than or equal to \alpha).

The challenges of this problem is mainly in the query efficiency which involves the costly probability computation. Moreover, due to high dimensionality of uncertain time series, it is not trivial how to conduct the reduction over uncertain data (instead of certain data).

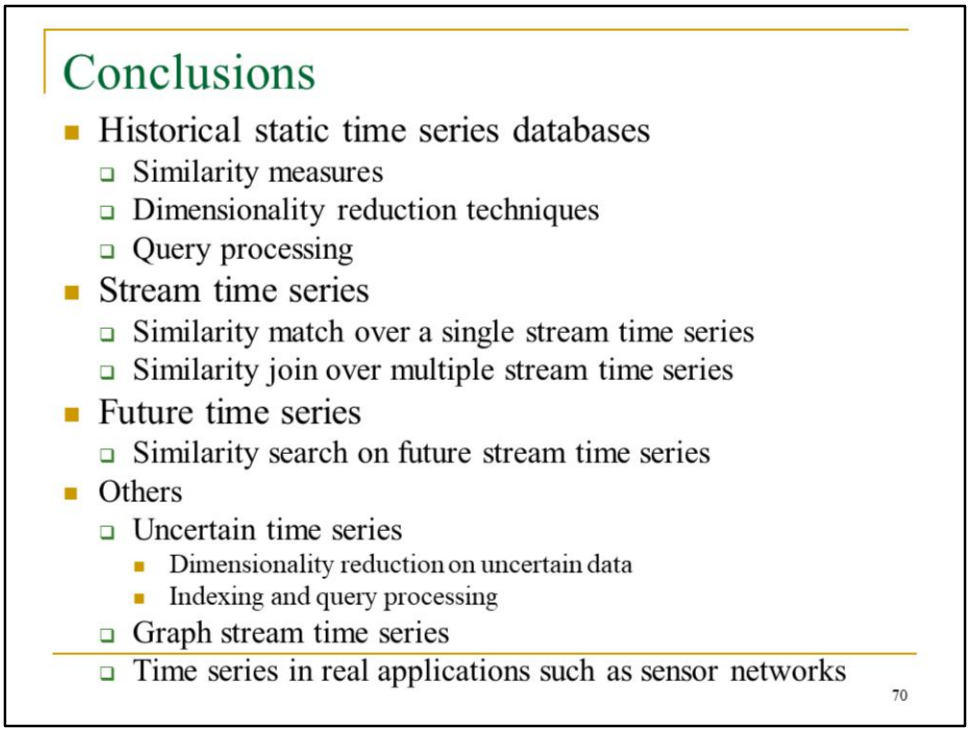
Pruning Method

- Pruning Heuristics
  - Given an *uncertain time series* T_i and a query series Q, let $LB(Q, T_i)$ be a *lower bound distance* between Q and T_i
    - $Pr\{LB(Q, T_i) \le r\} \ge Pr\{dist(Q, T_i) \le r\}$
  - Thus, if $Pr\{LB(Q, T_i) \le r\} < \alpha$, then $Pr\{dist(Q, T_i) \le r\} < \alpha$, indicating that T can be safely pruned
- How to obtain the probability upper bound $Pr\{LB(Q, T_i) \le r\}$?
  - Dimensionality reduction on uncertain data

69

In order to improve the query efficiency over uncertain time series, we can propose some pruning heuristics. Intuitively, we can derive an upper bound of the probability that two series T_i and Q match with each other. If this probability upper bound is less than the threshold \alpha, then T_i can be safely pruned (since the matching probability is low).

The remaining issue is how to derive this probability upper bound, and conduct the dimensionality reduction on uncertain data, which is an open problem for you. If you are interested in it, you can try to solve the problem by yourself.



In summary, in this chapter, we talked about one specific type of big application data, time series. We cover the research topics such as historical static time series, stream time series, and the predicted future time series. We also discuss some potential research topics for time series, including uncertain time series.

'In your project, you may also consider query processing over time series, or other application data such as biological data, big spatial data, big graph data (e.g., social networks), and so on.