Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping

T. Rakthanmanon, et. Al

KDD'12

Introduction

- Time Series Data Mining

  $T = [(p_1, t_1), (p_2, t_2), (p_3, t_3) \ldots (p_i, t_i), \ldots (p_n, t_n)]$

  $t_1 < t_2 < \ldots < t_i < \ldots < t_n$ where, p_i is a data point in d-dimensional space and each t_i is a time stamp at which p_i occurs

  - Two key aspects of achieving effective and efficient Time-Series data mining process:
    - Representation Methods – e.g Discrete Fourier Transform, Single Value Decomposition, etc
    - Similarity Measures – e.g Euclidean Distance, Dynamic Time Warping, Longest Common Subsequence

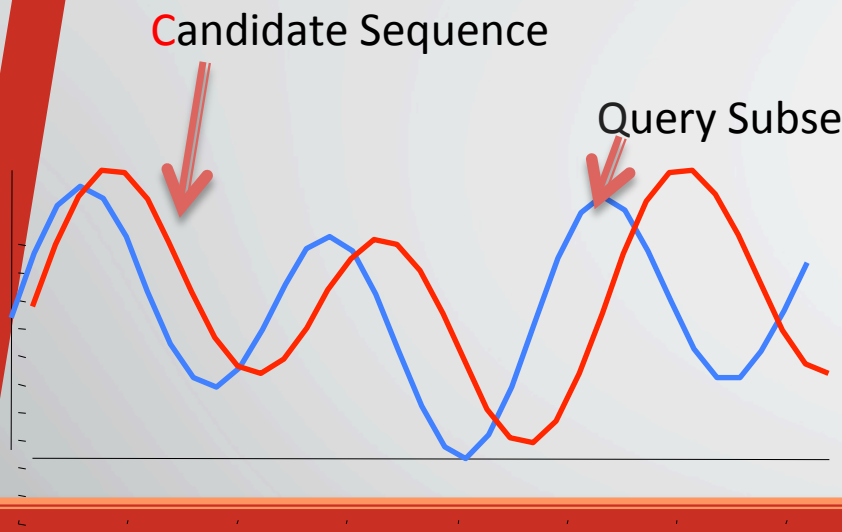
For a detailed comparison study please refer: H. Ding et. al, Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. VLDB '08,

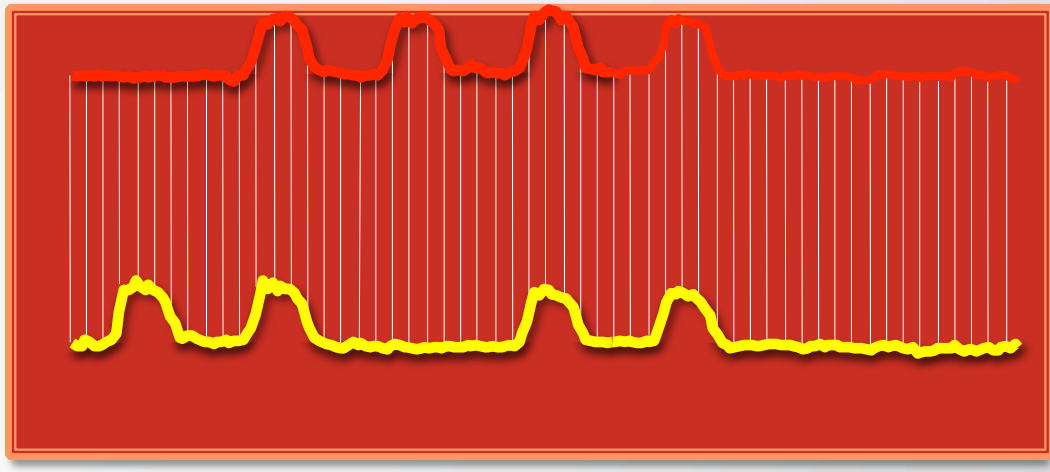
Similarity Measures – Euclidean Distance

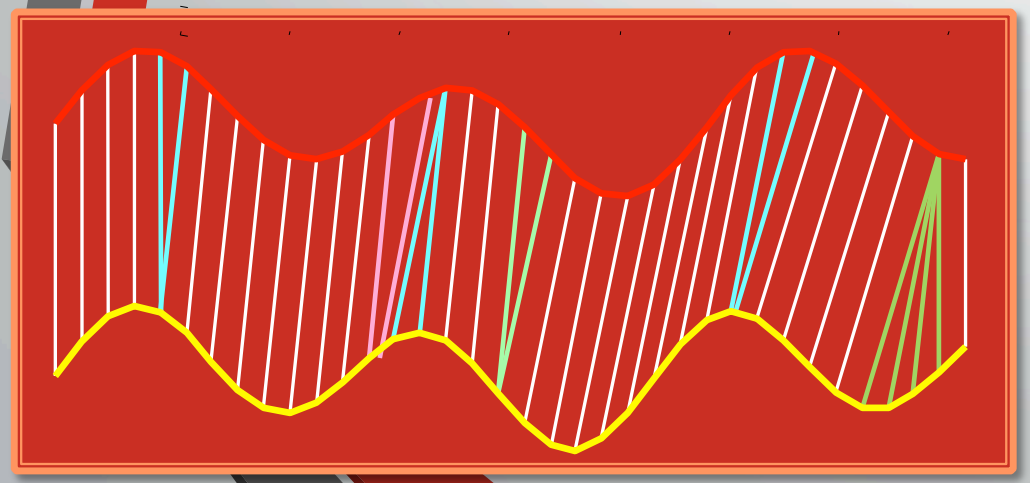
- Linear and easy to compute
- Parameter-free
- But, very sensitive to noise, misalignments in time (out-of-phase time series such as “spoken speech recognition”
- Solution: Dynamic Time Warping (DTW)
  - Berndt & Clifford: Using dynamic time warping to find patterns in time series. KDD Workshop, 1994
  - Also, Parameter-free
  - Amortized cost is linear
  - Suitable for long query sequences

TIME WARPING

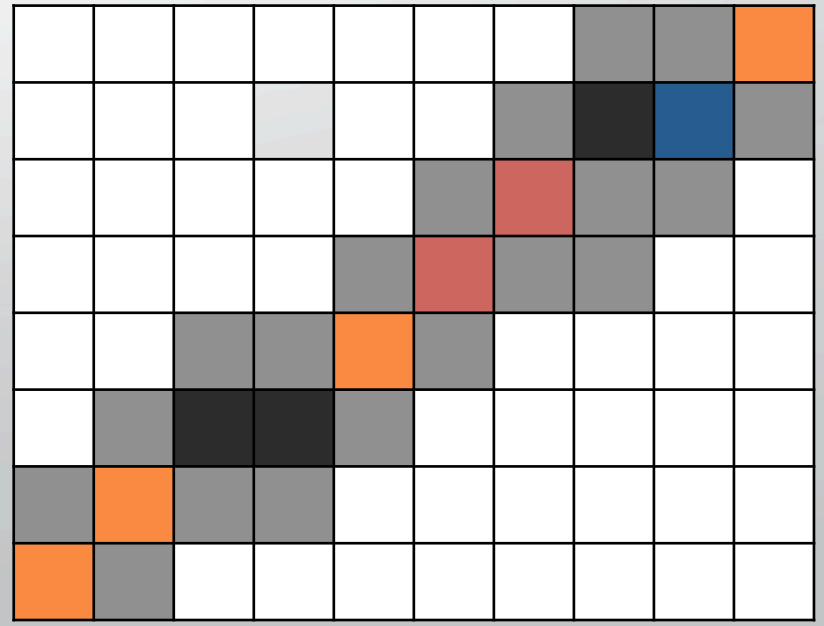
One to one mapping in Euclidean
Distance measure







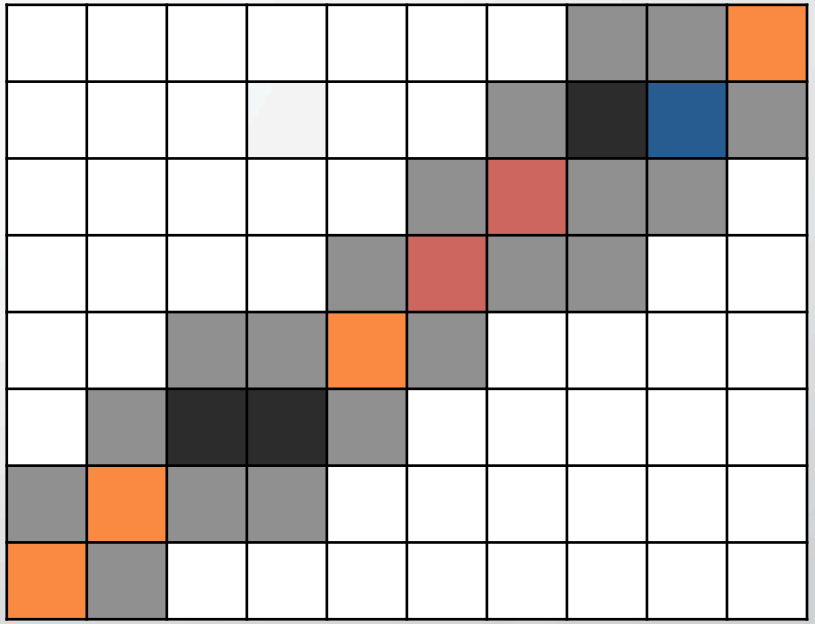
Euclidean Distance



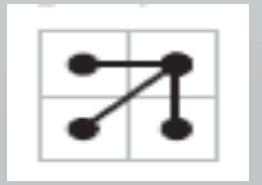
Warping Path

Dynamic Time Warping

- Allows time-series to be compressed and stretched. Hence, the name “time warping”
- Basic Observations:
  - Monotonicity – (i, j) indexes either stay the same or increase, but never decrease
  - Continuity - Path always advances 1 step, and doesn’t include jumps
  - Boundary condition – each sequence has a start and end
  - Adjustment/Warping Window or “Reach” – A good path never wanders too much from the diagonal
  - Slope Constraint – Should not be too steep or too shallow
- These observations lead us to optimizations as well



$$\gamma(i,j) = d(q_i,c_j) + \min\{ \gamma(i\text{-}1,j\text{-}1) , \gamma(i\text{-}1,j) , \gamma(i,j\text{-}1) \}$$



Assumptions #1

- Both the dataset and query subsequence must be *normalized*.

Un-normalized and 1NN classifier error-rate 0.326

Z-normalized and DTW-1NN classifier error-rate 0.087

Un-normalized and 1NN classifier error-rate 0.193

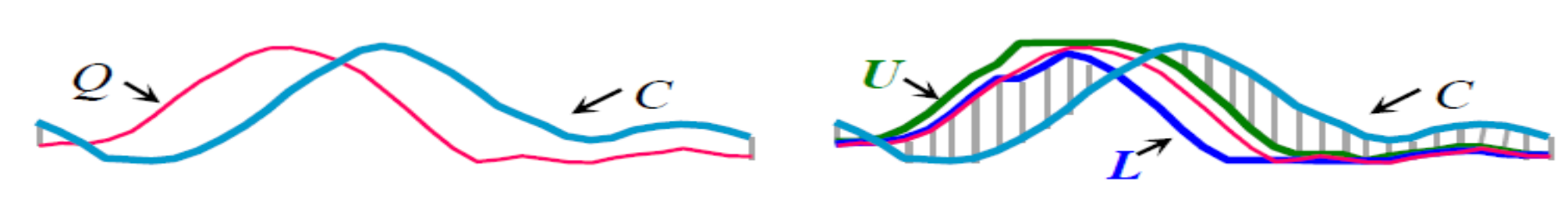
Other assumptions

2. DTW is best measure possible,
   - and that's the one we should optimize
3. Arbitrary Query lengths cannot be indexed
   - Due to High growth of time series data, and Indexes roughly occupy close 1/10 the size of data hence posing "Space Constraints"
   - Indexes need to build for each supported query lengths
4. Waiting time is not much of a concern

Known optimizations

1. Omit the the square root calculation
   - Even then both ED and DTW will be monotonic, making calculations faster
2. Use multi-cores – Parallelization

3. Lower Bounding
   - Prune-off unpromising candidate subsequences

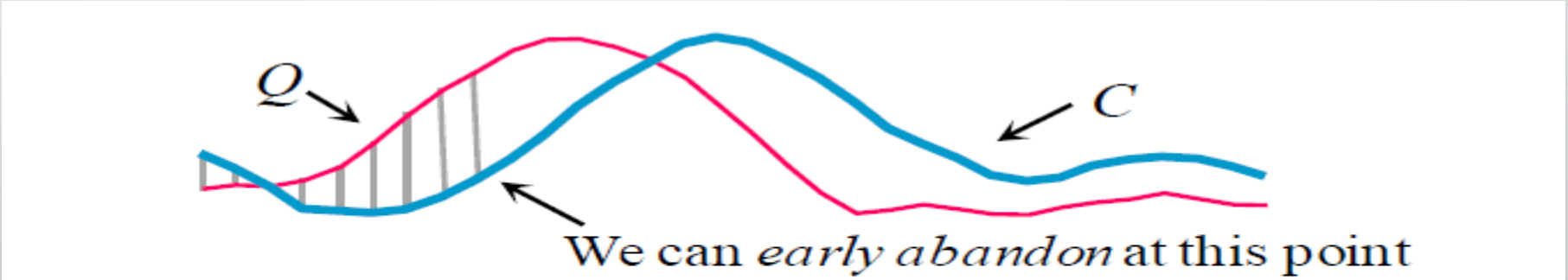


LB_Kim: Using First & Last Pair of points in Candidate and Query subsequence as lower bound. O(1)

LB_Keogh: Using Euclidean distance between Candidate sequence and closer of {U,L} as lower bound . O(n)

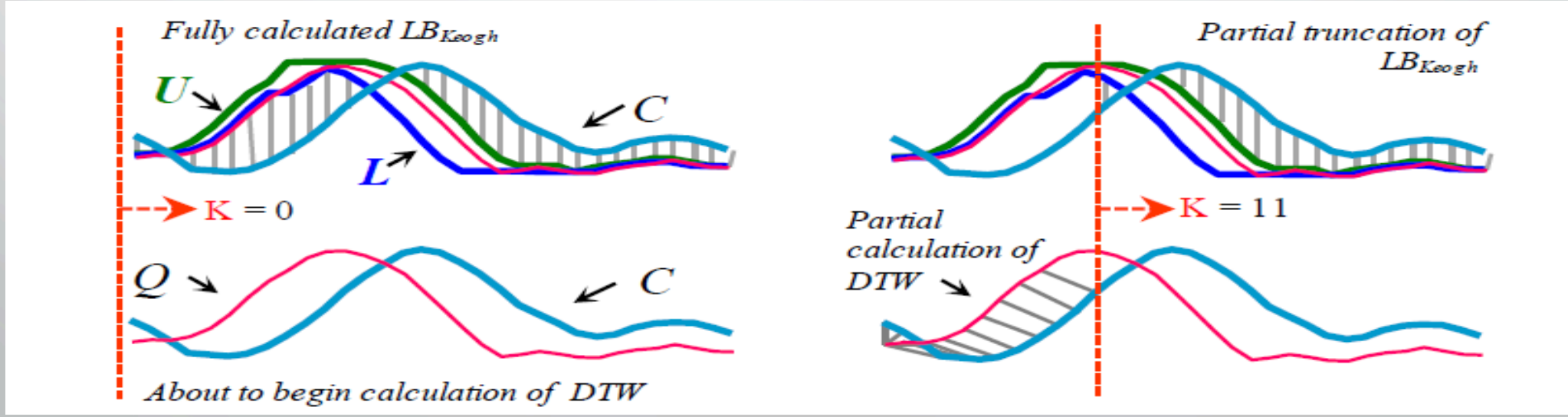
Known Optimizations (contd.)

4. Early Abandoning of ED and LB_keogh
   - During computation of Similarity measure if current sum of squared differences between each pair of data points is greater than “best-so-far” then we abandon!



Known Optimizations (contd.)

5. Early Abandoning of DTW
   - Do calculation of DTW along with LB computation
   - Stop and Abandon if exceeds “best-so-far”



Novel Optimizations

1. Early Abandoning Z-Normalization
   - Optimize the normalization step
   - Incrementally compute Z-normalization and also compute Euclidean Distance by maintaining 2 running sums of long candidate time series which have a lag of exactly ‘m’ values

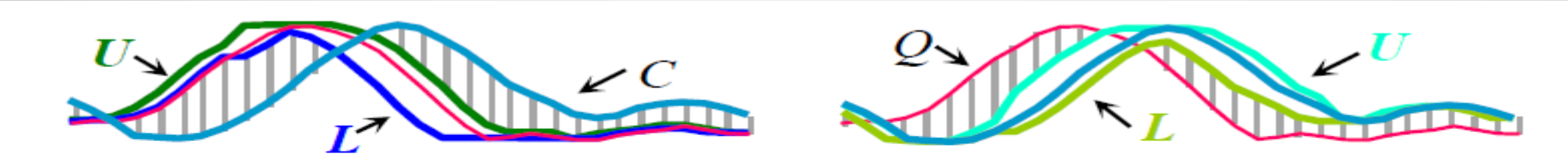
Algorithm	Similarity Search
Procedure	[*nn*] = *SimilaritySearch* (T,*Q*)
1	*best-so-far* ← ∞, *count*← 0
2	*Q* ← *z-normalize*(*Q*)
3	**while** !*next*(T)
4	*i*← *mod*(*count*,*m*)
5	X[*i*] ←*next*(T)
6	*ex*← *ex*+X[*i*], *ex2*← *ex2*+X[*i*]^2
7	**if** *count* ≥ *m*-1
8	*μ*← *ex*/*m*, σ← *sqrt*(*ex2*/*m* - *μ* ^{2})
9	*j*← 0, *dist*← 0
10	**while** *j* < *m* **and** *dist* < *best-so-far*
11	*dist*← *dist* + (*Q*[*j*]-(X[*mod*(*i*+1+*j*,*m*)]-*μ*)/σ)^2
12	*j*← *j*+1
13	**if** *dist* < *best-so-far*
14	*best-so-far* ←*dist*, *nn*← *count*
15	*ex*← *ex*-X[*mod*(*i*+1,*m*)]
16	*ex2*← *ex2*-X[*mod*(*i*+1,*m*)]^2
17	*count* ← *count*+1

Novel Optimizations

2. Re-ordering early abandoning
   - Universal optimal order is to sort based on absolute values of the Z-*normalized Q*
   - Intuition: Is that sections farthest from the mean, will have the largest contribution to the distance measure

Novel Optimizations

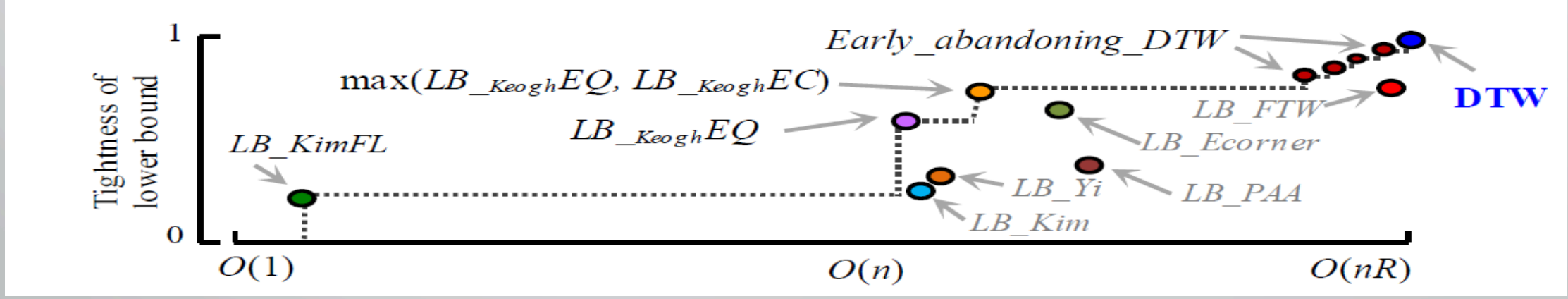
3. Reversing the Query/Data Role in LB_keogh
   - Build LB envelope around the candidate sequence as well, in "*just-in-time*" fashion *only if* other methods fail to prune off



Novel Optimization

4. Cascading Lower Bounds

- Use a bunch of LB methods (based on their tightness of lower bound and computation complexity) in a "*cascade*" i.e one after the other



Experimental Results

- Setup:
  - Naive
    - Each subsequence is Z-normalized from scratch. The full ED or the DTW is used at each step.
  - State-of-the-art (SOTA)
    - Each sequence is Z-normalized from scratch, early abandoning is used, and the LB_Keogh lower bound is used for DTW
  - UCR
    - Use all of the speedup techniques in this paper

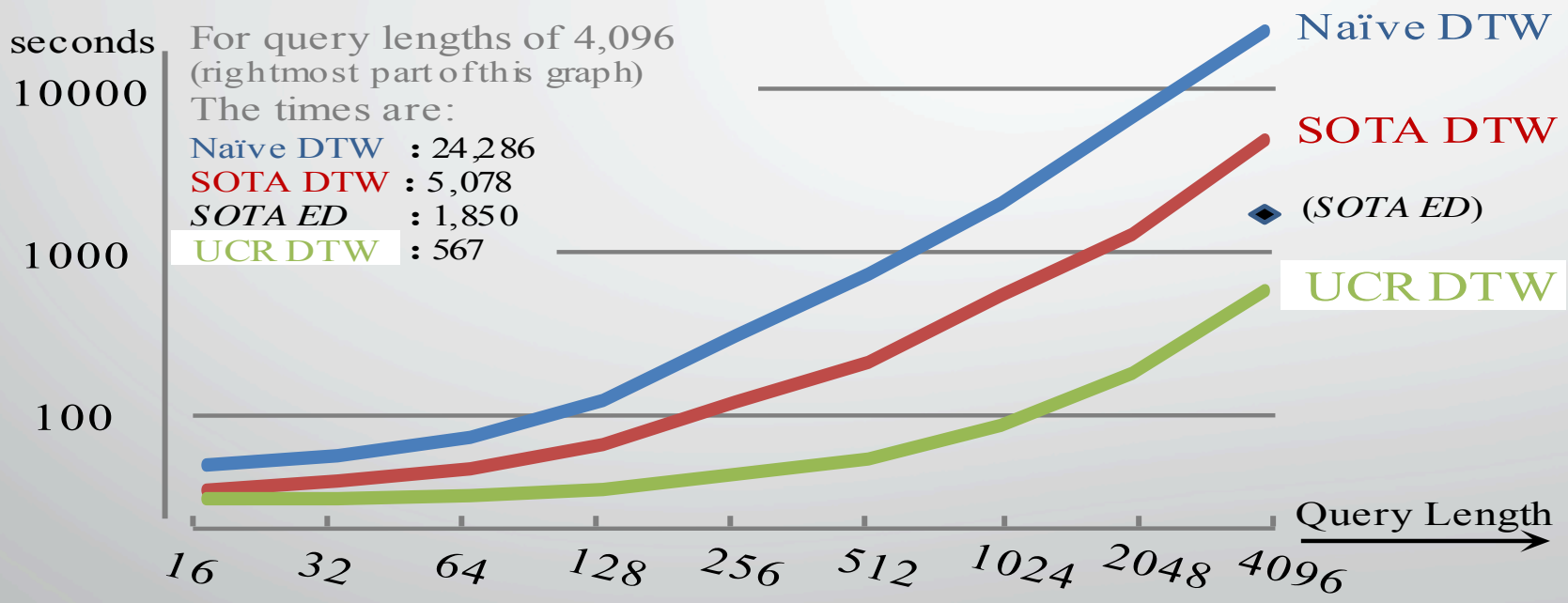
Baseline tests on Random Walk

- Random walks model financial data very well and are often used to test similarity search schemes
- Easily reproducible
- Using a bunch of high quality random number generator (that has a period longer than the longest dataset in consideration).

- Using short queries |Q| = 128

	Million (*Seconds*)	Billion (*Minutes*)	Trillion (*Hours*)
UCR-ED	0.034	0.22	3.16
SOTA-ED	0.243	2.40	39.80
UCR-DTW	0.159	1.83	34.09
SOTA-DTW	2.447	38.14	472.80

- Using long queries |Q| = 4096



Real-world Data Examples

- EEG
  - From a single patient gathered 0.3 trillion data points
  - Query made up of prototypical epileptic spike (from averaging spikes from other patient)
    - |Q| = 7000

Note that only ED is considered here because DTW may produce false positives caused by eye blinks		UCR-ED	SOTA-ED
	EEG	3.4 hours	494.3 hours

- DNA Sequences
  - Converted a section of Human Chromosome 2 in to time series
  - |C | =~ $2 * 10^9$ base-pairs in length
  - |Q| = 72500

	Naïve	SOTA	UCR
Time Taken	38.7 days	34.6 days	14.6 hours

Conclusion

- Paper, focuses on fast sequential search for DTW, is faster than all known methods
- After including all optimizations speed-up is 2 to 164 times faster than the state-of-the-art, depending on the query/data.
- Applicable to several real-time problems such Gesture recognitions, tasks which were prohibitive with DTW

Adaptive Selectivity Estimation Using Query Feedback

C. M. Chen and N. Roussopoulos

Proceedings of the
1994 ACM-SIGMOD International Conference on Management of Data, 1994.

Introduction

- Query optimizers in RDBMS use several measures to choose the right plan for query execution
- One of them is "*Selectivity*"

  Selectivity = cardinality / #records * 100

  cardinality – number of unique values that can appear in column of table

  - Example: Use of Indexes based on selectivity
    - For Lower the selectivity do not use indexes
  - Attribute Value Distribution: Refers to values that columns in data-tables can take
    - Useful to know before hand when executing complex queries

Selectivity

- Different estimation procedures:
  - Non-parameteric: dividing “attribute value distribution” into histogram like data-structures and using special ML algorithms to estimate selectivity measure
    - High storage costs
  - Parametric: Using statistical tech. to approx. actual distribution using some mathematical distributions function
    - But assumes database value distribution conforms to those distributions thus must know a priori
  - Curve-fitting: Using a polynomial function and then using Least-Squares-Sum estimate to approx. atttributte value distribution
    - But, if degree of polynomial is high it leads to oscillation error and rounding off errors
  - Sampling: Collect samples of tuples from relations in RDBMS to collect statistics
    - Sampling must be re-done after enough updates to database

Main Recipe

- Approximate “Attribute Value Distribution” using query feedback to better estimate “selectivity”
- By using feedback from each query we avoid over-head of statistics collection
- With each query run on the database, we regress the distribution
- Such that it adapts better to query localities and database updates
- Inspiration: Improved Data Knowledge Acquisition
  - since the statistics gathering tasks are prohibitive due to large database sizes

Strategy: Recursive and Weighted Least-Square-Error

- Given a sequence of queries q_o, q_1, q_2, … and distribution *f* as a sequence f_o, f_1, f_2, ….
- Where f_{i-1} is used to estimate the selectivity of query q_i ,
- After q_i is executed, f_{i-1} is further adjusted into f_i using the feedback T_i that contains the actual selectivity s_i

- Estimated Selectivity:

$$\hat{s} = \int_l^{h+1} f(x)dx = F(h+1) - F(l) = \sum_{j=0}^{n} a_j[\Phi_j(h+1) - \Phi_j(l)].$$

$$\sum_{i=1}^{m}(\hat{s}_i - s_i)^2 = \sum_{i=1}^{m}(\sum_{j=0}^{n} a_j[\Phi_j(h_i+1) - \Phi_j(l_i)] - s_i)^2.$$

Applying Transformations

$$A^* = (X^t X)^{-1} X^t Y.$$

Possible Drawbacks: Size of X and Y increases proportionally to the number of query feedbacks ‘m’. Hence needs to be re-computed every time

$$P = X^t X = \sum_{i=1}^{m} X_i^t X_i, \quad N = \sum_{i=1}^{m} X_i^t s_i,$$

$$A_i^* = P_i^{-1} N_i, \text{ for } i = n+1, n+2, \ldots \text{ where}$$

$$P_i = P_{i-1} + X_i^t X_i, \quad N_i = N_{i-1} + X_i^t s_i, \text{ for } i = 1, 2, \ldots,$$

Applying above transformation, dimension of both P and N are now independent of ‘m’

Finally: Recursive LSE

- Another transformation: to get rid of expensive transformation P_i^{-1}
- Using a concept of Gain Matrix

$$
\begin{aligned}
A_i^* &= A_{i-1}^* - G_i X_i^t (X_i A_{i-1}^* - s_i), \\
G_i &= G_{i-1} - G_{i-1} X_i^t (1 + X_i G_{i-1} X_i^t)^{-1} X_i G_{i-1},
\end{aligned}
$$

- Choosing appropriate values of A_o and G_o, so that A^* converges efficiently is important

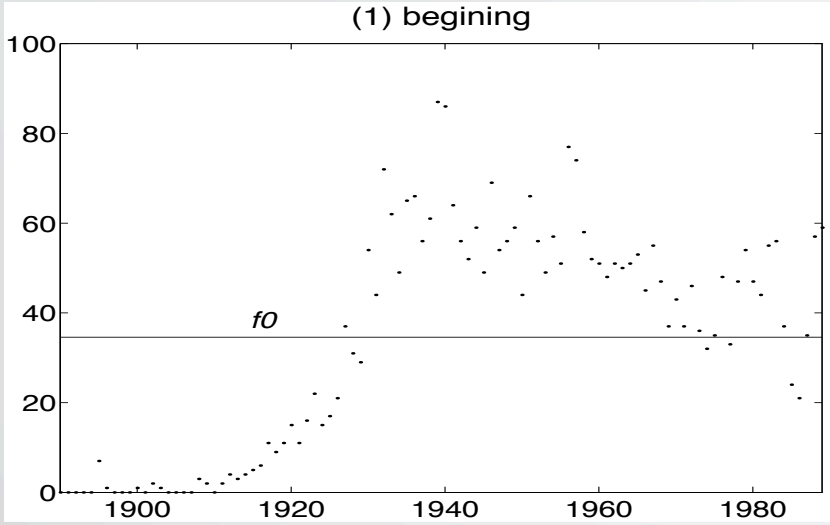
Weighted version

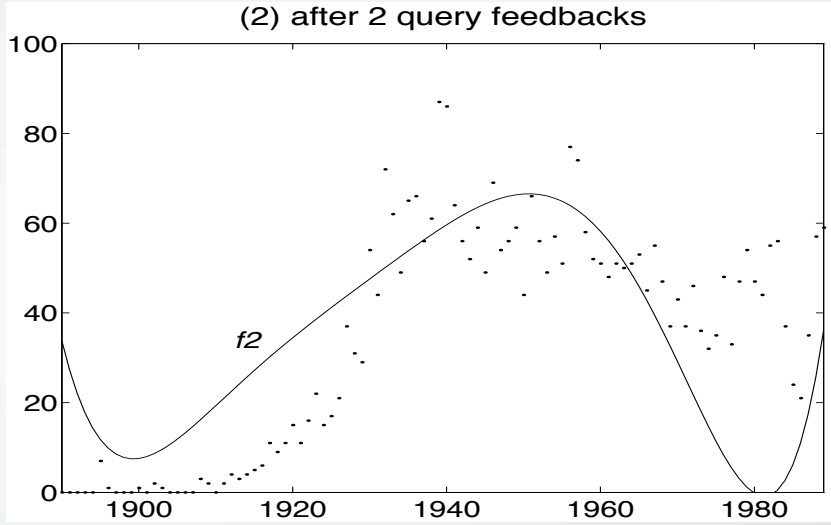
- Accommodating Update Adaptiveness: using Weighted LSE
  - By associating different weights with the query feedbacks, such that outdated feedback are suppressed
  - β_i - importance weight
  - α_i - fading weight to estimation errors of all previous queries

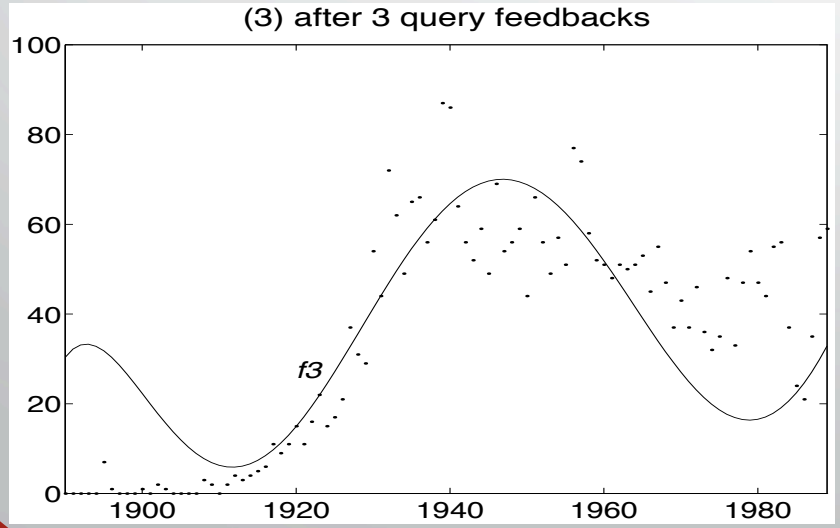
$$\sum_{i=1}^{m}[(\prod_{j=i+1}^{m}\alpha_j)\cdot\beta_i\cdot(\hat{s}_i - s_i)]^2.$$

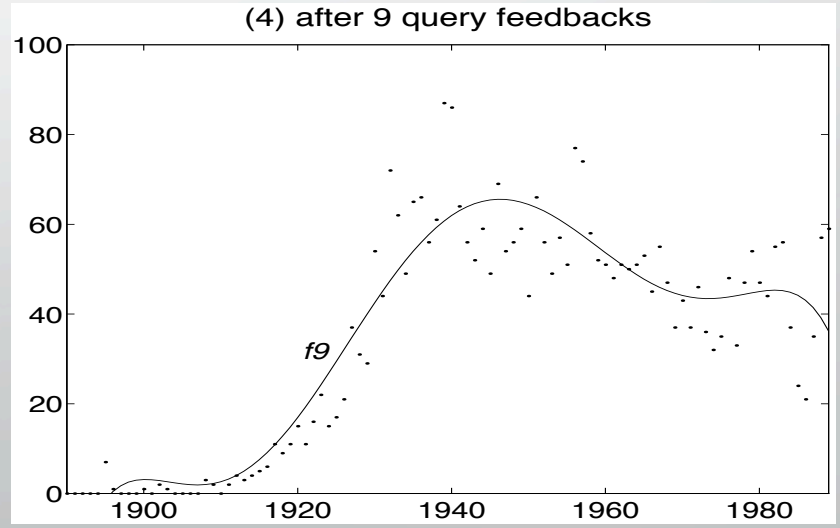
$$\begin{aligned} A_i^* &= A_{i-1}^* - \beta_i^2 G_i X_i^t (X_i A_{i-1}^* - s_i), \\ G_i &= (\frac{1}{\alpha_i})^2 G_{i-1} - (\frac{\beta_i}{\alpha_i})^2 G_{i-1} X_i^t (\alpha_i^2 + \beta_i^2 X_i G_{i-1} X_i^t)^{-1} X_i G_{i-1}, \end{aligned}$$

Experimental Result: Adaptation Dynamics









Online Data Mining for Co-Evolving Time Sequences

B.K. Yi , N.D Sidiropoulos, T. Johnson , H. V. Jagadish , C. Faloutsos , A. Biliris

Carnegie mellon university

Outilne

- Introduction
- Challenges
- MUSCLES and its capabilities
- Applications
- Methodology
- Experiments
- conclusion

Introduction

- Data in Many applications: are multiple sequences evolve over time
- Sequences are not independent and frequently have high correlation
- Much useful information lost if each sequence is analyzed individually
  - Example:
  - Network traffic data from different network elements
  - Patient data varying over time
  - …

- Goal: study the entire set of sequences
- Challenges:
  - Number of sequences can be very large
  - Being Online: Results of analysis being available immediately
    - Require: capability to repeat analysis over next element(or batch of elements)
  - Analytical technique should have low incremental computational complexity

MUSCLES (MUlti-SequenCe LEast Squares)

- A technique for analysis multiple time series:
- Can estimate delayed/ missed / corrupted values
- Can recognize the outlier
- Can extract the correlation of the sequence with other sequences
- It is online and fast, operating in time independent of number of samples(N) or number of past time-ticks
- It scale up well with the number of time sequences, k.
- It is capable of adapting quickly to the change of trends

Applications

- Network Management: Time sequences of measurements (Number of packets lost, sent, received for a collection of nodes)
  - Find missing/ delayed values, find outliers as trouble or spam, forecast and alarm for troubles
- Web and intranet management: Time sequence of number of hits per minute for each site
  - Find correlation between access patterns
  - Forecast future requests(prefetching and caching)
  - Detect outlier as intruders

MUSCLES (MUlti-SequenCe LEast Squares)

- Problem1: Delayed Sequence:
- Input: k time sequences: $s_1,\ldots, s_k$ is updated at every time-tick
- One of them(s_1) be consistently late
- Make the best guess for $s1[t]$

sequence time	s_1 packets-sent	s_2 packets-lost	s_3 packets-corrupted	s_4 packets-repeated
1	50	20	10	3
2	55	20	10	10
⋮	⋮	⋮	⋮	⋮
$N-1$	73	25	18	12
N	??	25	18	18

Symbol	Definition
MUSCLES	Multi-Sequence Least Squares
v	number of independent variables in multi-variate regression
k	number of co-evolving sequences
y	the *dependent* variable, that we try to estimate
$\hat{y}$	estimate of the dependent variable y
$\mathbf{y}$	the column vector with all samples of the dependent variable x
$y[j]$	the j-th sample of the dependent variable y
x_i	the i-the independent variable
$x_i[j]$	the j-th sample of the variable x_i
$\mathbf{x}_i$	the column vector with all the samples of the variable x_i
$\mathbf{x}[j]$	the row vector with j-th samples of variables x_i
w	span of tracking window
b	count of 'best' ind. variables, used for Sel. MUSCLES
λ	forgetting factor (1, when we don't forget the past)

All samples of one variable x1

$X = \blacksquare x1 \& x2 \ldots \& xn @ \blacksquare 102023 \& \blacksquare 112325 \ldots$

j^{th} samples of each sequence

Proposed Solution: Multi variant linear Regression

- Two source of information:
- The past samples of the given time sequence s1: $s_1[t-1]$, $s_1[t-2]$, …
- The past and present samples of the other time sequences: s_2, s_3, … , s_v
- Estimate $s_1[t]$ as linear combination of other values in a window of size w

$$
\begin{aligned}
\widehat{s_1}[t] = \quad & a_{1,1}s_1[t-1] + \ldots + a_{1,w}s_1[t-w] + \\
a_{2,0}s_2[t] + \quad & a_{2,1}s_2[t-1] + \ldots + a_{2,w}s_2[t-w] + \\
& \ldots \\
a_{k,0}s_k[t] + \quad & a_{k,1}s_k[t-1] + \ldots + a_{k,w}s_k[t-w],
\end{aligned}
$$

Multi variant linear Regression: minimize least square error

- Find regression coefficients $a_{i,j}$ which minimize the sum $(s1[t] - \hat{s}1[t])^2$ for all t = w+1,…,N
- Optimal regression coefficients are:

$$\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y})$$

- y is desired values($s_1[t]$)

All samples values of independent variable x1

$$X = \begin{matrix} x1 & x2 & \dots & xn \\ 102023 & 112325 & \dots & \end{matrix}$$

Observation of each sequence at time t

Time Complexity and memory usage

- Optimal regression coefficients are: $\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y})$
- X is $N \times v$ matrix where:
  - v= number of independent variables = $k \times (w+1) - 1$
  - N = number of samples for each variable
- Memory usage to store X : $O(N \times v)$
- Computational cost: $O(v2 \times (N+v))$
- Computation of $(\mathbf{X}^T \times \mathbf{X})^{-1}$ needs quadratic disk I/O operation
- N is not fixed and can grow indefinitely!

All samples values of independent variable x1

$X = \blacksquare x1 \& x2 \ldots \& xn @ \blacksquare 102023 \& \blacksquare 112325 \ldots \&$

Observation of each sequence at time t

Solution: Matrix Inversion Lemma

- Let X_n denote the X with the first n=N samples
- Define G_n as $(X\downarrow n\uparrow T \times X\downarrow n)\uparrow -1$
- x[n] is a row vector of the n^{th} sample values
- G_n can be calculated using G_{n-1}:

$X = \blacksquare x1 \& x2 \ldots \& xn @ \blacksquare 102023 \& \blacksquare 112325 \ldots \&$

X_{n-1}

x[n]

$$\mathbf{G}_n = \mathbf{G}_{n-1} - (1 + \mathbf{x}[n] \times \mathbf{G}_{n-1} \times \mathbf{x}[n]^T)^{-1} \times (\mathbf{G}_{n-1} \times \mathbf{x}[n]^T) \times (\mathbf{x}[n] \times \mathbf{G}_{n-1}), \quad n > 1$$

Advantages of Matrix Inversion Lemma

$$\mathbf{G}_n = \mathbf{G}_{n-1} - (1 + \mathbf{x}[n] \times \mathbf{G}_{n-1} \times \mathbf{x}[n]^T)^{-1} \times (\mathbf{G}_{n-1} \times \mathbf{x}[n]^T) \times (\mathbf{x}[n] \times \mathbf{G}_{n-1}), \quad n > 1$$

- Inside of the inversion is just a scalar and, hence, no matrix inversion is required in the above equation
- G_n($v \times v$ matrix) is much smaller than Xn ($N \times v$ matrix) because $N \gg v$
- We don't need to keep the original matrix Xn explicitly.
- It is more likely that we can keep Gn in main memory: reduce I/O cost
- computational cost for updating regression coefficients a, is $O(v^2)$ for each new sample

Comparision

- A dataset of 100 sequences with 10000 samples for each one, with $\mathbf{a} = (\mathbf{X}^T \times \mathbf{X})^{-1} \times (\mathbf{X}^T \times \mathbf{y})$ takes almost 84 hours to do estimation

- Larger dataset with 100 sequences with 100000 samples for each sequence, with new method takes only one hour to run.

Adaptiveness

- There are some changes in the processes that cause multiple sequences to be correlated
- How to adapt the changes over time?
- The old coefficients are no longer correct: wrong estimation

Exponentially Forgetting MUSCLES

- Modify MUSCLES to forget older samples gracefully
- $0<\lambda<1$: Forgetting factor: determines how fast the effect of older samples fades away
- try to find the optimal regression coefficient vector a to minimize:

$$\min_{\mathbf{a}} \sum_{i=1}^{N} \lambda^{(N-i)} (y[i] - \hat{y}[i])^2$$

$$\mathbf{G}_n = \lambda^{-1}\mathbf{G}_{n-1} - \lambda^{-1}(\lambda + \mathbf{x}[n] \times \mathbf{G}_{n-1} \times \mathbf{x}[n]^T)^{-1} \times (\mathbf{G}_{n-1} \times \mathbf{x}[n]^T) \times (\mathbf{x}[n] \times \mathbf{G}_{n-1}), \quad n > 1.$$

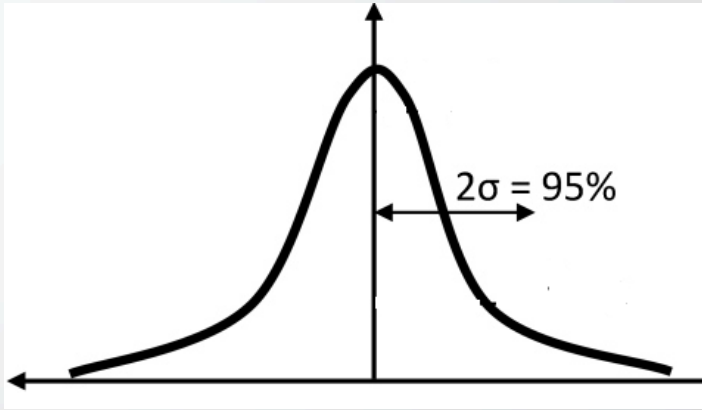
Other usage of MUSCLES

- **Estimate Missing values:** For each sequence, s_i, calculate coefficients of regression based on previous data: estimate missing value immediately

- **Correlation Detection**: A high absolute value for a regression coefficient means that the corresponding variable is highly correlated to the dependent variable

Other usage of MUSCLES

- **On-line outlier detection**: an outlier is a value that is very different from what we expected:
  - if distance of sample value and estimated value is larger than 2σ it detects an outlier

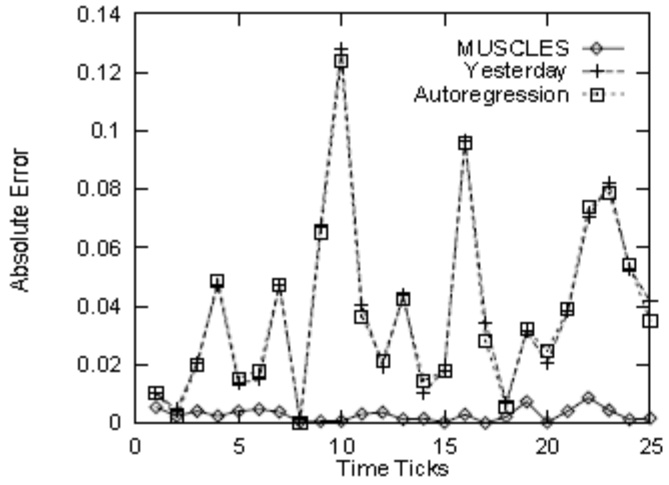


- **Corrupted data and back-casting:** If a value is

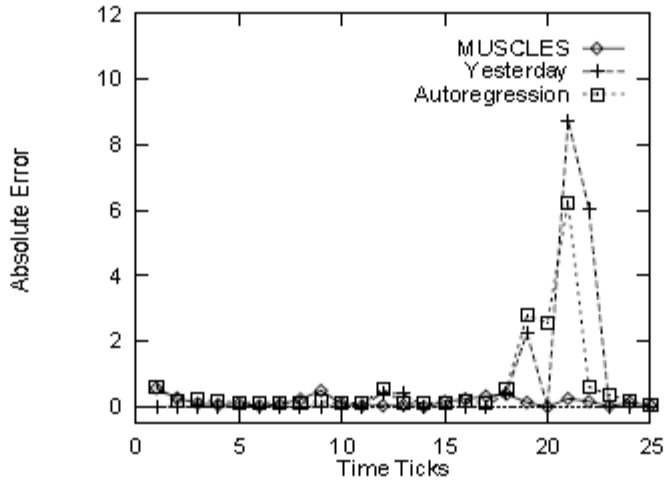
  Corrupted or suspected in our time sequences
- Treat it as “delayed”, and forecast it :
- Estimate past values of the time sequences as a function of the future values, and set up a multi-sequence regression model.

Accuracy

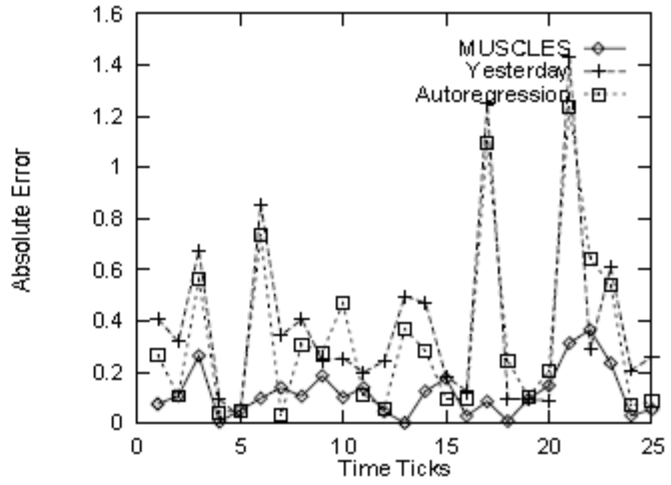
- Compare absolute estimation error of MUSCLES with 2 successful prediction methods: yesterday and single-sequence AR(auto-regressive) analysis



(a) US Dollar (**CURRENCY**)



(b) 10-th modem (**MODEM**)

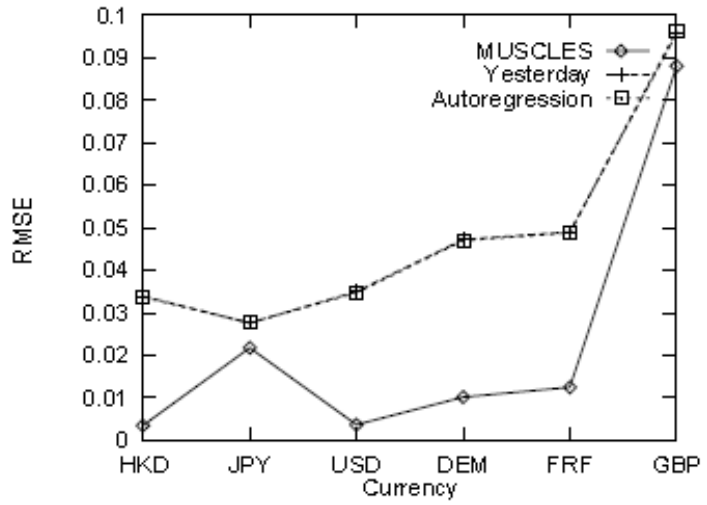


(c) 10-th stream (**INTERNET**)

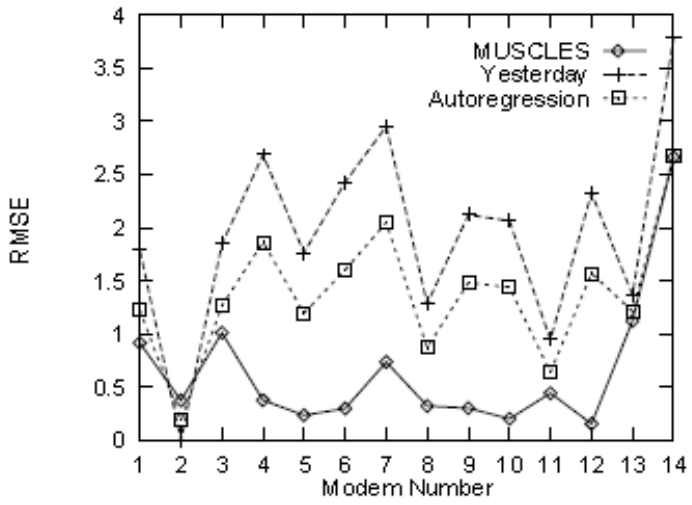
Absolute estimation error as time evolves for the selected sequences.

Accuracy-Cont.

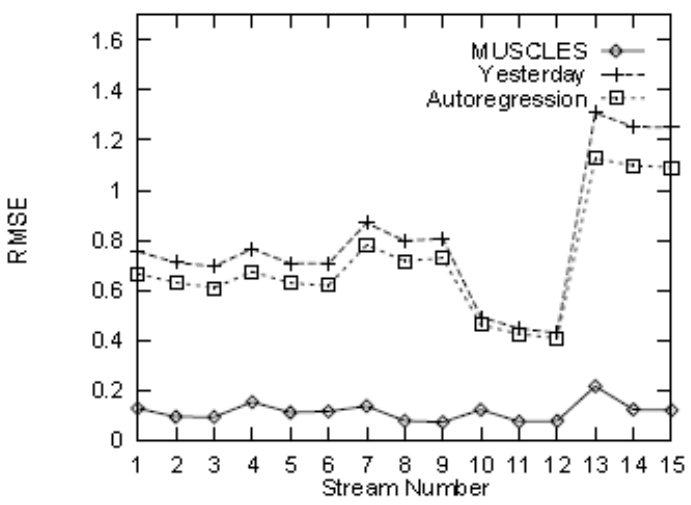
- RMS: Root Mean Square error (mean over time-ticks)



(a) **CURRENCY**



(b) **MODEM**



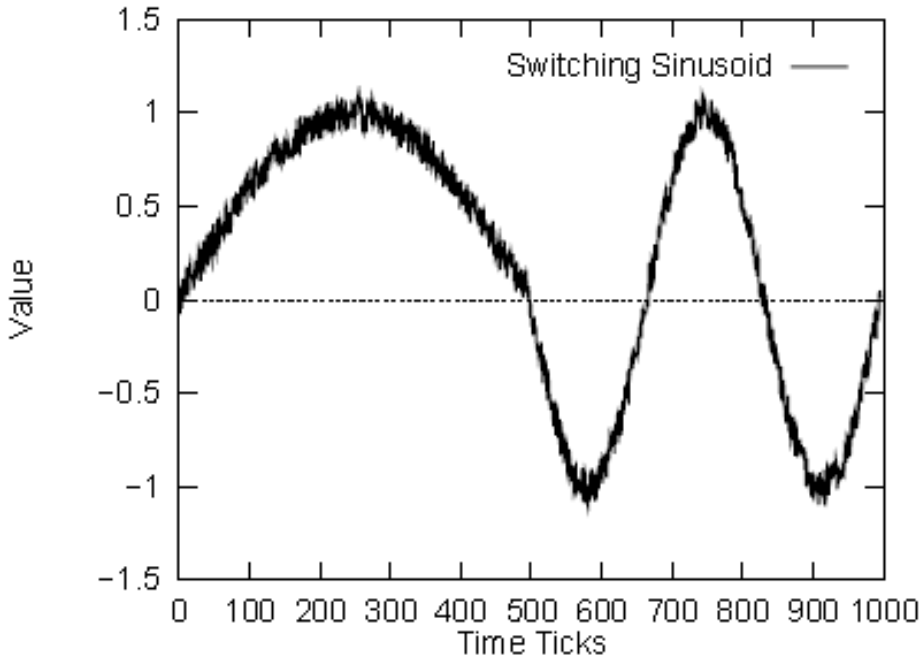
(c) **INTERNET**

RMS error comparisons of several alternatives.

Adapting to change

- Generate 3 sequences:

$$
\begin{aligned}
s_1[t] &= s_2[t] + 0.1 * n[t] && t \leq 500 \\
&= s_3[t] + 0.1 * n'[t] && t > 500 \\
s_2[t] &= \sin(2\pi t/N) \\
s_3[t] &= \sin(2\pi 3t/N)
\end{aligned}
$$



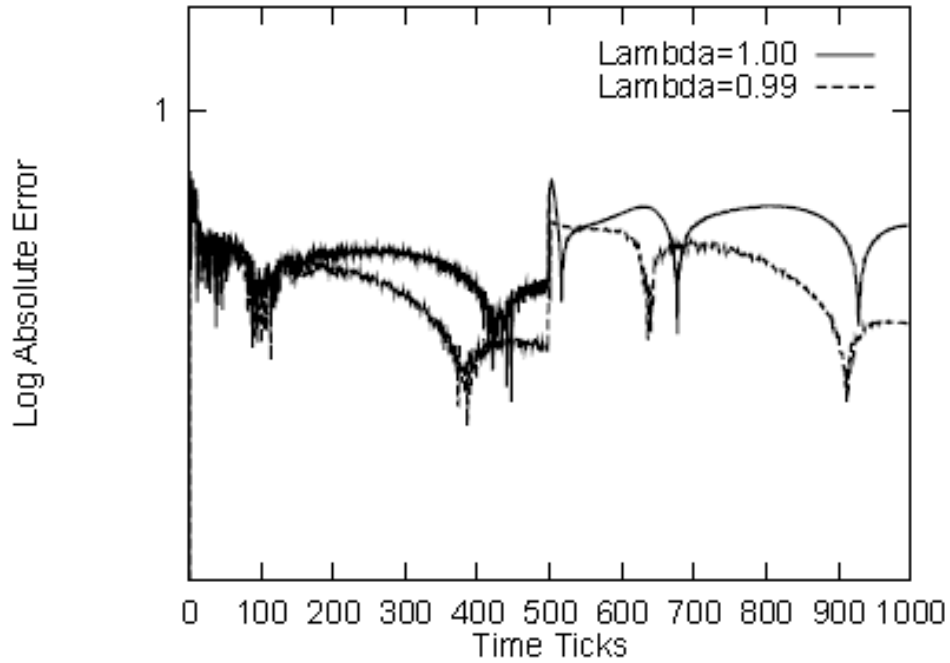
Adapting to change

- Non-forgetting MUSCLES: $\lambda=1$

$$\widehat{s_1}[t] = 0.499 * s_2[t] + 0.499 * s_3[t]$$

- Forgetting MUSCLES: $\lambda=0.99$

$$\widehat{s_1}[t] = 0.0065 * s_2[t] + 0.993 * s_3[t]$$



Prediction errors with $\lambda = 1$ and $\lambda = 0.99$

Scaling-up: Selective MUSCLES

- Too many time sequences: nodes in a network, producing information about their load every minute
- MUSCLES suffer to analyze these data
- Selective MUSCLES: make the estimation of missing value by using small number of selected sequences.
- Do some preprocessing to find a promising subset of sequences

Problem : Subset selection

- Input: v independent variables $x_1, x_2, \ldots, x_v$ and a dependent variable y and N sample each,
- Goal: find the best b(< v) independent variables to minimize the mean-square error for y
- Need a good measure of goodness to select a subset of b variables
  - Use expected estimation error:

$$\mathrm{EEE}(S) = \sum_{i=1}^{N} (y[i] - \widehat{ys}[i])^2$$

  - S is selected subset of variable (until now)
  - $ys[i]$ is the estimation based on S for the i^{th} sample

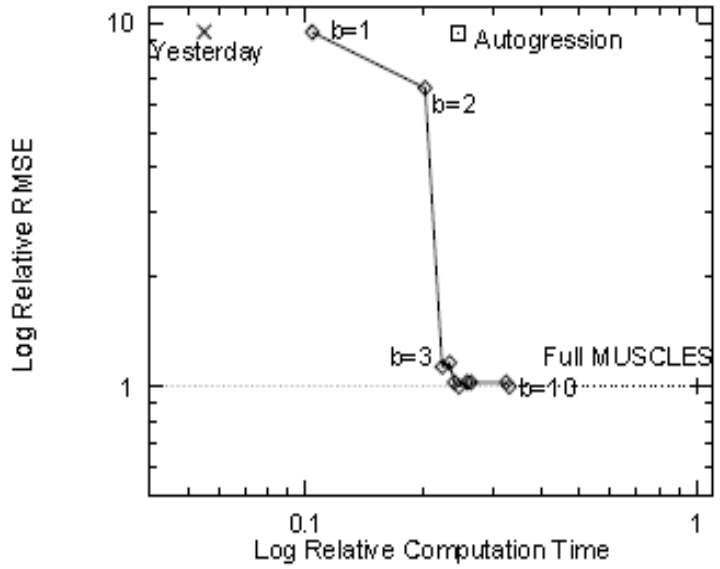
Algorithm to select b variables

```
algorithm Selection
    S := {};                    /* selected var's */
    R := {x_1, ..., x_v};       /* remaining var's */
    while ( S contains less than b variables )
        foreach x in R
            Compute EEE for S ∪ {x};
        pick x with minimum EEE;
        remove x from R and add to S;
    end while
    report variables in S;
end algorithm
```

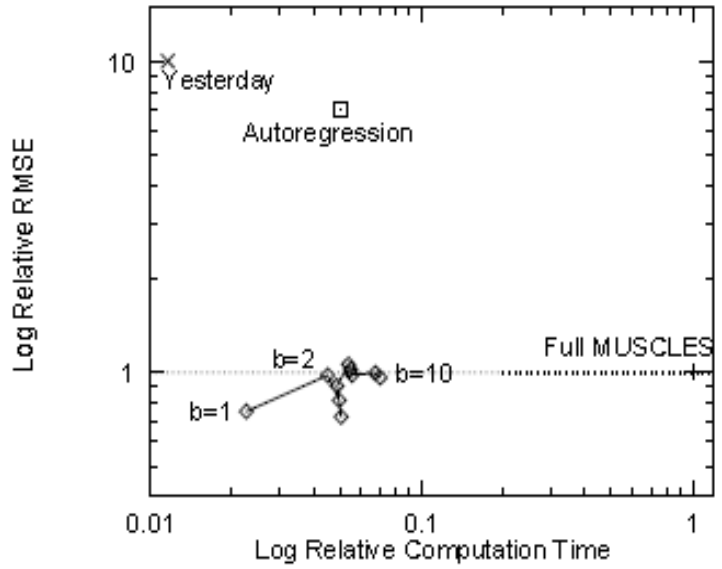
The complexity of algorithm is $O(N \times v \times b2)$

Experiments: speed-accuracy trade off

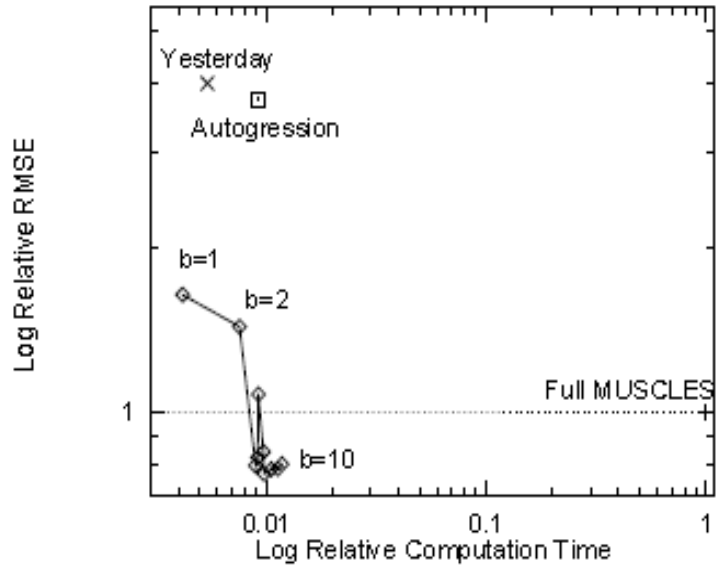
- RMS error VS. Computational time (b=1, … , 10)



(a) US Dollar (**CURRENCY**)



(b) 10-th modem (**MODEM**)



(c) 10-th stream (**INTERNET**)

The relative RMS error vs. relative computation time, for several values of b 'best-picked' independent variables. proposed(◇) with varying b, Full MUSCLES(+), yesterday (×), and auto-regression(□).

- b=3-5 suffice for accurate estimation
- Selective MUSCLES some times have more accuracy than Full-Muscles

Question?