Stream Monitoring under the Time Warping Distance

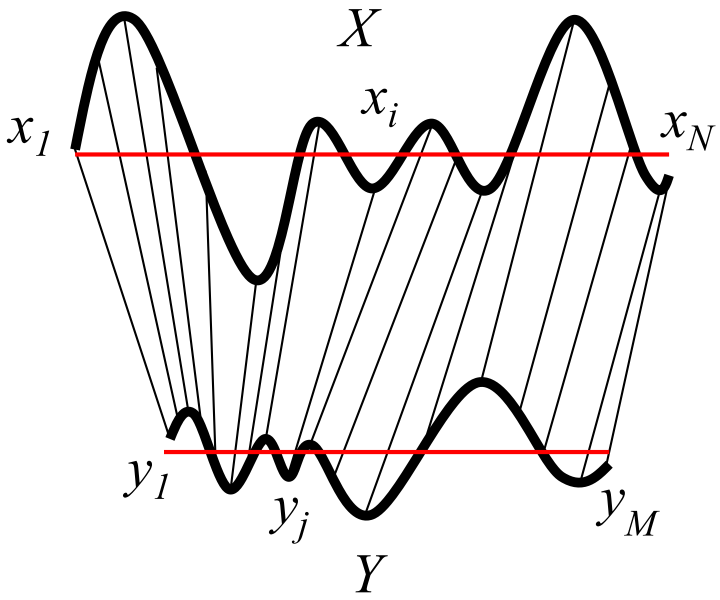
Yasushi Sakurai (NTT Cyber Space Labs)
Christos Faloutsos (Carnegie Mellon Univ.)
Masashi Yamamuro (NTT Cyber Space Labs)

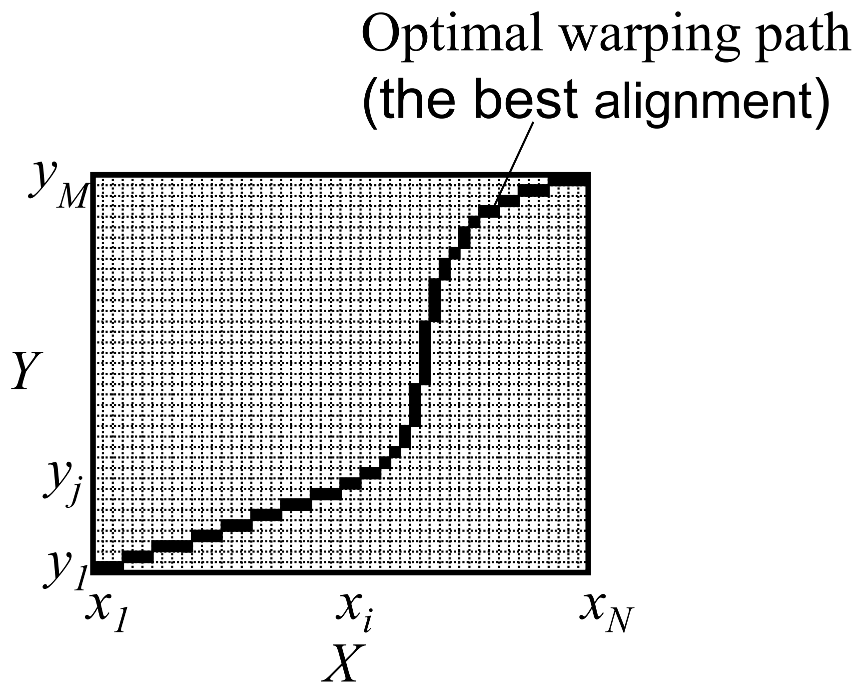
Introduction

- Data-stream applications
  - Network analysis
  - Sensor monitoring
  - Financial data analysis
  - Moving object tracking
- Goal
  - Monitor numerical streams
  - Find subsequences similar to the given query sequence
  - Distance measure: Dynamic Time Warping (DTW)

Introduction

- DTW is computed by dynamic programming
  - Stretch sequences along the time axis to minimize the distance
  - Warping path: set of grid cells in the time warping matrix





Time warping matrix

Related Work

- Sequence indexing, subsequence matching
  - Agrawal et al. (FODO 1998)
  - Keogh et al. (SIGMOD 2001)
  - Faloutsos et al. (SIGMOD 1994)
  - Moon et al. (SIGMOD 2002)
- Fast sequence matching for DTW
  - Yi et al. (ICDE 1998)
  - Keogh (VLDB 2002)
  - Zhu et al. (SIGMOD 2003)
  - Sakurai et al. (PODS 2005)

Related Work

- Data stream processing for pattern discovery
  - Clustering for data streams
    Guha et al. (TKDE 2003)
  - Monitoring multiple streams
    Zhu et al. (VLDB 2002)
  - Forecasting
    Papadimitriou et al. (VLDB 2003)
  - Detecting lag correlations
    Sakurai et al. (SIGMOD 2005)
- DTW has been studied for finite, stored sequence sets
- We address a new problem for DTW

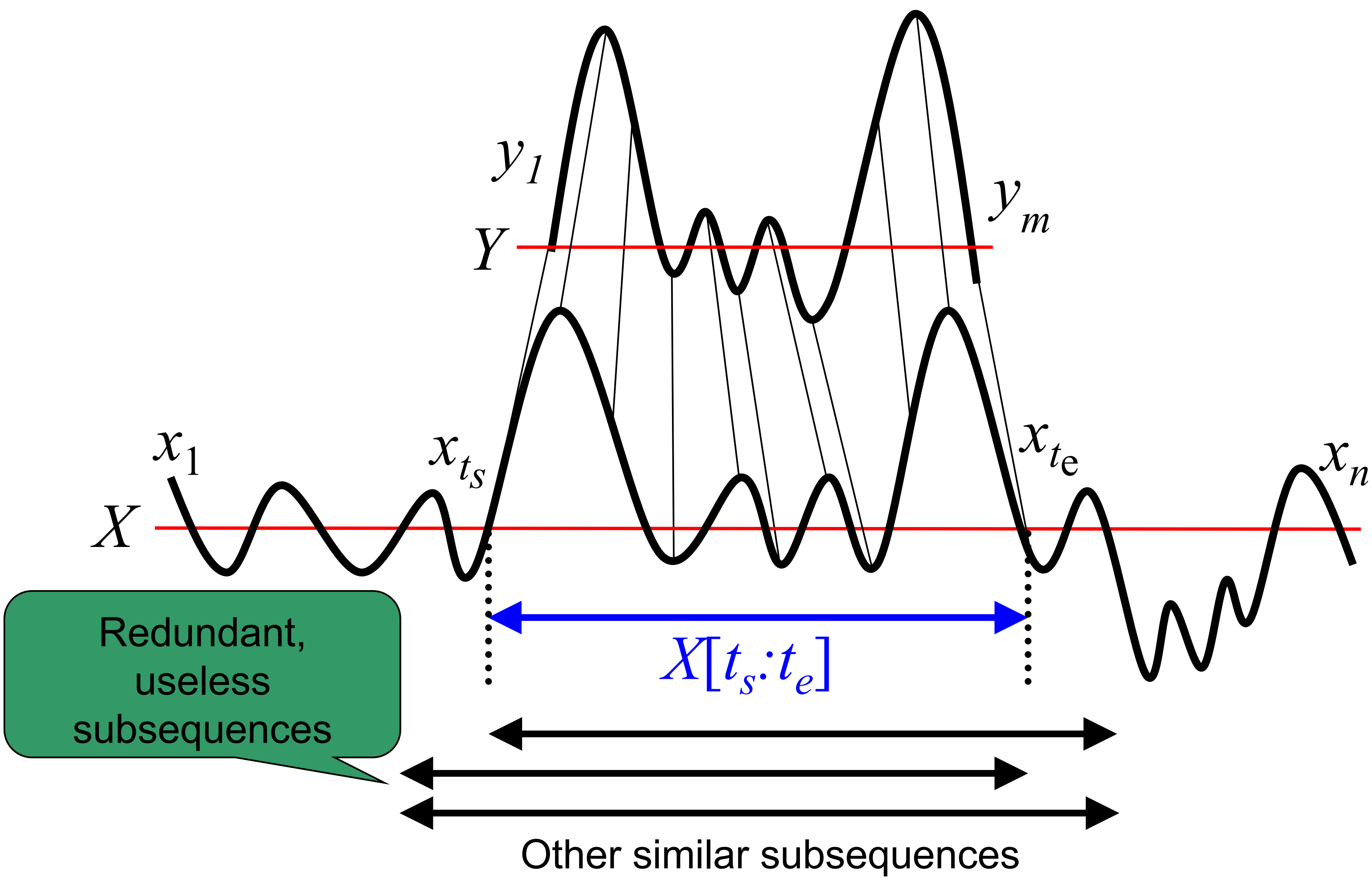
Overview

- Introduction / Related work
- Problem definition
- Main ideas
- Experimental results

Problem Definition

- Subsequence matching for data streams
  - (Fixed-length) query sequence $Y=(y_1, y_2, \ldots, y_m)$
  - Sequence (data stream) $X=(x_1, x_2, \ldots, x_n)$
  - Find all subsequences $X[t_s, t_e]$ such that $D(X[t_s : t_e], Y) \leq \varepsilon$

Subsequence Matching



Problem Definition

- Subsequence matching for data streams
  - (Fixed-length) query sequence Y
  - Sequence (data stream) $X=(x_1, x_2, \ldots, x_n)$
  - Find all subsequence $X[t_s,t_e]$ such that $D(X[t_s:t_e],Y) \leq \varepsilon$
- Multiple matches by subsequences which heavily overlap with the "local minimum" best match

  [double harm]
  - Flood the user with redundant information
  - Slow down the algorithm by forcing it to keep track of and report all these useless "solutions"
- Eliminate the redundant subsequences, and report only the "optimal" ones

Problem Definition

- Problem: Disjoint query
  - Given a threshold ε, report all $X[t_s:t_e]$ such that

1. $D(X[t_s : t_e], Y) \leq \varepsilon$
2. Only the local minimum

   $D(X[t_s : t_e], Y)$ is the smallest value in the group of overlapping subsequences that satisfy the first condition

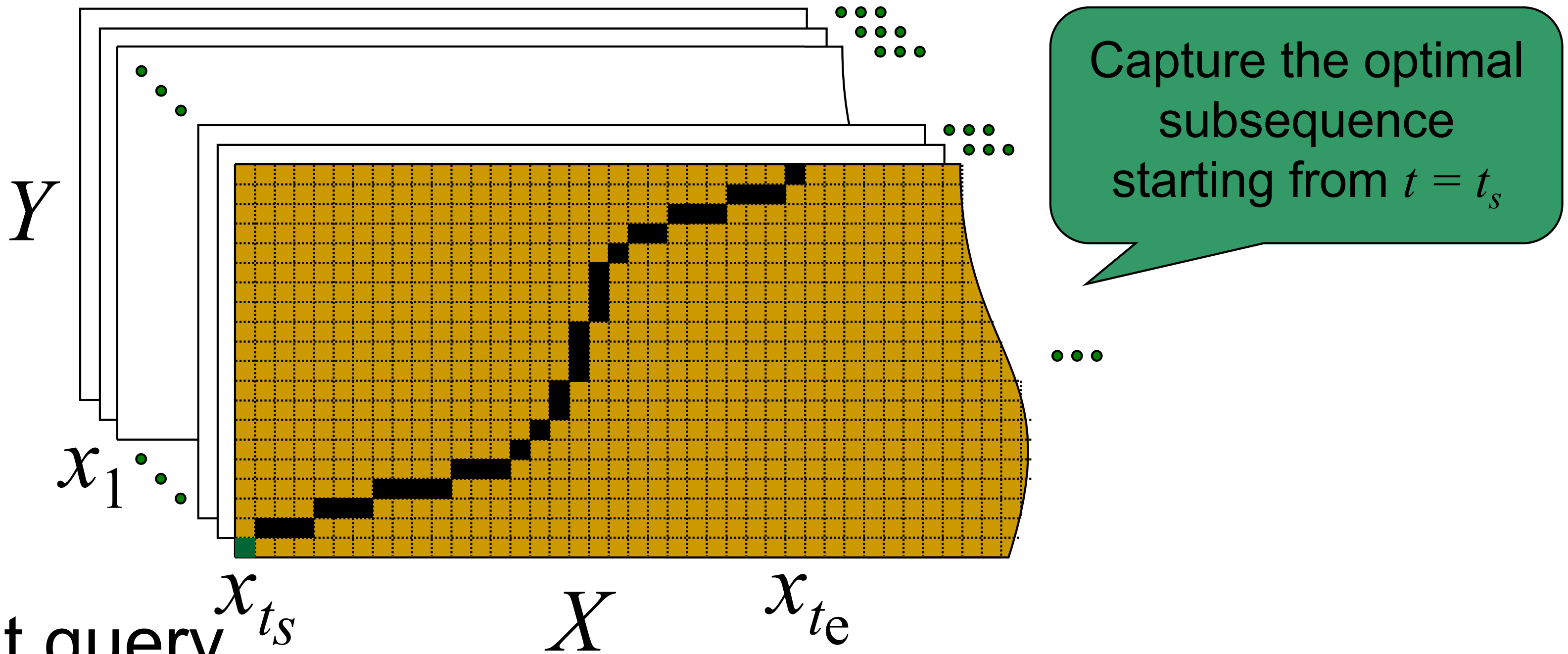
- Additional challenges: streaming solution
  - Process a new value of X efficiently
  - Guarantee no false dismissals
  - Report each match as early as possible

Overview

- Introduction / Related work
- Problem definition
- Main ideas
- Experimental results

Why not 'naive'?

- Compute the time warping matrices starting from every time-tick
  - Need $O(n)$ matrices, $O(nm)$ time per time-tick



- Disjoint query
  - Compute all the possible subsequences and then choose the optimal ones

Main idea (1)

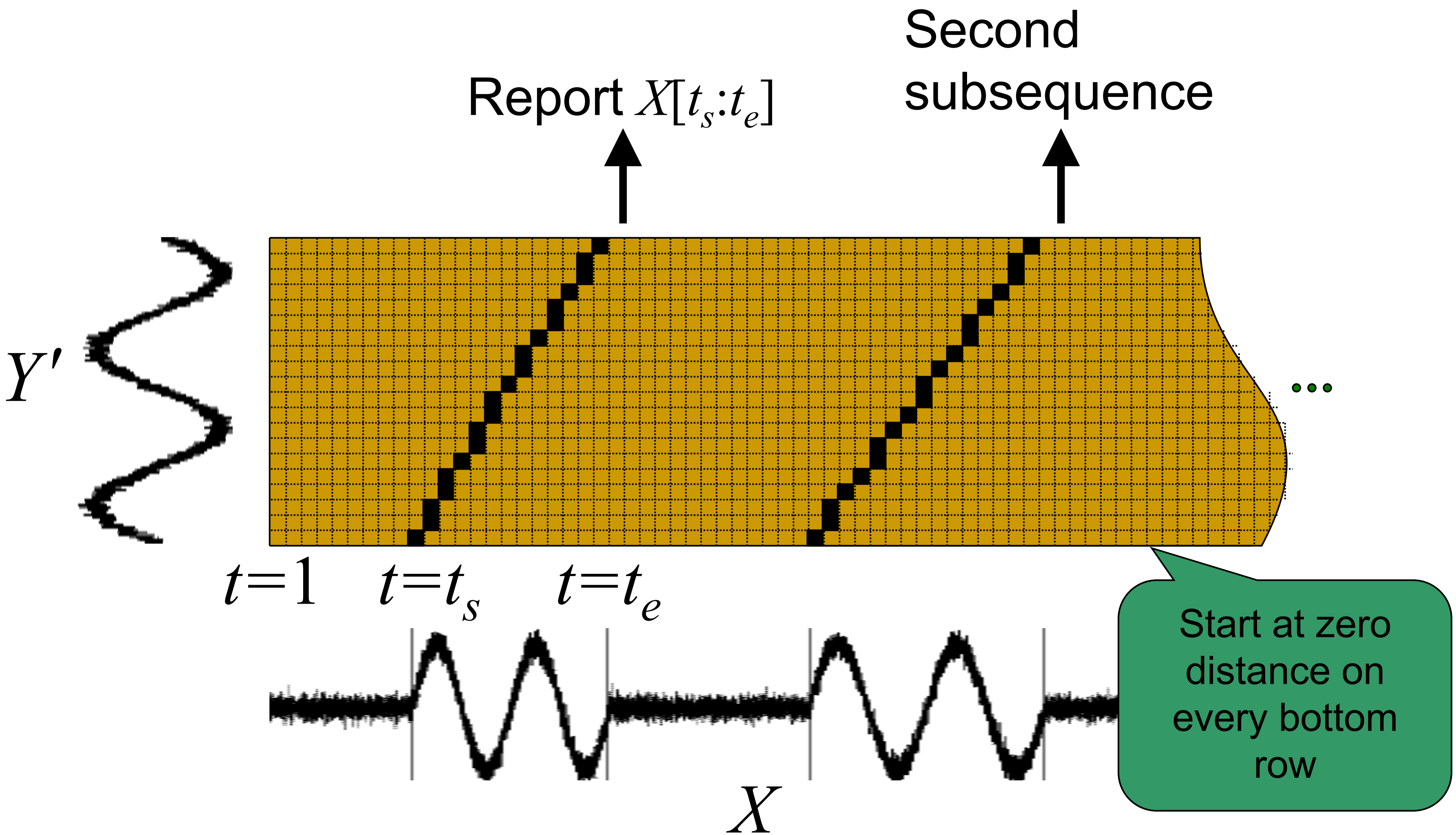
- Star-padding
  - Use only a single matrix
    (the naïve solution uses n matrices)
  - Prefix Y with '*', that always gives zero distance
  - instead of $Y=(y_1, y_2, \ldots, y_m)$, compute distances with Y'

$$Y' = (y_0, y_1, y_2, \ldots, y_m)$$

$$y_0 = (-\infty : +\infty)$$

  - $O(m)$ time and space (the naïve requires $O(nm)$)

SPRING



Second subsequence
Report $X[t_s:t_e]$
Y'
...
$t=1$
$t=t_s$
$t=t_e$
Start at zero distance on every bottom row
X

Main idea (2)

- STWM (Subsequence Time Warping Matrix)
  - Problem of the star-padding: we lose the information about the starting time-tick of the match
  - After the scan, “which is the optimal subsequence?”
- Elements of STWM
  - Distance value of each subsequence
  - Starting position
- Combination of star-padding and STWM
  - Efficiently identify the optimal subsequence in a stream fashion

Main idea (3)

- Algorithm for disjoint queries
- Designed to:
  - Guarantee no false dismissals
  - Report each match as early as possible

Algorithm for disjoint queries

1. Update m elements (distance and starting position) at every time-tick
2. Keep track of the minimum distance d_{min} when a subsequence within ε is found
3. Report the subsequence that gives d_{min} if (a) and (b) are satisfied
   (a) the captured optimal subsequence cannot be replaced by the upcoming subsequences
   (b) the upcoming subsequences dot not overlap with the captured optimal subsequence

Algorithm for disjoint queries

- distance (upper number), starting position (number in parentheses)
- X=(5,12,6,10,6,5,13), Y=(11,6,9,4), $\varepsilon = 20$

$y_4 = 4$	54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$	53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$	37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$	36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Algorithm for disjoint queries

- distance (upper number), starting position (number in parentheses)
- X=(5,12,6,10,6,5,13), Y=(11,6,9,4), $\varepsilon = 20$
- ■ optimal subsequence, ■ ■ redundant subsequences

$y_4 = 4$	54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$	53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$	37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$	36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Algorithm for disjoint queries

- distance (upper number), starting position (number in parentheses)
- X=(5,12,6,10,6,5,13), Y=(11,6,9,4), $\varepsilon = 20$
- ■ optimal subsequence, ■ ■ redundant subsequences

$y_4 = 4$	54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$	53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$	37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$	36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Algorithm for disjoint queries

- distance (upper number), starting position (number in parentheses)
- X=(5,12,6,10,6,5,13), Y=(11,6,9,4), $\varepsilon = 20$
- ■ optimal subsequence, ■ ■ redundant subsequences

$y_4 = 4$	54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$	53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$	37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$	36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Algorithm for disjoint queries

- Guarantee to report the optimal subsequence
  (a) The captured optimal subsequence cannot be replaced
  (b) The upcoming subsequences do not overlap with the captured optimal subsequence

$y_4 = 4$	54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$	53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$	37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$	36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Algorithm for disjoint queries

- Guarantee to report the optimal subsequence
  - Finally report the optimal subsequence $X[2:5]$ at $t=7$
  - Initialize the distance values ($d_2=51$, $d_3=18$, $d_4=88$)

$y_4 = 4$	54 (1)	110 (2)	14 (2)	38 (2)	6 (2)	7 (2)	88 (2)
$y_3 = 9$	53 (1)	46 (2)	10 (2)	2 (2)	10 (4)	17 (4)	18 (4)
$y_2 = 6$	37 (1)	37 (2)	1 (2)	17 (4)	1 (4)	2 (4)	51 (4)
$y_1 = 11$	36 (1)	1 (2)	25 (3)	1 (4)	25 (5)	36 (6)	4 (7)
x_t	5	12	6	10	6	5	13
t	1	2	3	4	5	6	7

Overview

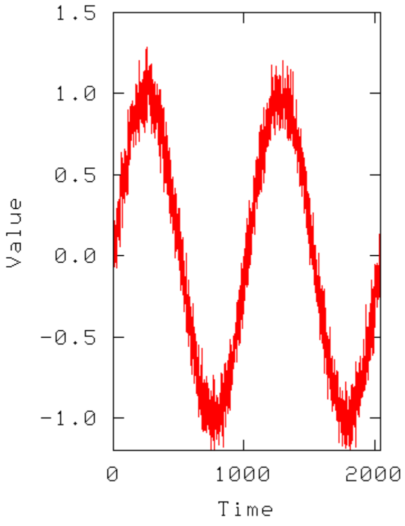
- Introduction / Related work
- Problem definition
- Main ideas
- Experimental results

Experimental Results

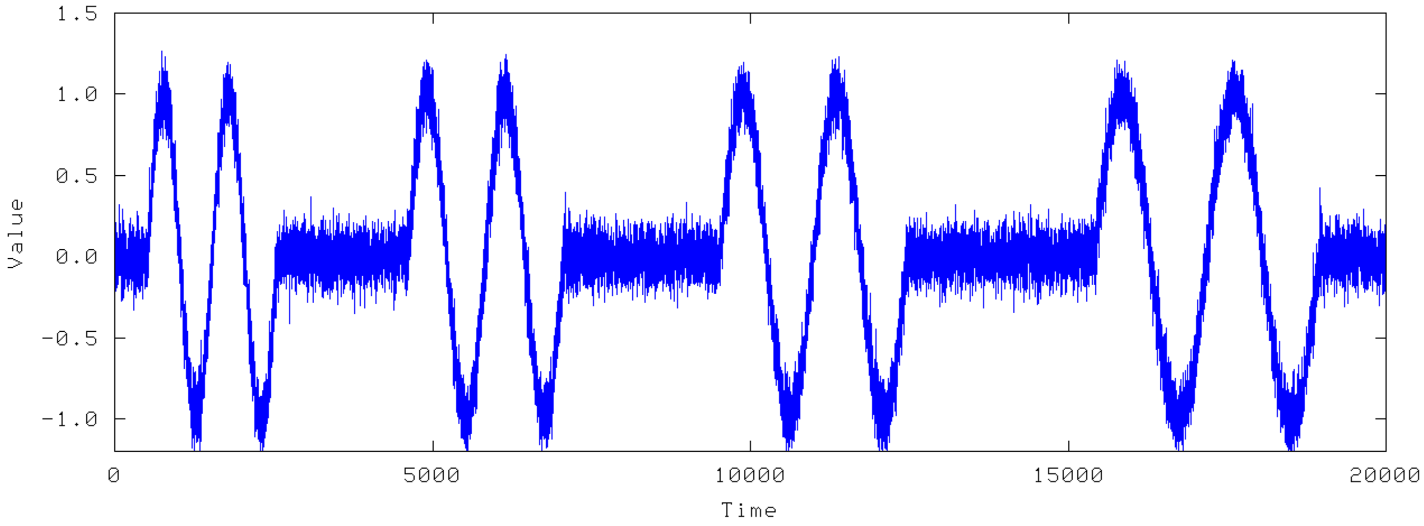
- Experiments with real and synthetic data sets
  - *MaskedChirp, Temperature, Kursk, Sunspots*
- Evaluation
  - Accuracy for pattern discovery
  - Computation time
  - (Memory space consumption)

Pattern Discovery

- *MaskedChirp*



Query sequence

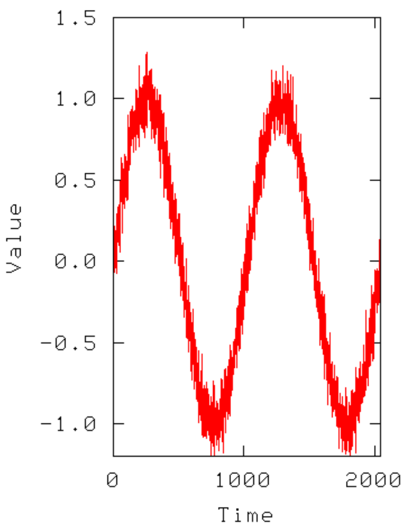


Data stream

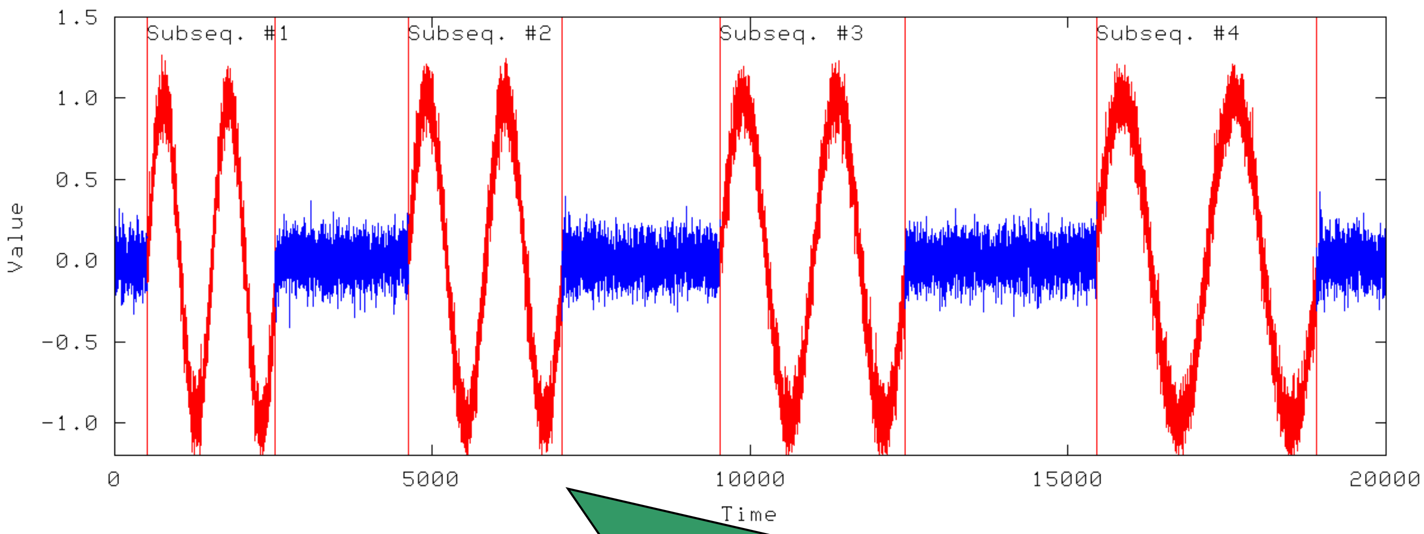
Pattern Discovery

- *MaskedChirp*

SPRING identifies all sound parts with varying time periods



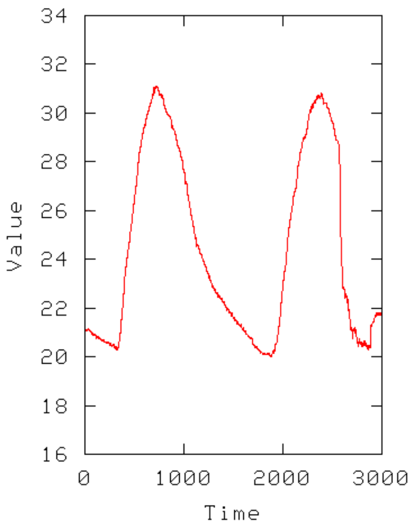
Query sequence



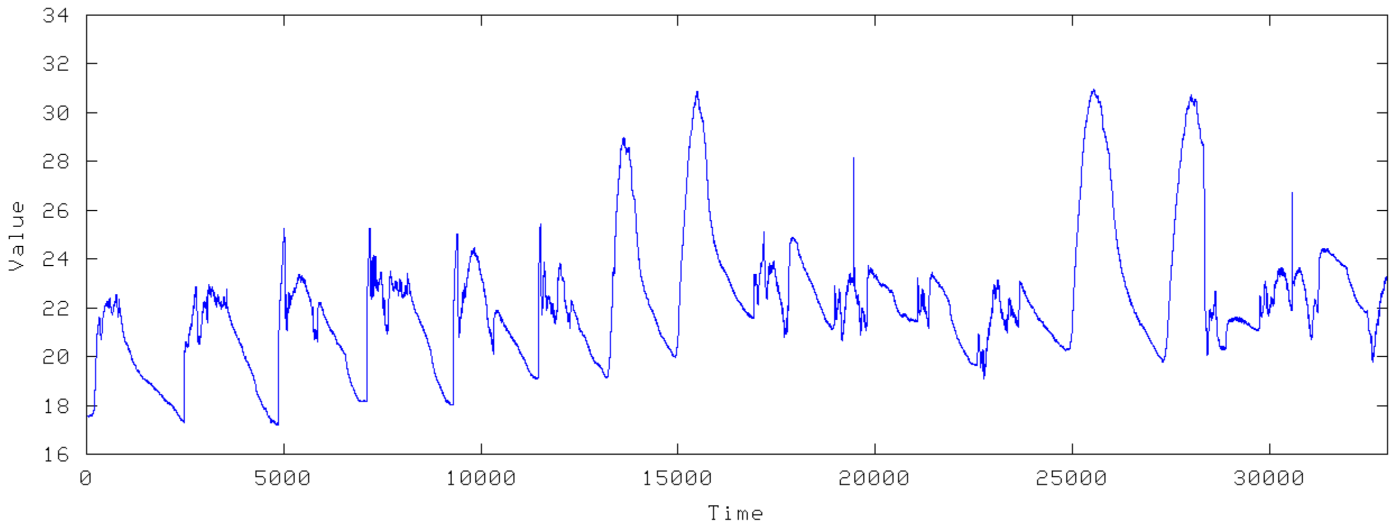
The output time of each captured subsequence is very close to its end position

Pattern Discovery

- *Temperature*



Query sequence

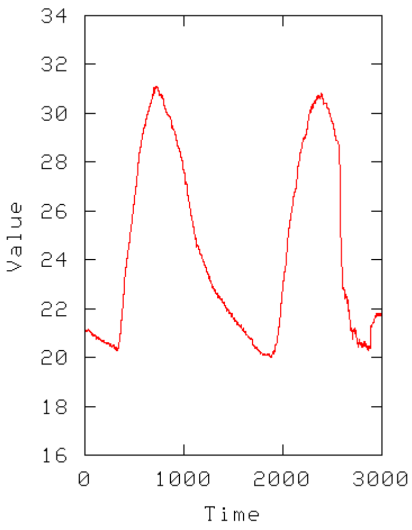


Data stream

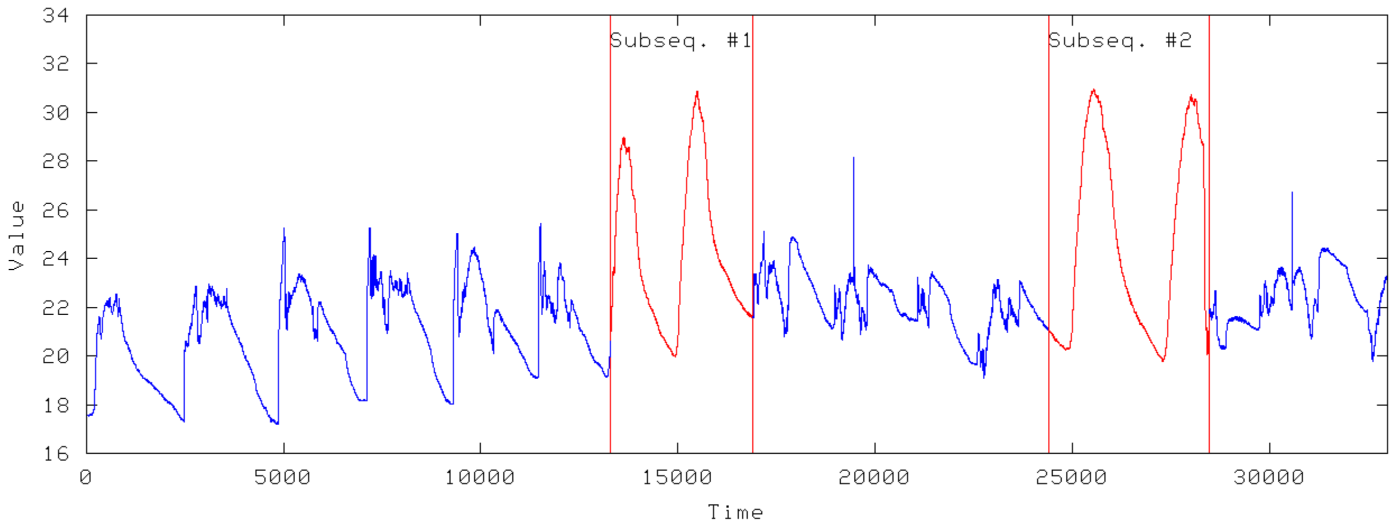
Pattern Discovery

- *Temperature*

SPRING finds the days when the temperature fluctuates from cool to hot



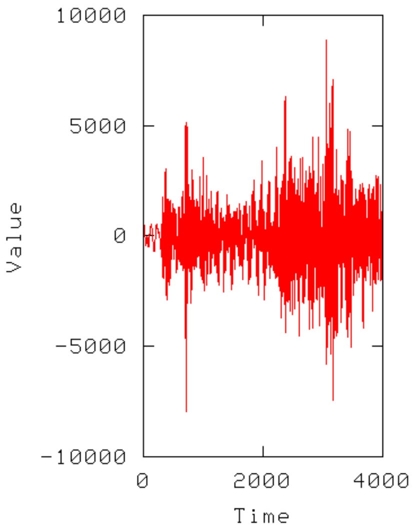
Query sequence



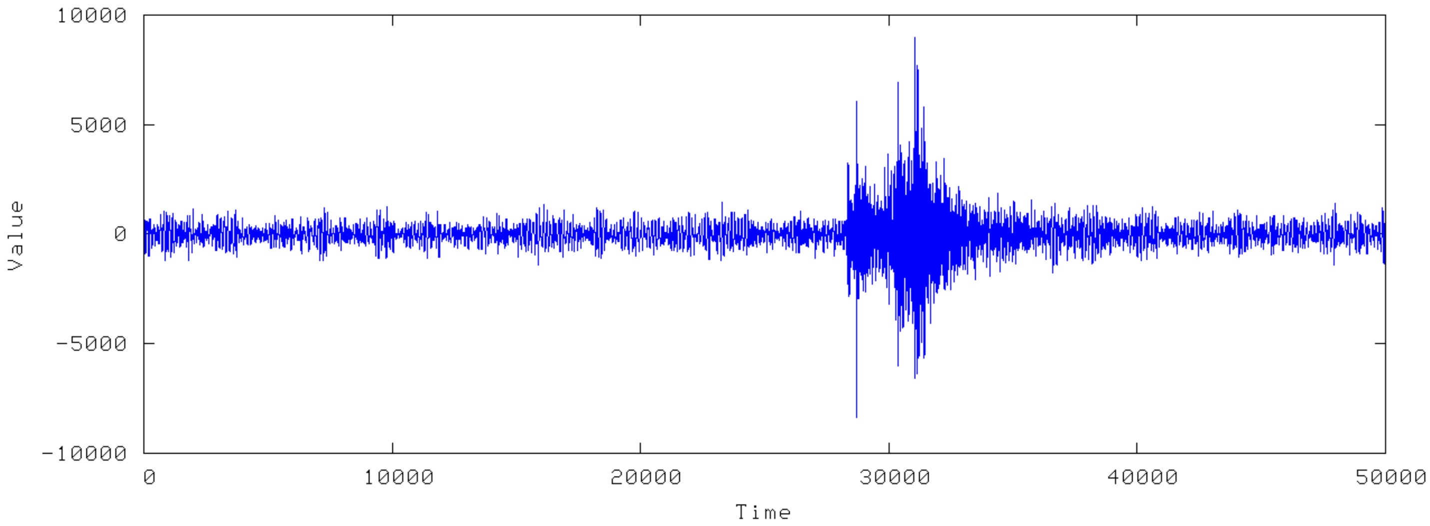
Data stream

Pattern Discovery

- *Kursk*



Query sequence

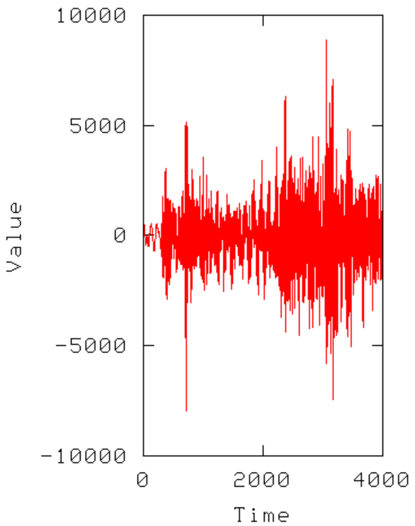


Data stream

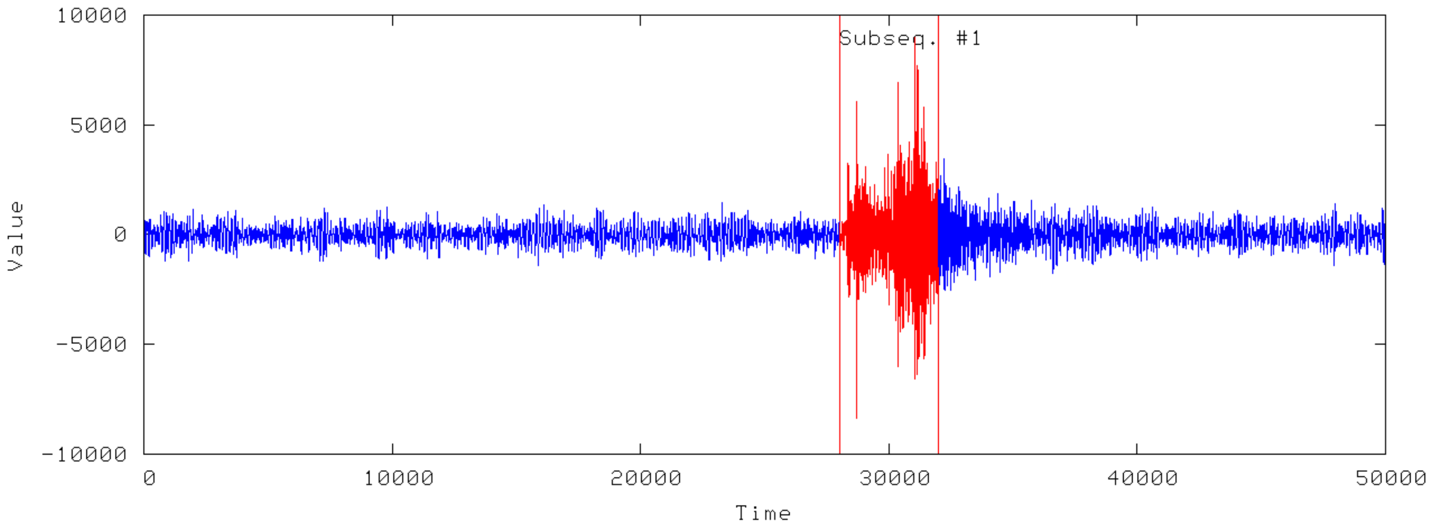
Pattern Discovery

- *Kursk*

SPRING is not affected by the difference in the environmental conditions



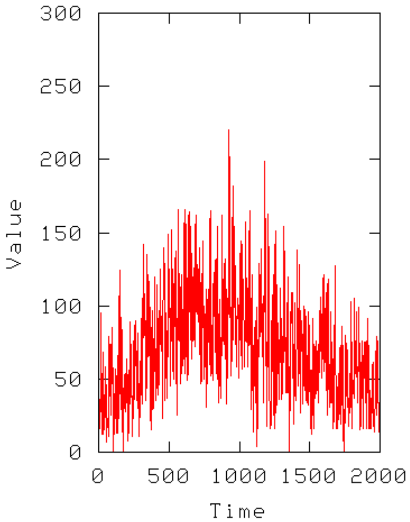
Query sequence



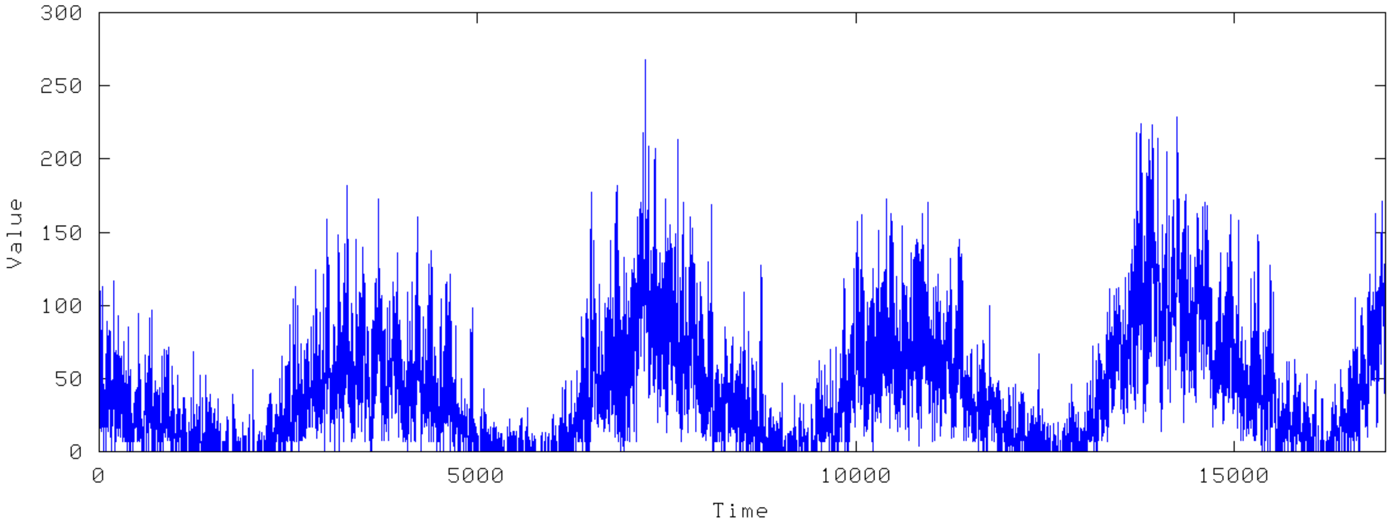
Data stream

Pattern Discovery

- *Sunspots*



Query sequence

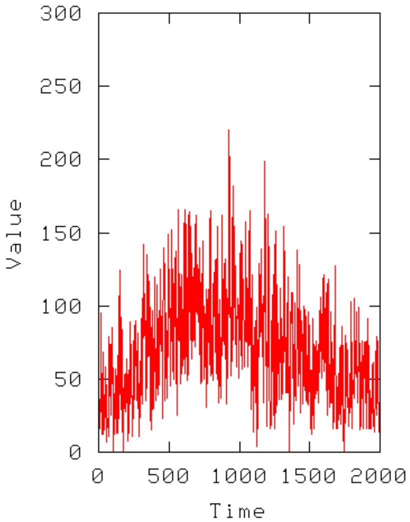


Data stream

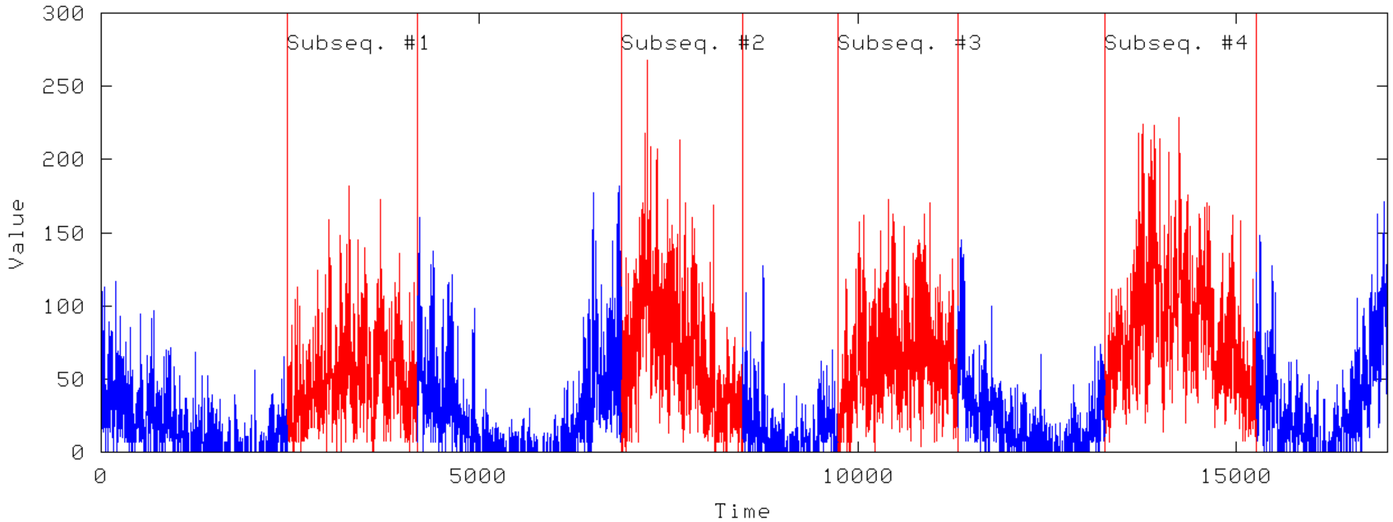
Pattern Discovery

- *Sunspots*

SPRING can capture bursty periods and identify the time-varying periodicity



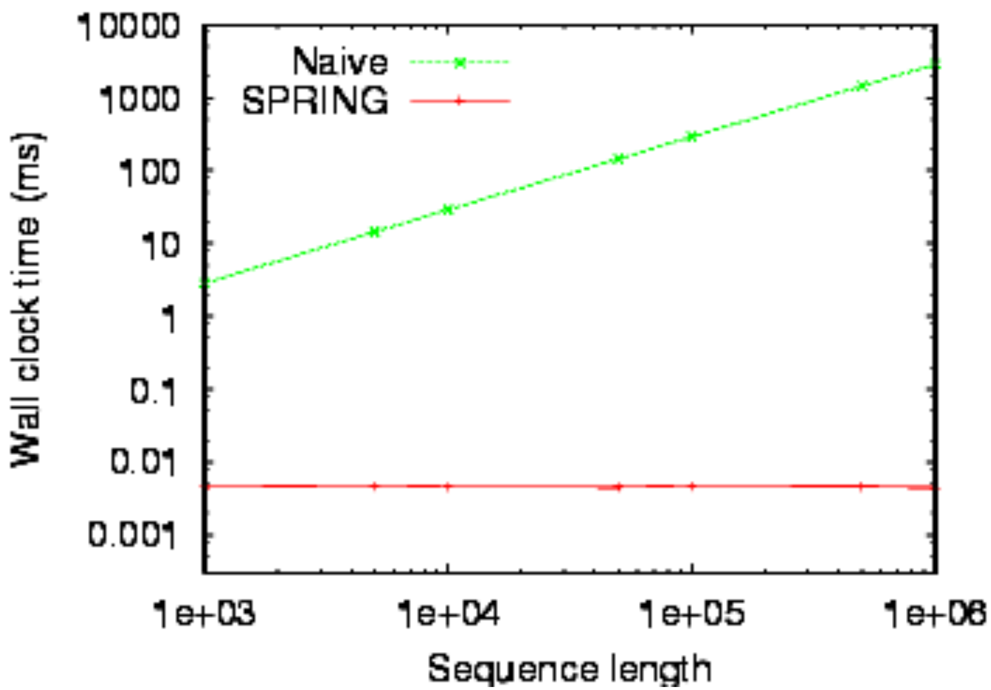
Query sequence



Data stream

Computation time

- Wall clock time per time-tick
  - Naïve method: $O(nm)$
  - SPRING: $O(m)$, not depend on sequence length n



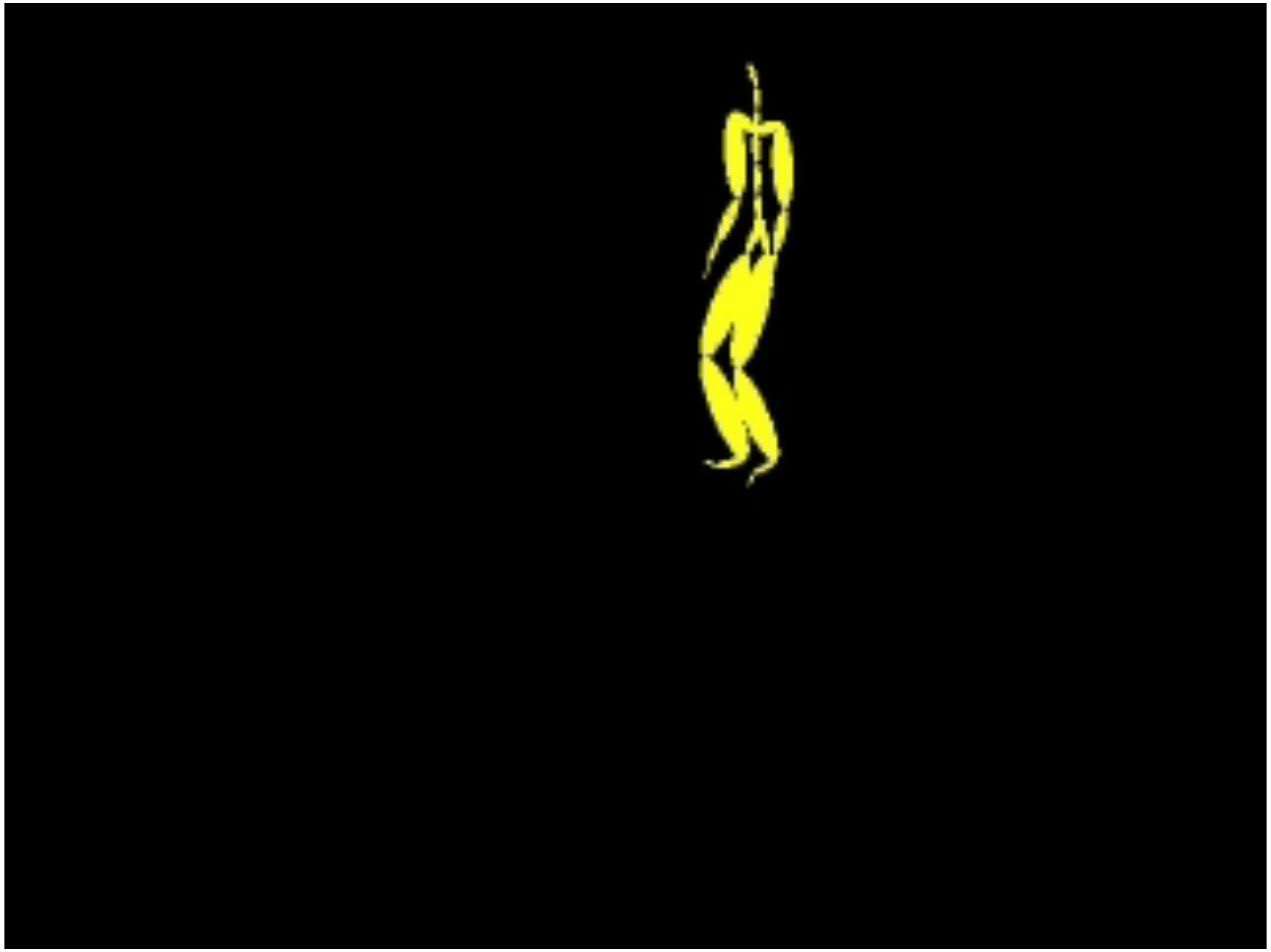
Extension to multiple streams

- Motion capture data
  - Place special markers on the joints of a human actor
  - Record their x-, y-, z-velocities
  - Use 16-dimensional sequences
  - Capture motions based on the similarity of rotational energy

$$E_{rotation} = \frac{1}{2} I \omega^2$$

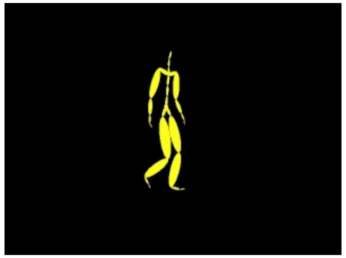
$E_{rotation}$: rotational energy
I : moment of inertia
ω : angular velocity

High-speed Motion Capture

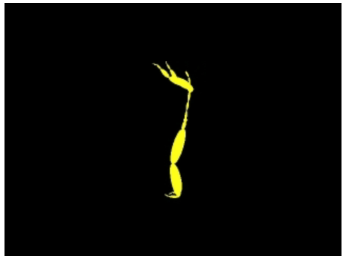


High-speed Motion Capture

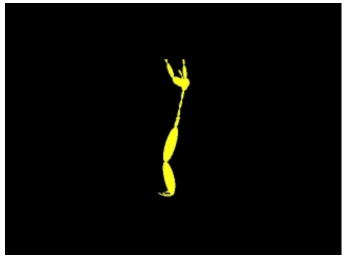
- Recognize all motions in a stream fashion
  - Entertainment applications, etc


Walk

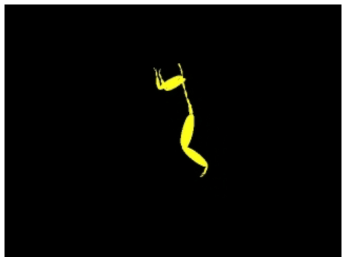

Swing

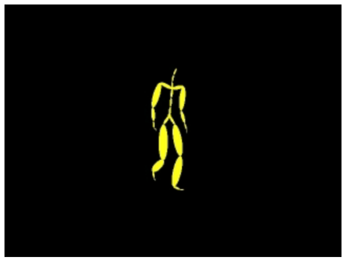

Rotate

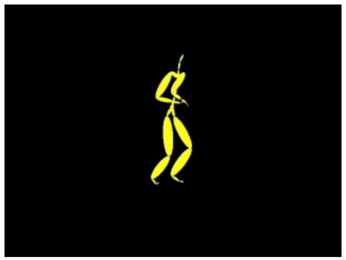

Swing

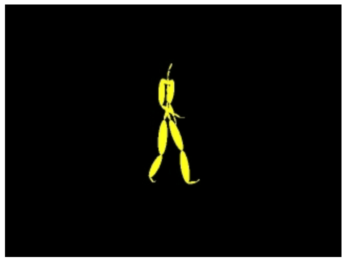

Rotate


One-leg jump


Jump


Walk


Run


Walk

Conclusions

- Subsequence matching under the DTW distance over data streams

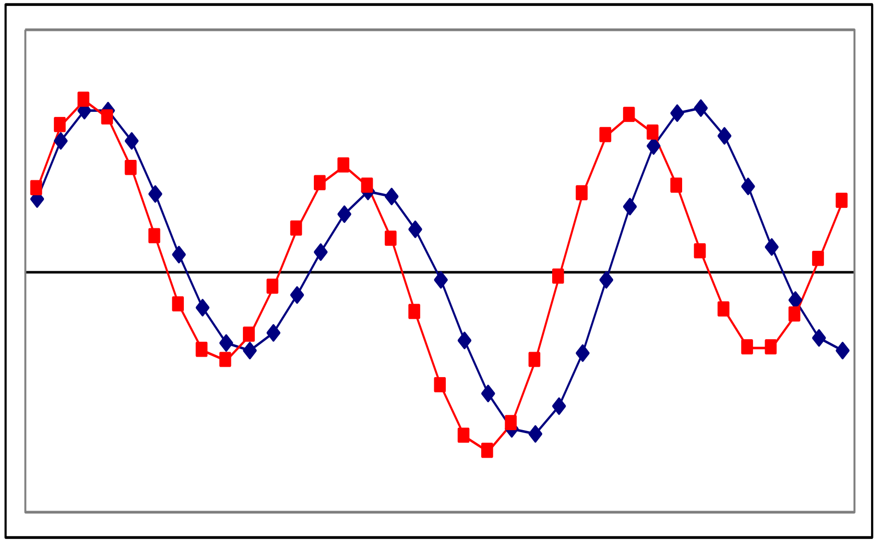
1. High-speed, and low memory consumption
   - $O(m)$ time and space; not depend on n
2. Accuracy
   - Guarantee no false dismissals

- Stored data sets
  - SPRING can be applied to stored sequence sets

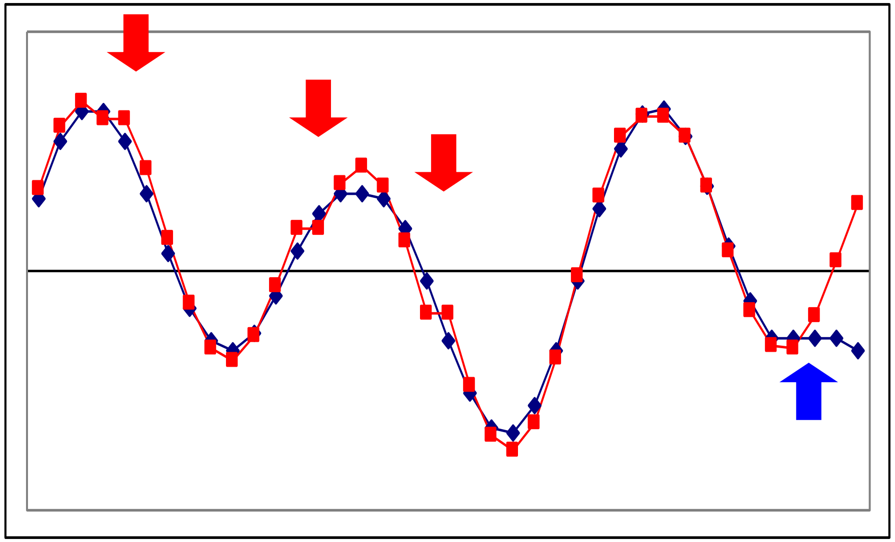
Appendix

Mini-introduction to DTW

- DTW allows sequences to be stretched along the time axis
  - Minimize the distance of sequences
  - Insert ‘stutters’ into a sequence
  - THEN compute the (Euclidean) distance



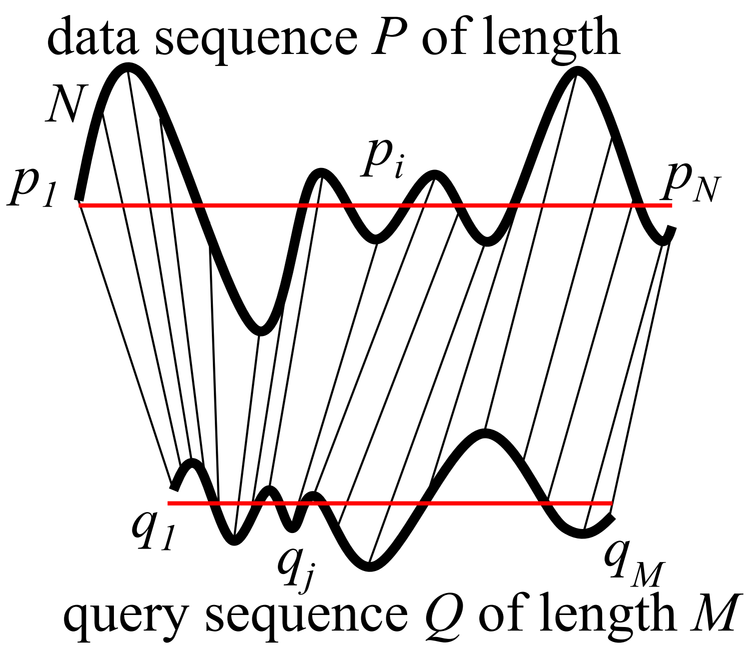
original

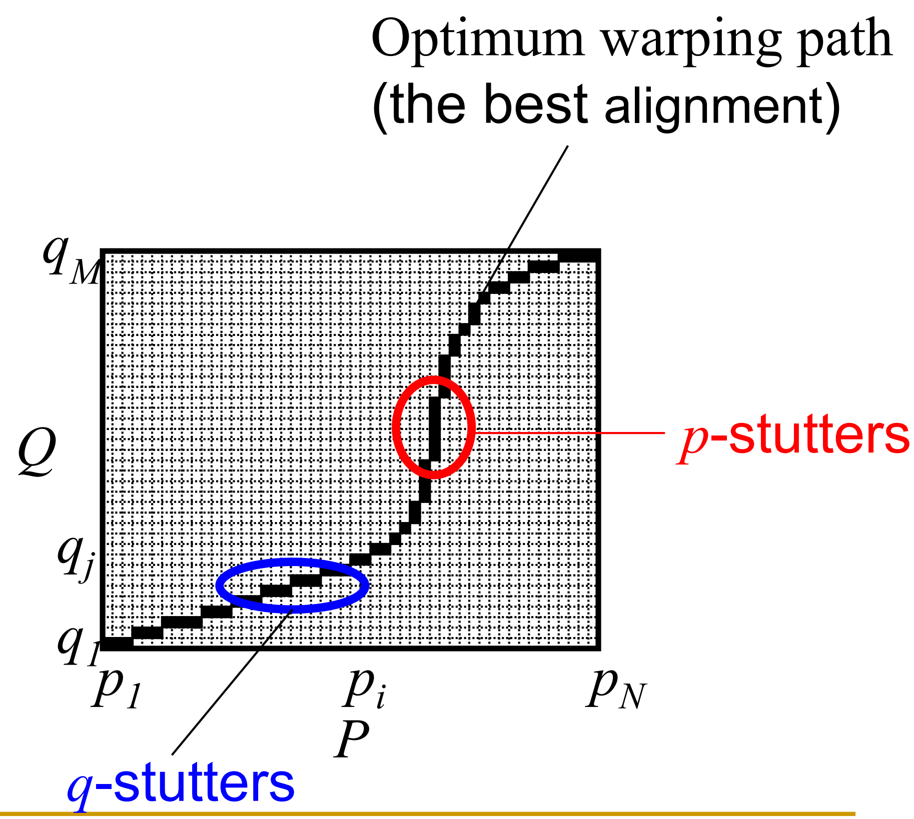


‘stutters’:

Mini-introduction to DTW

- DTW is computed by dynamic programming
  - Warping path: set of grid cells in the time warping matrix





Mini-introduction to DTW

- DTW is computed by dynamic programming

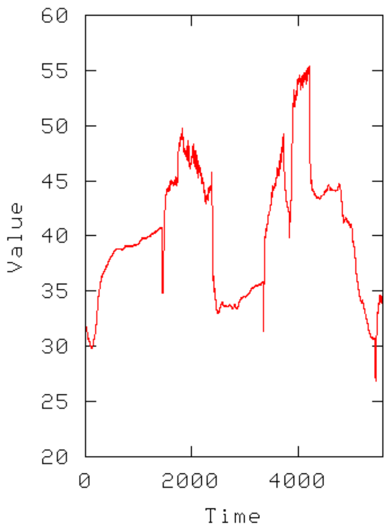
$$p_1, p_2, \ldots, p_{i,;} \qquad q_1, q_2, \ldots, q_j$$

$$D_{dtw}(P,Q) = f(N,M)$$

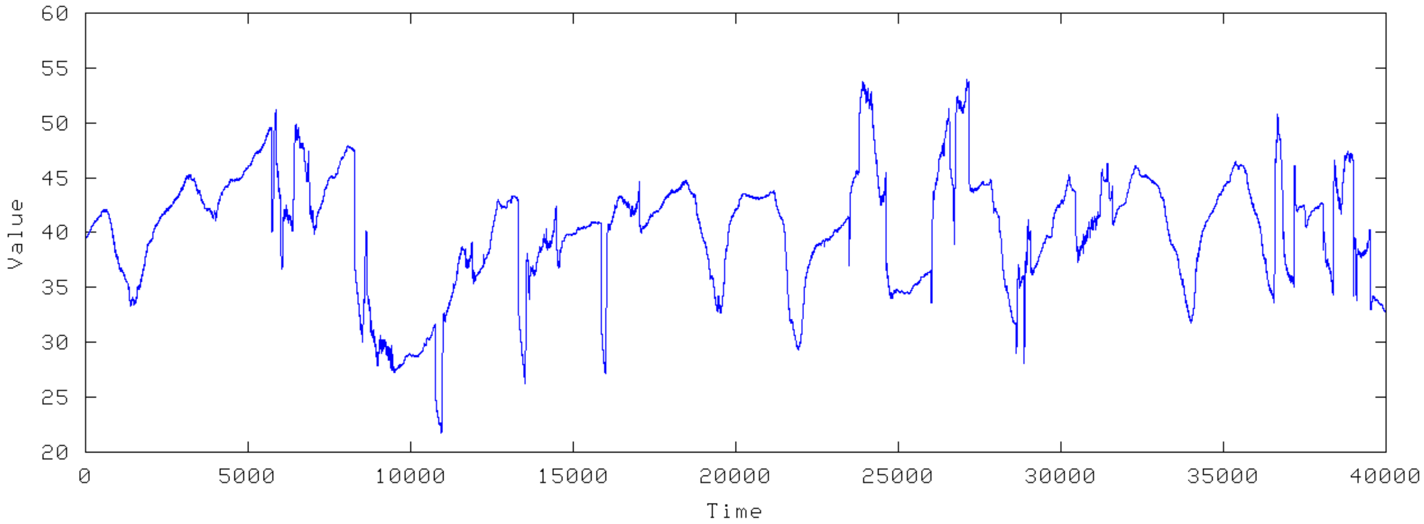
$$f(i,j) = \left\| p_i - q_j \right\| + \min \begin{cases} f(i, j-1) & p\text{-stutter} \\ f(i-1, j) & q\text{-stutter} \\ f(i-1, j-1) & \text{no stutter} \end{cases}$$

Pattern Discovery

- *Humidity*



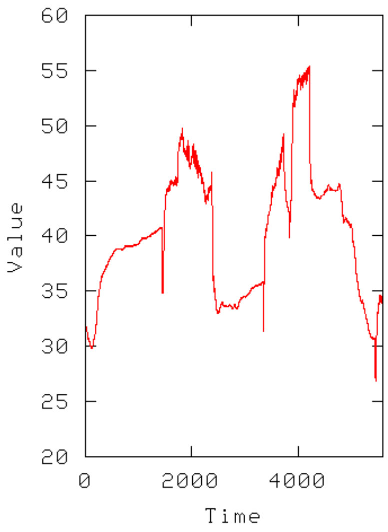
Query sequence



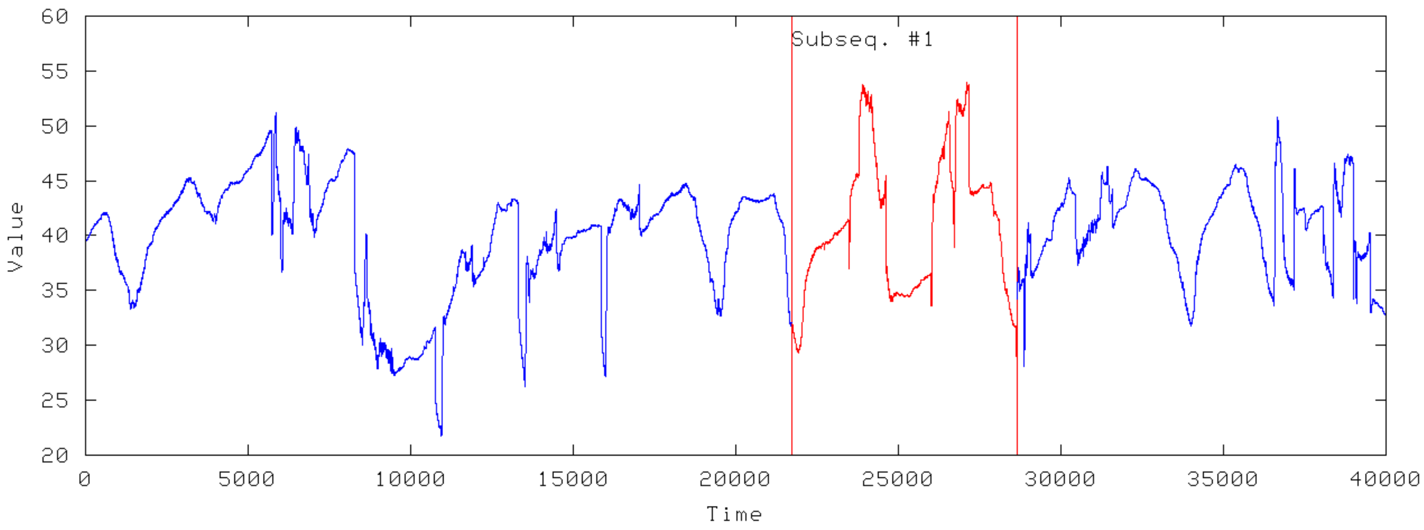
Data stream

Pattern Discovery

- *Humidity*



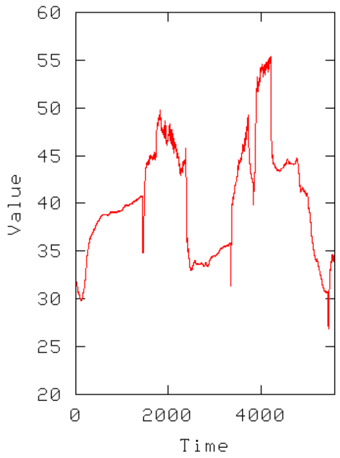
Query sequence



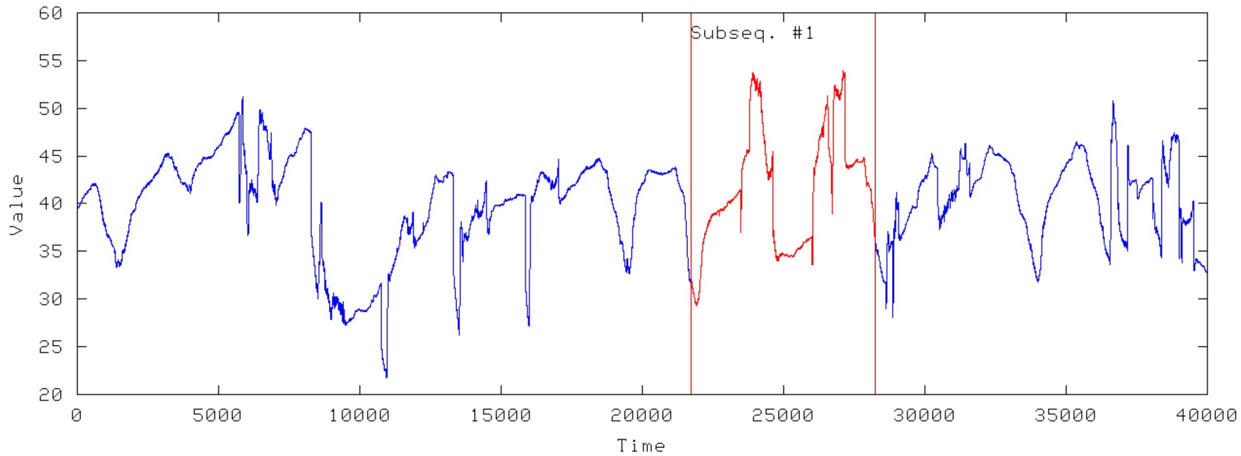
Data stream

Two Algorithms of SPRING

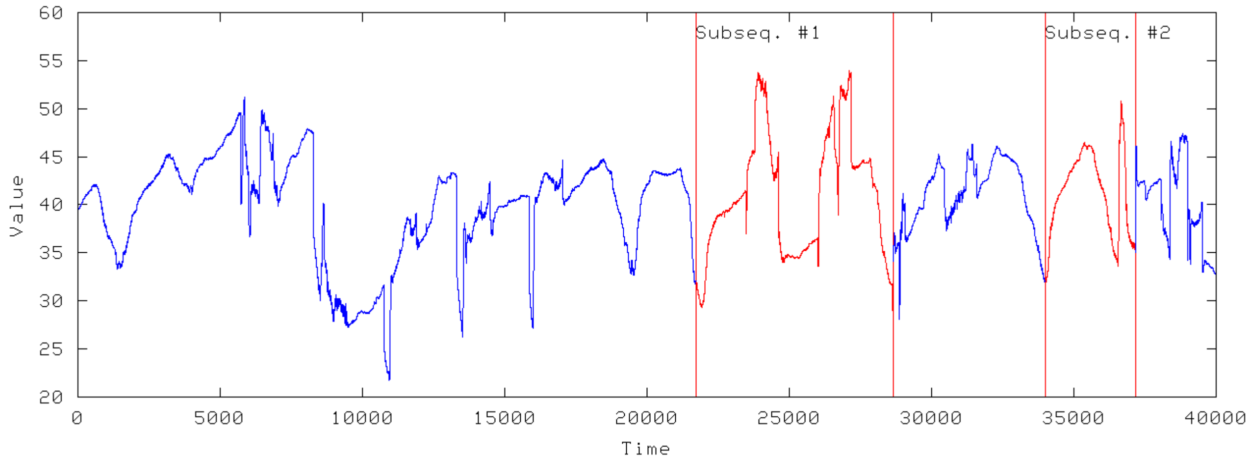
- *SPRING-optimal*



Query sequence



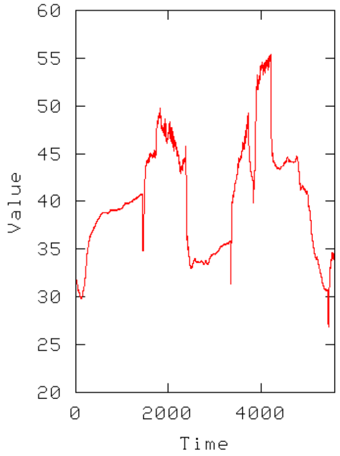
$\varepsilon = 10{,}000$



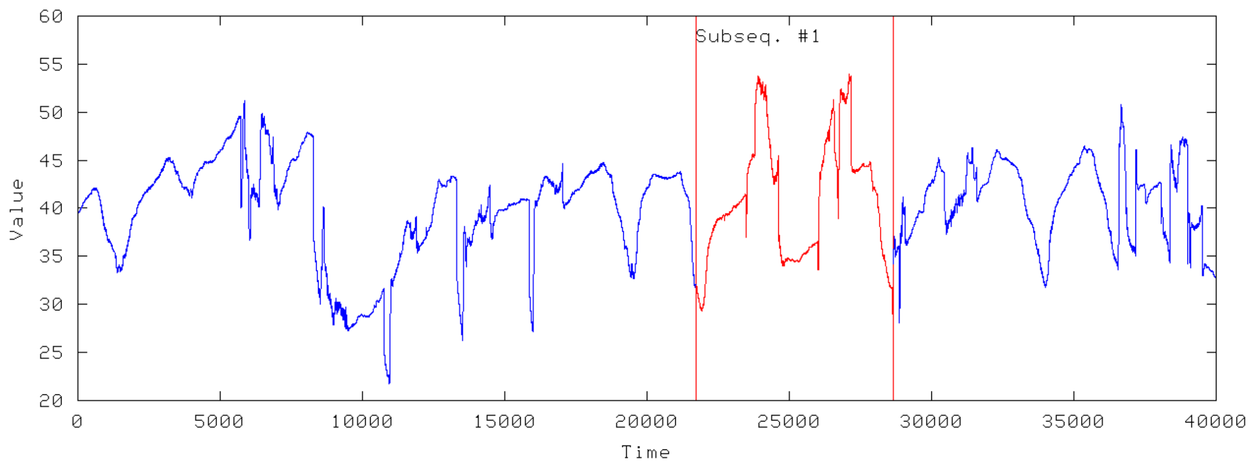
$\varepsilon = 15{,}000$

Two Algorithms of SPRING

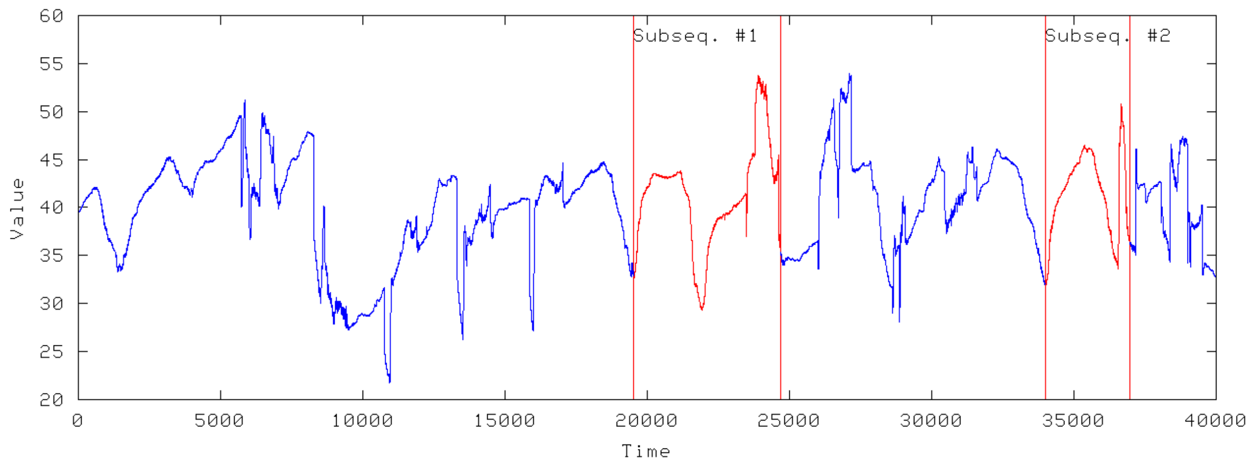
- *SPRING-first*



Query sequence



ε = 10,000



ε = 15,000

Memory space consumption

- Memory space for time warping matrix (matrices)
  - Naïve method: $O(nm)$
  - SPRING: $O(m)$, not depend on sequence length n
  - SPRING (path): clearly lower than that of the naïve method