Mining Time Series Data 3

Acknowledgements: Eamonn Keogh, Mueen Abdullah

Similarity Search On Large Datasets

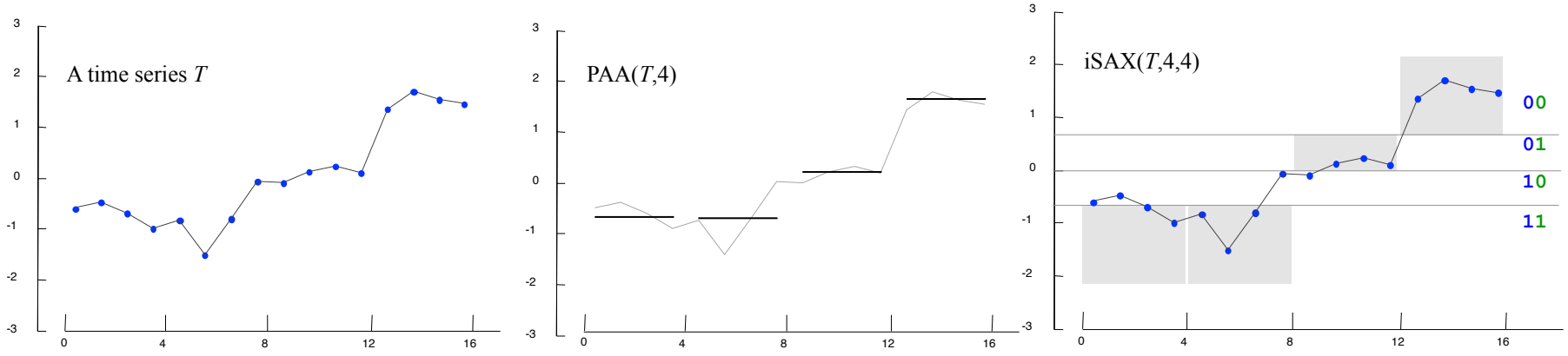
- High I/O costs a challenge
- Utilize an index to speed up similarity search
  - High dimensionality of time series makes indexing a challenge
- Generic framework
  - Map the data to a reduced representation
  - Obtain a candidate set in the reduced space
  - Verify results in the native representation
- Efficiency and effectiveness affected by characteristics of the reduced representation

Classic SAX

- **S**ymbolic **A**ggregate appro**X**imation (SAX)
  - **(1)** Represent a time series T of length ***n*** in ***w***-dimensional space using Piecewise Aggregate Approximation (PAA)
    - T typically normalized to $\mu = 0$, $\sigma = 1$
    - PAA(T,w) $\equiv \bar{T} = \bar{t}_1, \ldots, \bar{t}_w$ where

$$\bar{t}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} T_j$$

  - **(2)** Discretize into a vector of symbols
    - Breakpoints map to a small alphabet ***a*** of symbols



Classic SAX (cont.)

- SAX lower bounds Euclidean distance
- Why not just index using SAX?
- Example: index 1,000,000 time series using SAX
  - Choose SAX parameters
    - Symbol cardinality = 8, wordlength = 4
    - 8^4 = 4,096 possible SAX word labels
  - Place time series which map to the same label in the same file on disk
  - Compute label for query and retrieve matching file
    - Time series in file likely to be good approximate matches
  - Average label occupancy 1,000,000/4,096 = ~244 (reasonable)

Classic SAX (cont.)

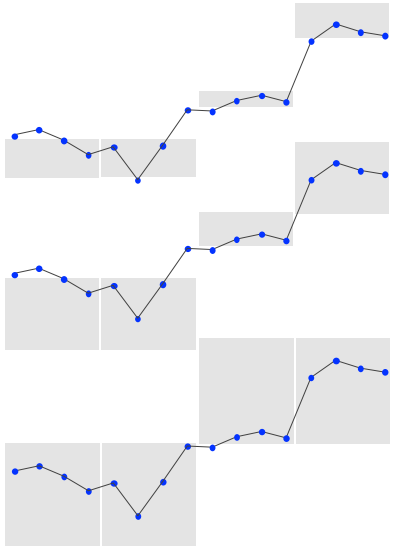
- In practice, the distribution of time series to SAX word labels is not uniform and is highly skewed!
  - Empty
  - Disproportionate percentage of the dataset
- **Ideal condition:** We want to give a threshold *th*, and have the number of entries *n* mapped to a label to be $1 \leq n \leq th$
  - Favor larger *n*
- How can we achieve this? We need to make SAX more flexible

*i*SAX Representation

- SAX uses a single hard-coded cardinality
  - Unable to differentiate **only** on dimensions of interest
- The indexing problem can be solved if we extend SAX to allow:
  - **Different cardinalities *within* a single word**
  - **Comparison of words with different cardinalities**
- This extension is called *indexable* SAX (*i*SAX)

*i*SAX Representation (cont.)

- Multi-resolution property allows conversion to any lower resolution that differs by a power of two



= {6, 6, 3, 0} = {110 ,110 ,011 ,000}

= {3, 3, 1, 0} = {11 ,11 ,01 ,00 }

= {1, 1, 0, 0} = {1 ,1 ,0 ,0 }

- Lower bounding distance between *i*SAX words enforced through examination of both sets of breakpoints
- *i*SAX offers a bit aware, quantized, multi-resolution representation with variable granularity

Comparing Different Cardinality

- iSAX(T, 4, 8) = T^8 = {110, 110, 011, 000}
- iSAX(S, 4, 2) = S^2 = {0 , 0 , 1 , 1 }
- How do we compare T and S?
  - Promoting S^2 word as S^8 = {0**, 0**, 1**, 1**}
  - For each unknown bit $S^k i$,

IF $S^k i$ forms a prefix for $T^8 i$ **THEN**,

*i = $T^8 i$ for all unknown bits

ELSE IF $S^k i$ is lexicographically smaller than corresponding bits in $T^8 i$, **THEN**,

*i = 1 for all unknown bits

ELSE

*i = 0 for all unknown bits

Indexing with *i*SAX

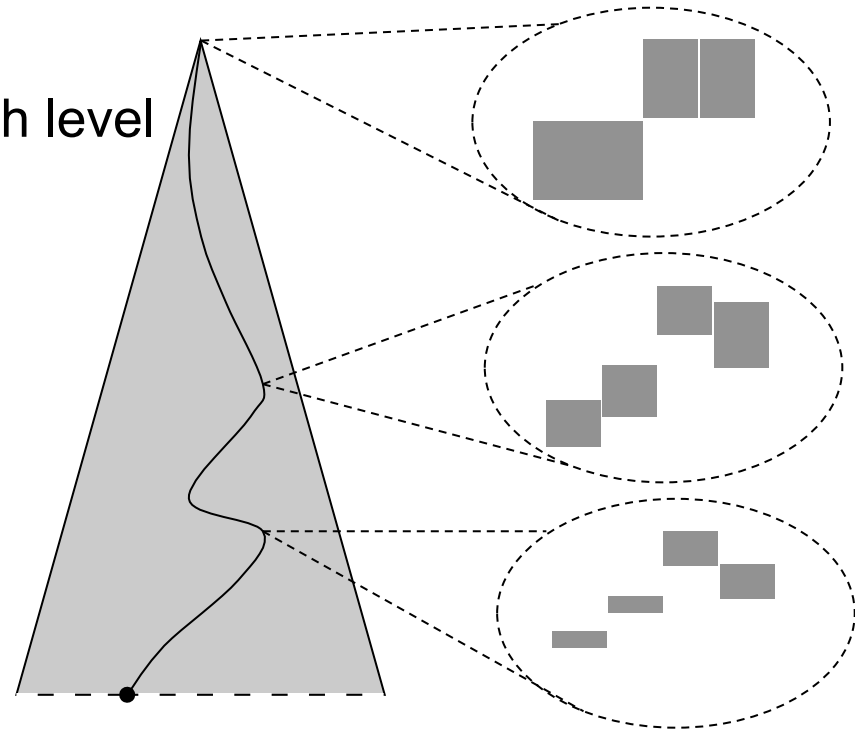
- Split a set of time series represented by a common *i*SAX word into mutually exclusive subsets (multi-resolution property / examining more bits):
  - Increase cardinality along dimensions ***d***, word length ***w***, $1 \leq \boldsymbol{d} \leq \boldsymbol{w}$
  - Fan-out rate bound by $2^{\boldsymbol{d}}$
- Iterative doubling
  - Alignment of breakpoints overlap
- Allows for index structures which are hierarchical, with non-overlapping regions, and a controlled fan-out rate

Indexing with *i*SAX (cont.)

- Demonstrate using simple tree-based index
  - (base cardinality ***b*** (optional), word length ***w***, threshold ***th***)
  - Hierarchically subdivides SAX space until num. entries ≤ ***th***

- **Approximate Search**
  - Match *i*SAX representation at each level

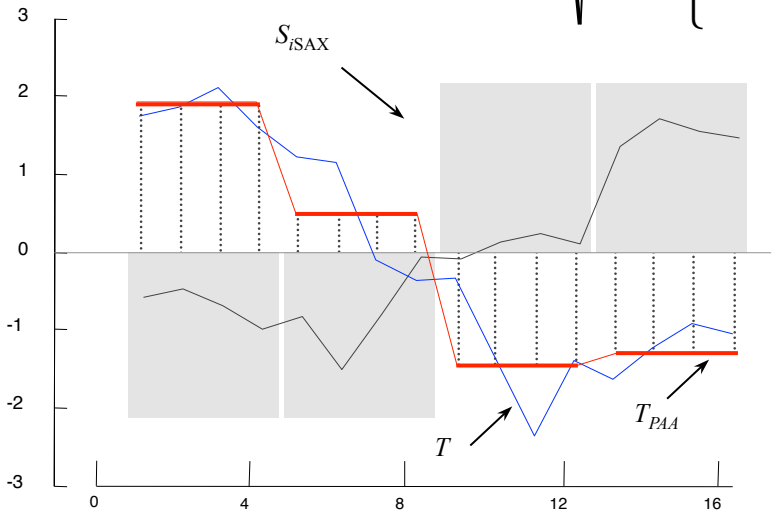
- **Exact Search**
  - Leverage approximate search
  - Prune search space
    - Lower bounding distance



Indexing with *i*SAX (cont.)

- MinDist function for query time series *T*
  - Let T_{PAA} be the PAA representation of time series *T*, S_{iSAX} be the *i*SAX representation of time series *S*
  - Recall the j^{th} cardinal value of S_{iSAX} derives from a PAA value, *v* between two breakpoints β_L, β_U, $\beta_L < v \leq \beta_U$, $1 \leq j \leq w$

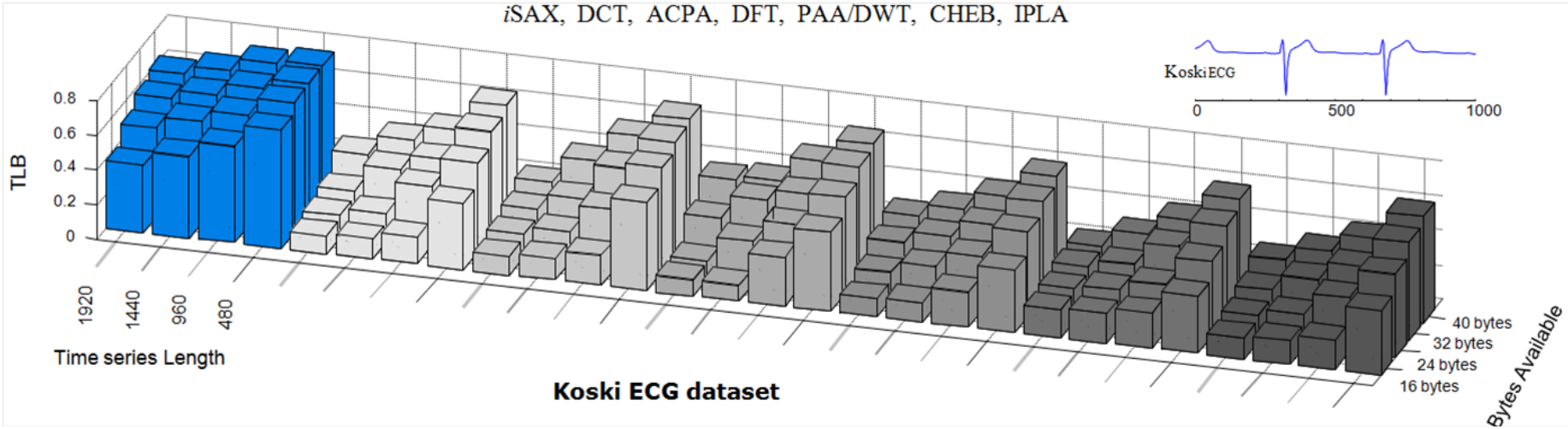
$$MINDIST\_PAA\_iSAX(T_{PAA}, S_{iSAX}) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} \begin{cases} (\beta_{Li} - T_{PAAi})^2 \; if \; \beta_{Li} > T_{PAAi} \\ (\beta_{Ui} - T_{PAAi})^2 \; if \; \beta_{Ui} < T_{PAAi} \\ 0 \quad otherwise \end{cases}}$$



Tightness of Lower Bounds

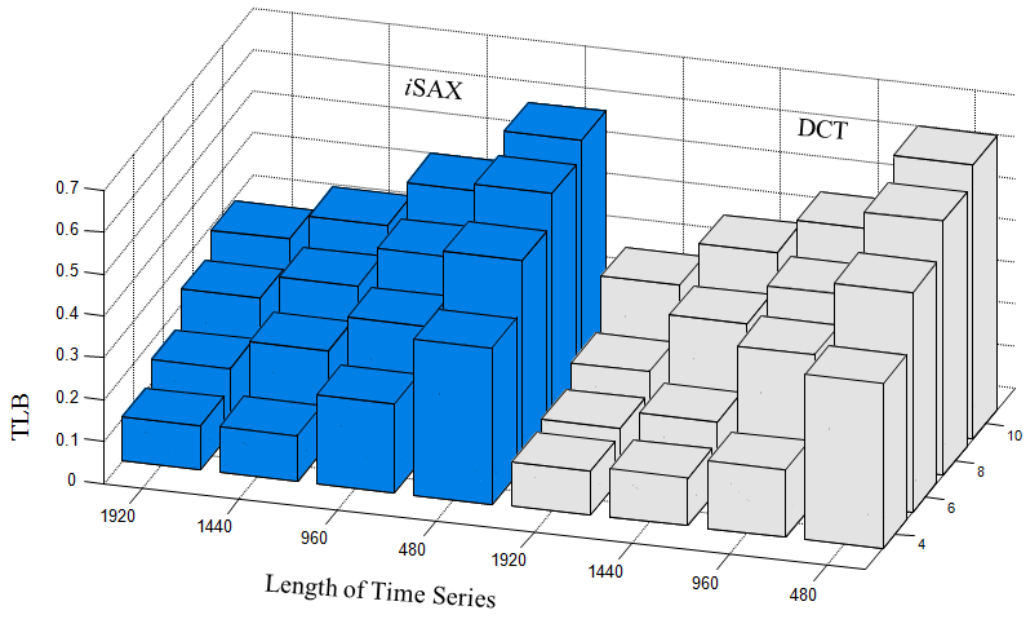
$$TLB = \frac{LowerBoundDist(T', S')}{EuclideanDist(T, S)}$$

- For a given dataset
  - Time series length [**480, 960, 1440, 1920**]
  - Bytes available for representation [**16, 24, 32, 40**]
  - Results similar across thirty datasets



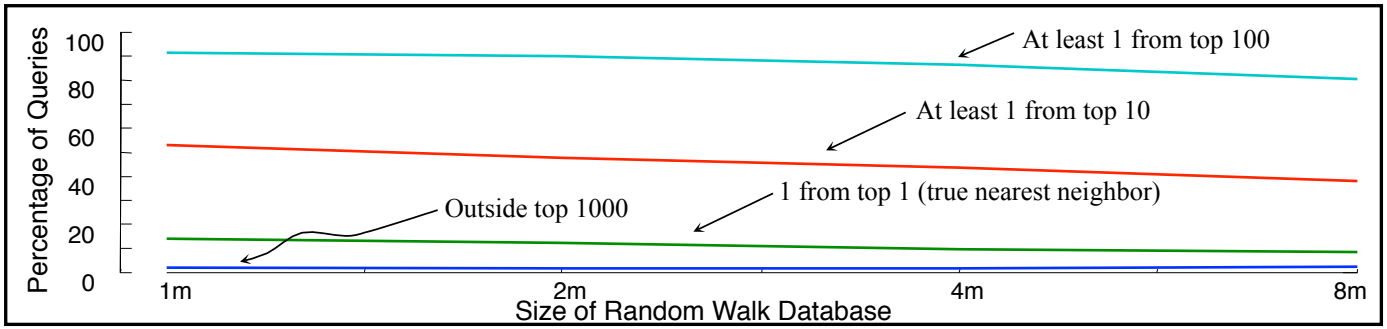
Tightness of Lower Bounds (cont.)

- Competitive even if naïvely encoded to precision of real-valued counterparts



Indexing Performance

- Indexed random walk datasets of [**1, 2, 4, 8**] million time series of length 256 (b = 4, w = 8, th = 100)
- Approximate Search (1000 queries):



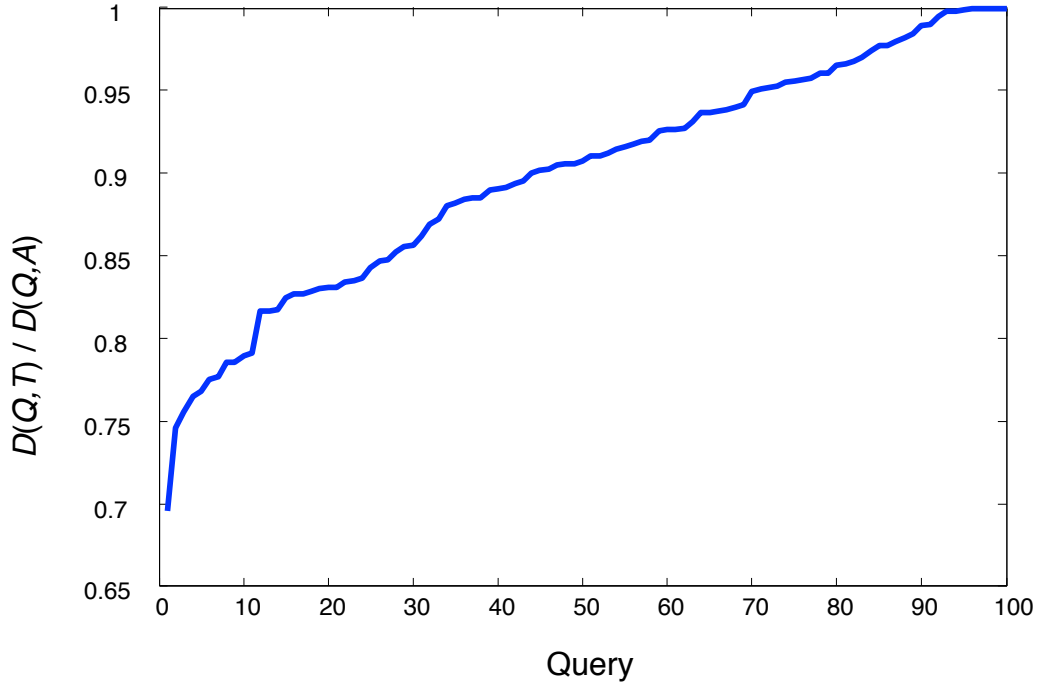
- Exact Search (100 queries):

	Avg. Time/Query (min)			
	1M	**2M**	**4M**	**8M**
Exact Search	3.8	5.8	9.0	14.1
Sequential Scan	71.5	104.8	168.8	297.6

	Avg. Disk Accesses/Query			
	1M	**2M**	**4M**	**8M**
Exact Search	2115.3	3172.5	4925.3	7719.1
Sequential Scan	39255	57365	92209	162340

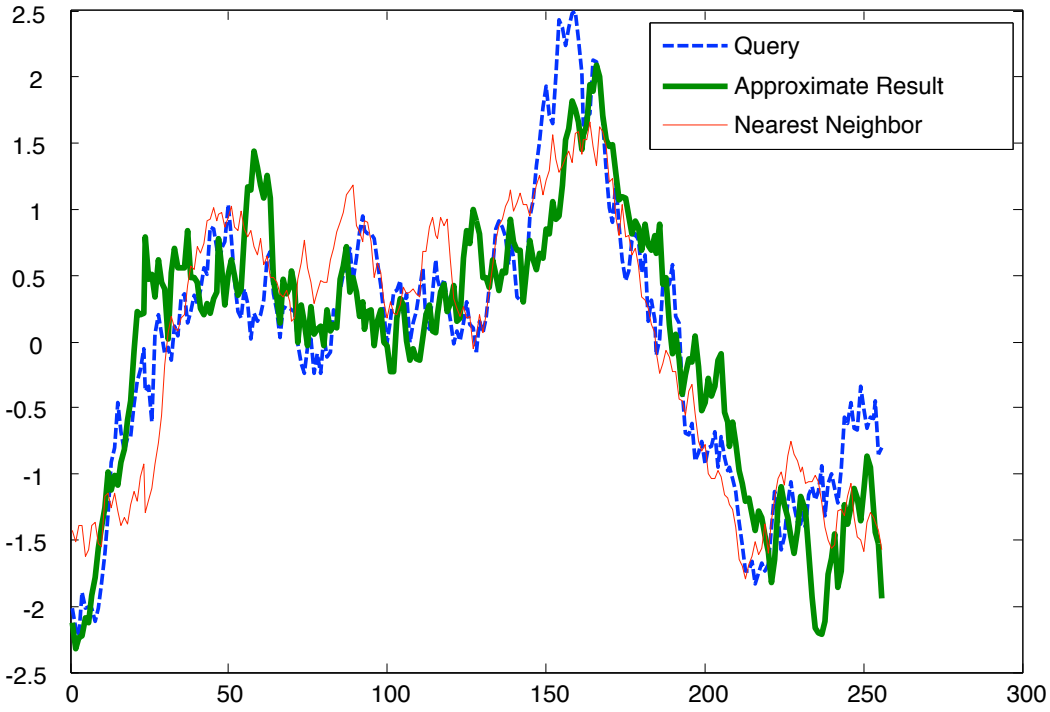
Approximate Search Quality

- To evaluate the quality of approximate search
  - Indexed ~10M time series of length 256
  - 100 random queries
  - Given: Query *Q*, True Nearest Neighbor *T*, Approximate Result *A*
  - *Distance Ratio = EuclideanDist(Q,T) / EuclideanDist(Q,A)*

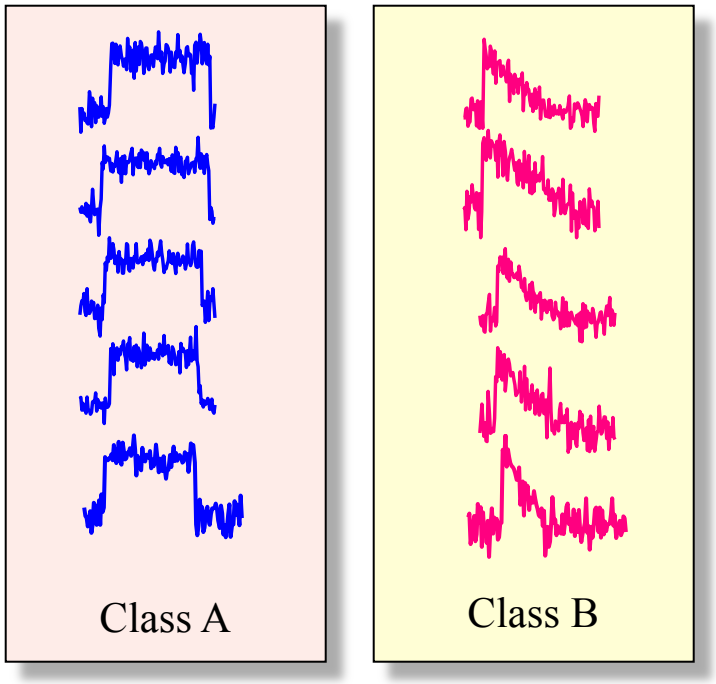


Approximate Search Quality (cont.)

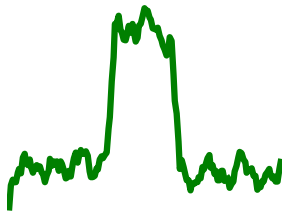
- Visually examine the lower median of distance ratios (0.907)



Classification in Time Series



Which class does

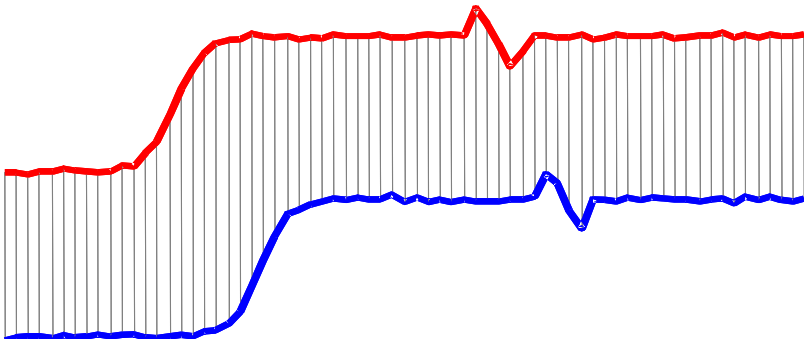


belong to?

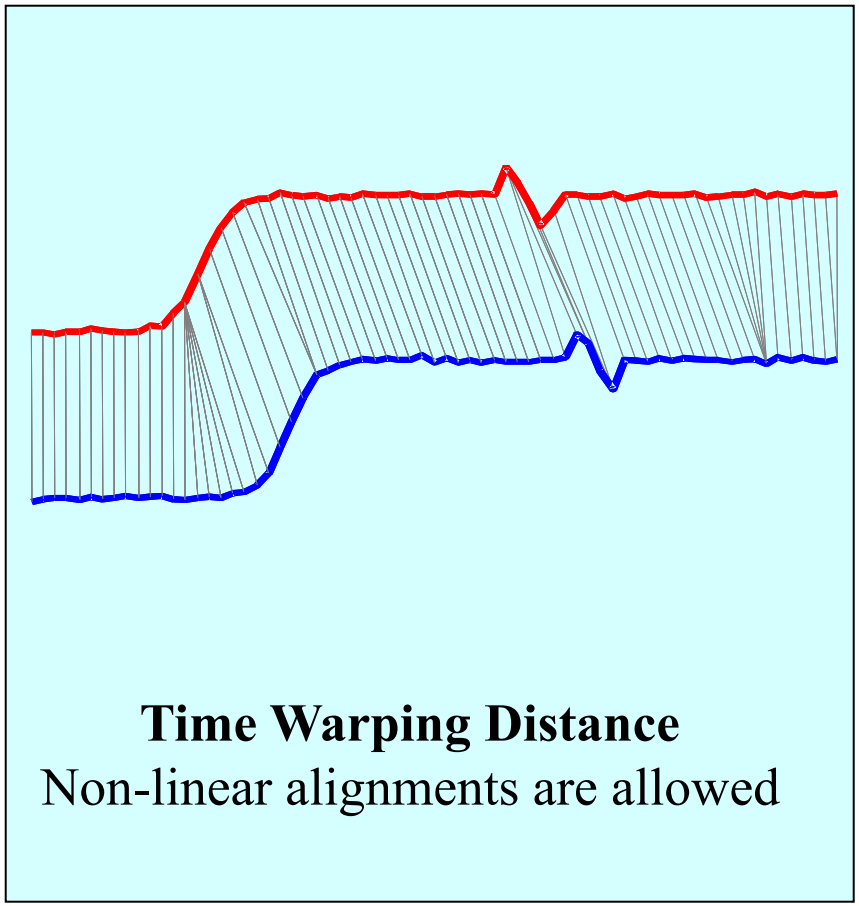
Classification in Time Series

- 1-Nearest Neighbor classification is one of the most common
- It's frequently used to compare the quality of time series representations or distance measures

Dynamic Time Warping (DTW)

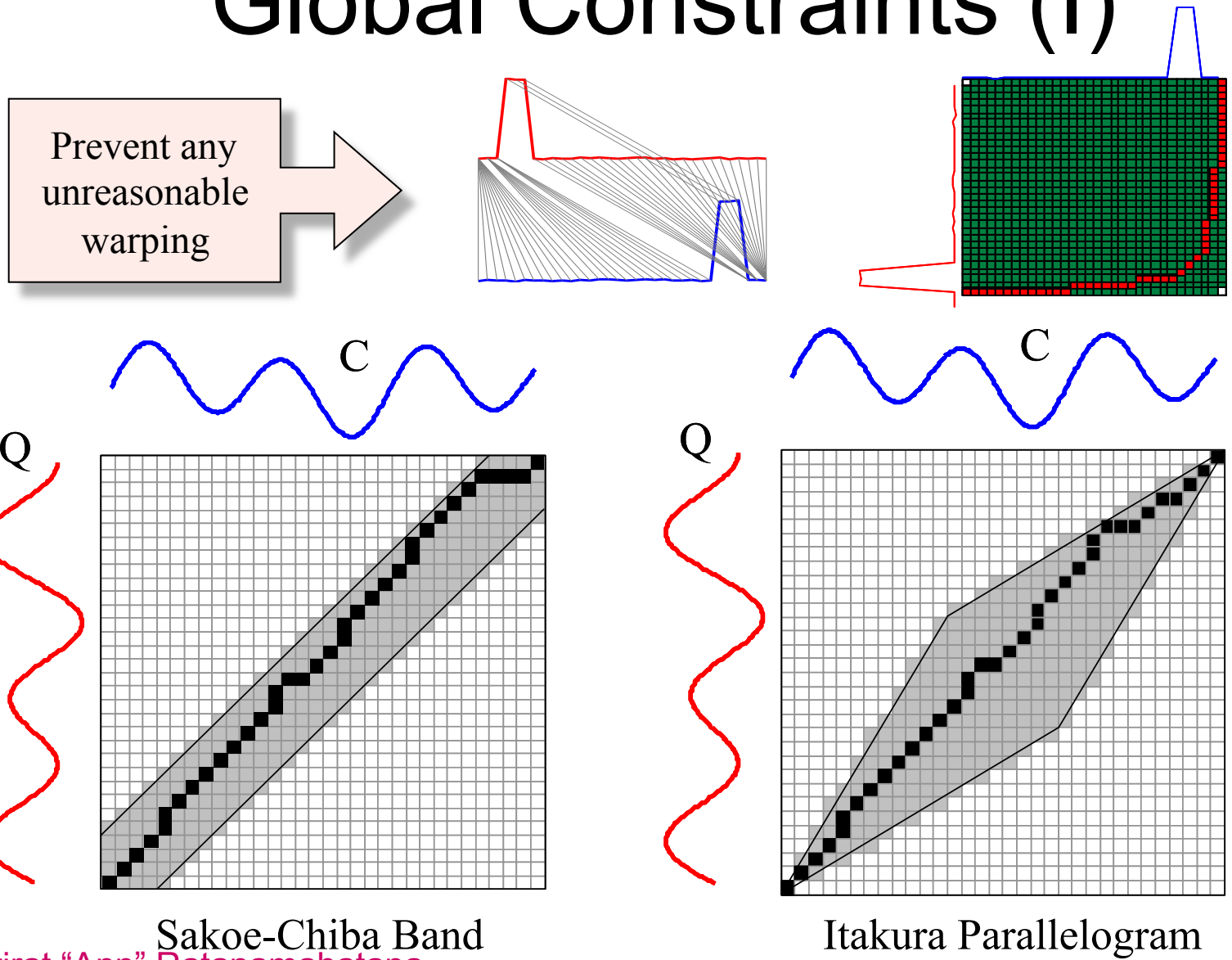


Euclidean Distance
One-to-one alignments



Time Warping Distance
Non-linear alignments are allowed

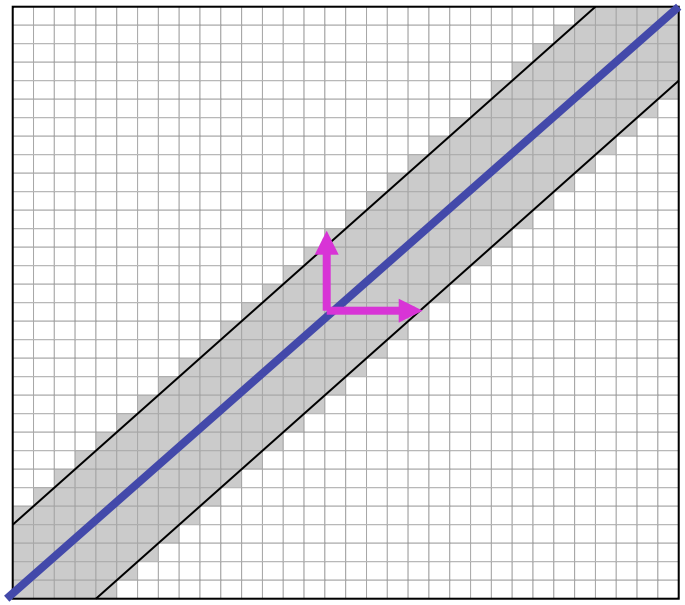
Global Constraints (I)



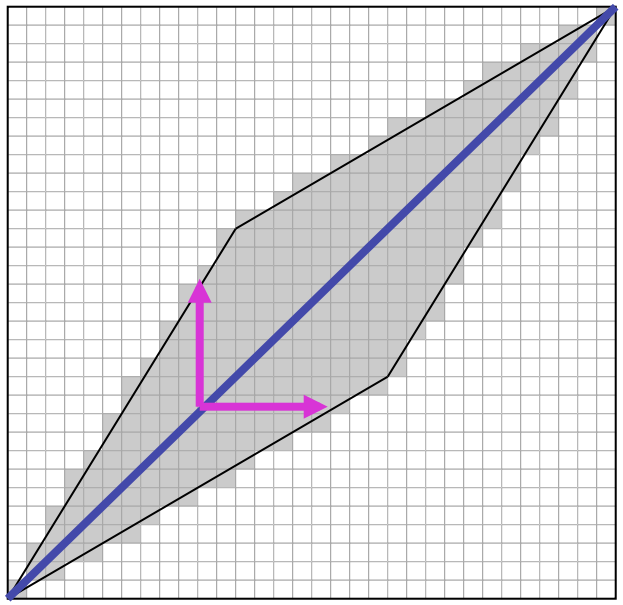
Global Constraints (II)

A Global Constraint for a sequence of size m is defined by R, where
$R_i = d \qquad 0 \le d \le m,\ 1 \le i \le m.$

R_i defines a freedom of warping above and to the right of the diagonal at any given point i in the sequence.



Sakoe-Chiba Band



Itakura Parallelogram

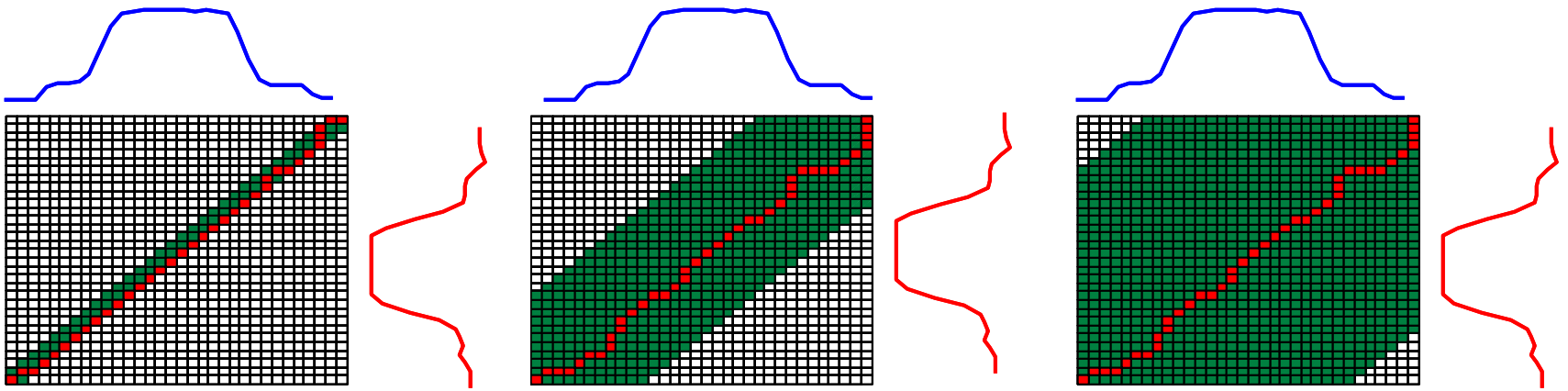
Is Wider the Band, the Better?

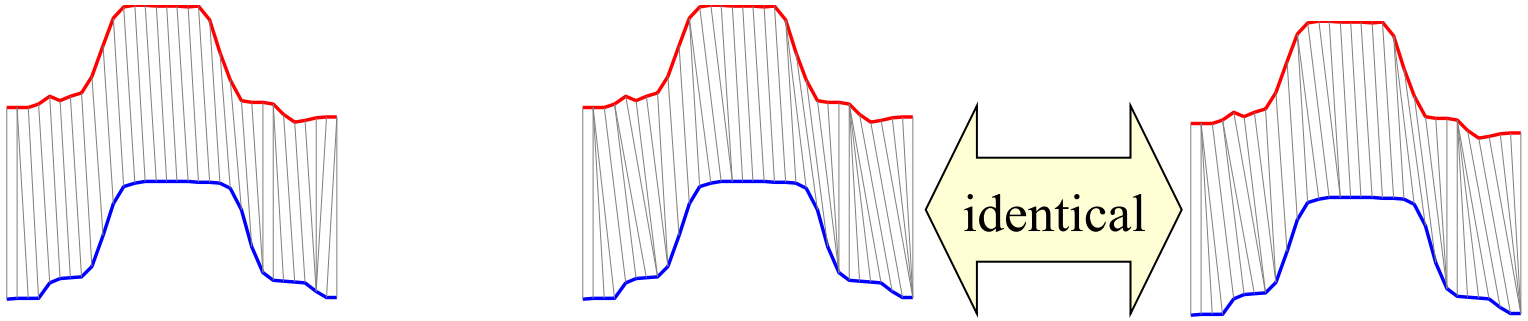
Euclidean distance = 2.4836

DTW dist = 1.6389
R = 1

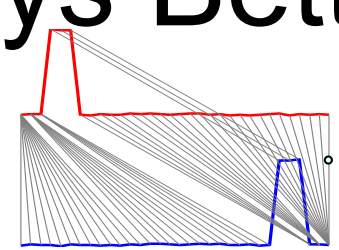
DTW dist = 1.0204
R = 10

DTW dist = 1.0204
R = 25



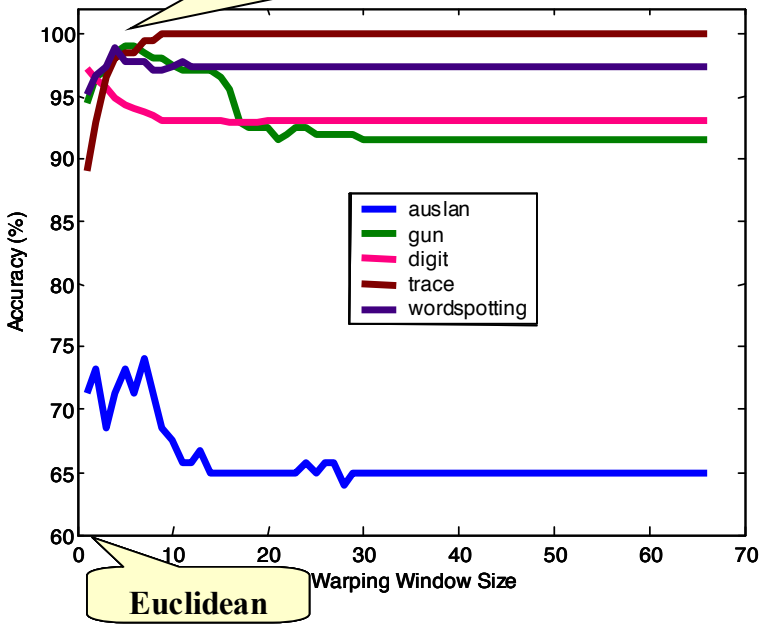


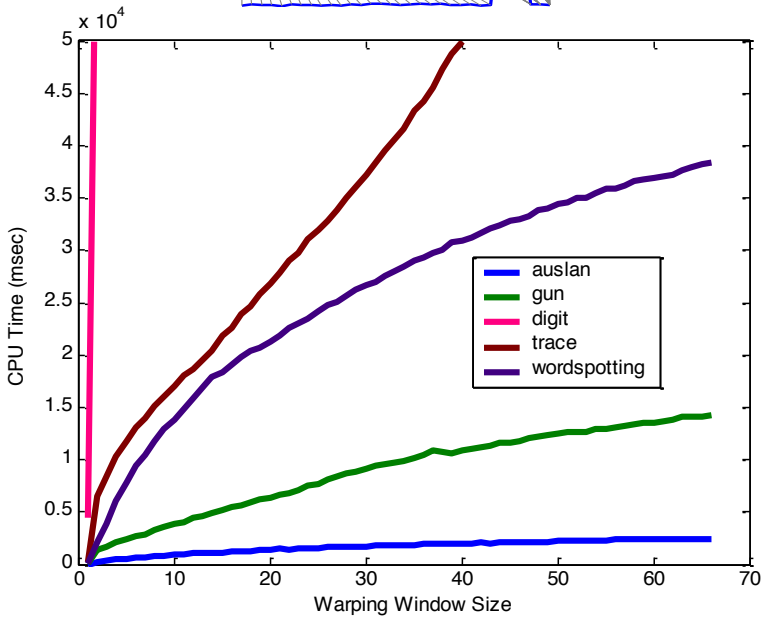
Wider Isn't Always Better

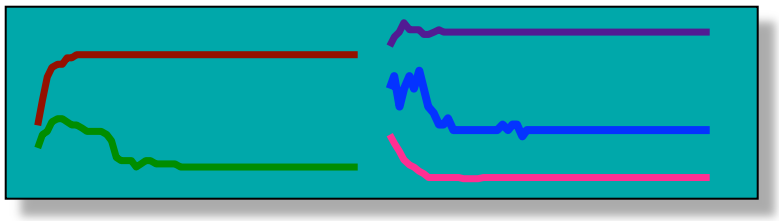


Recall this example

Most accuracies peak at smaller window size



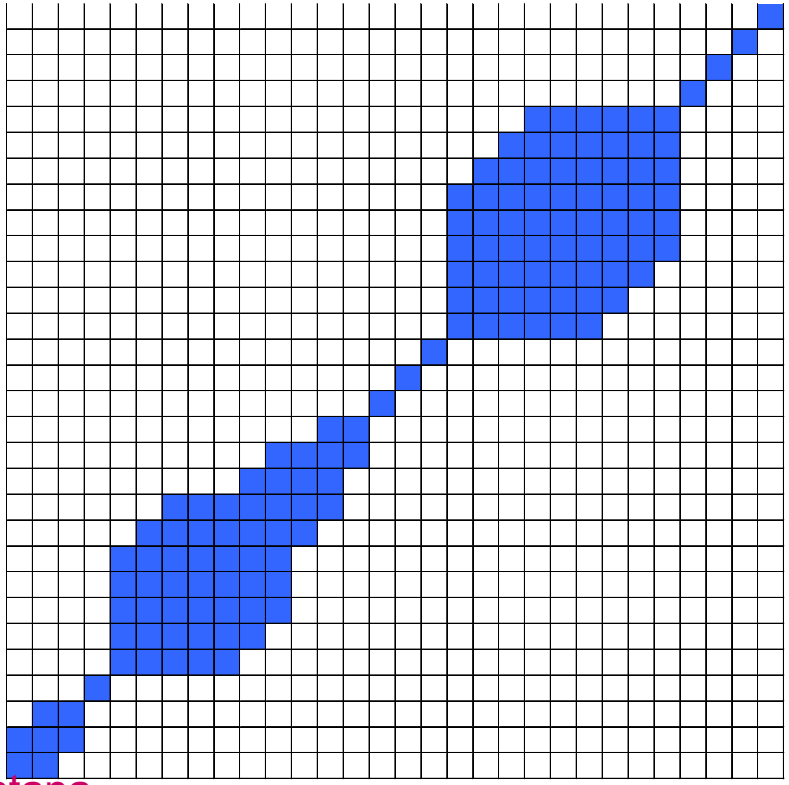




Larger warping window is not always a good thing.

Ratanamahatana-Keogh Band (*R-K Band*)

Solution: Create an arbitrary shape and size of the band that is appropriate for the data we want to classify.



How Many Bands Do We Need?

- Of course, we could use **ONE** same band to classify all the classes, as almost all of the researchers do.

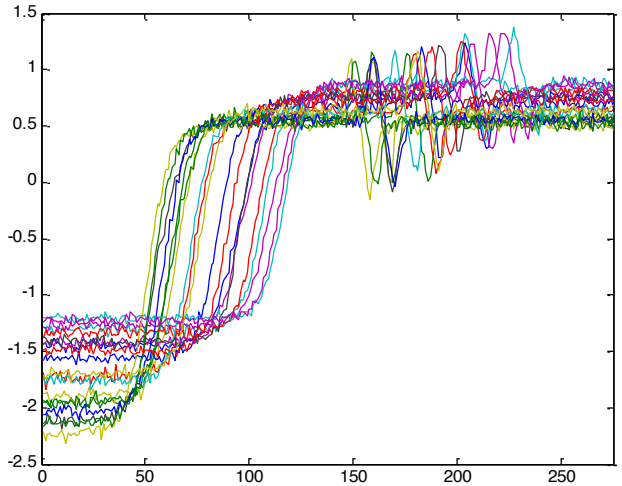
- But…the width of the band does depend on the characteristic of the data within each class. Having one single band for classification is unlikely to generalize.

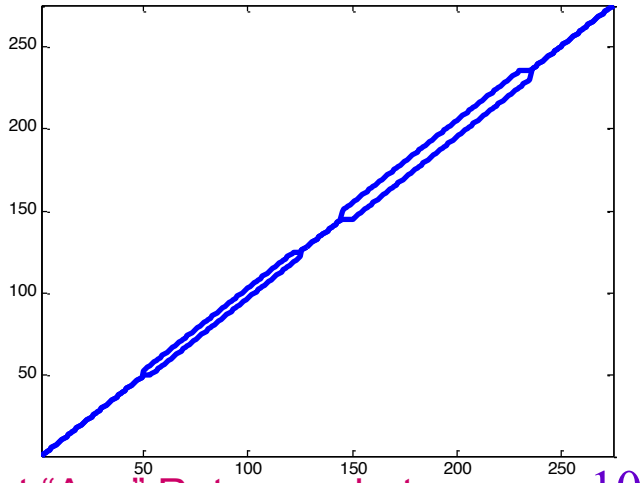
- Proposed solution:
  Create an arbitrary band (*R-K band*) for **each** class and use it accordingly for classification.

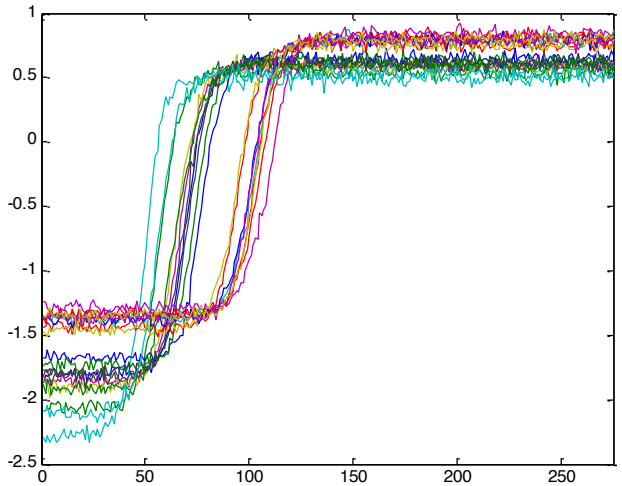
How Do We Create an *R-K Band*?

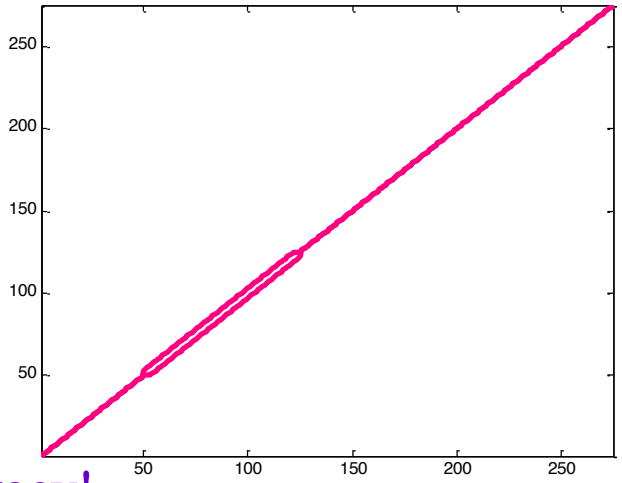
First Attempt: We could look at the data and *manually* create the shape of the bands.

(then we need to adjust the width of each band as well until we get a good result)







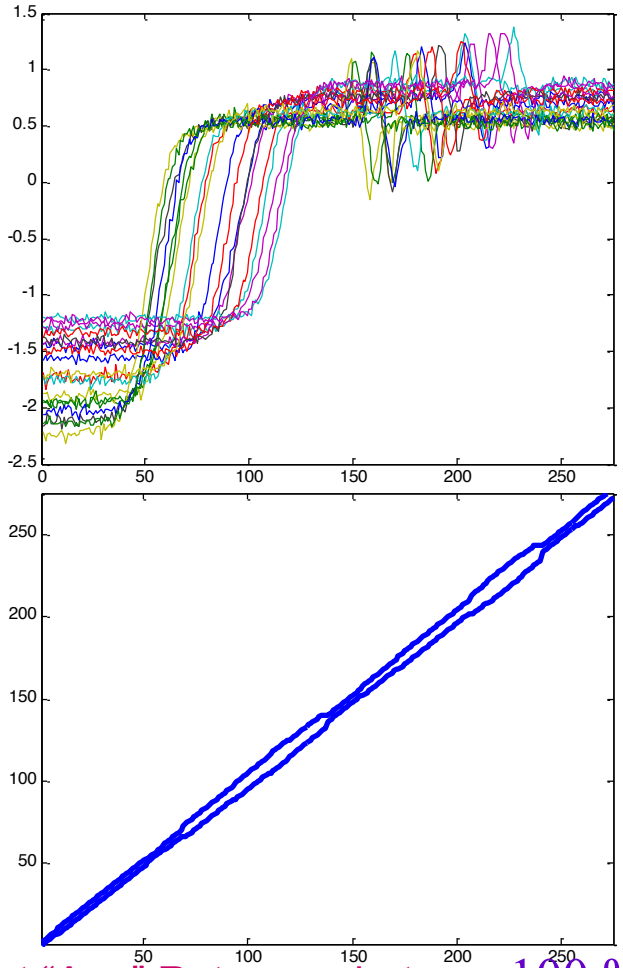


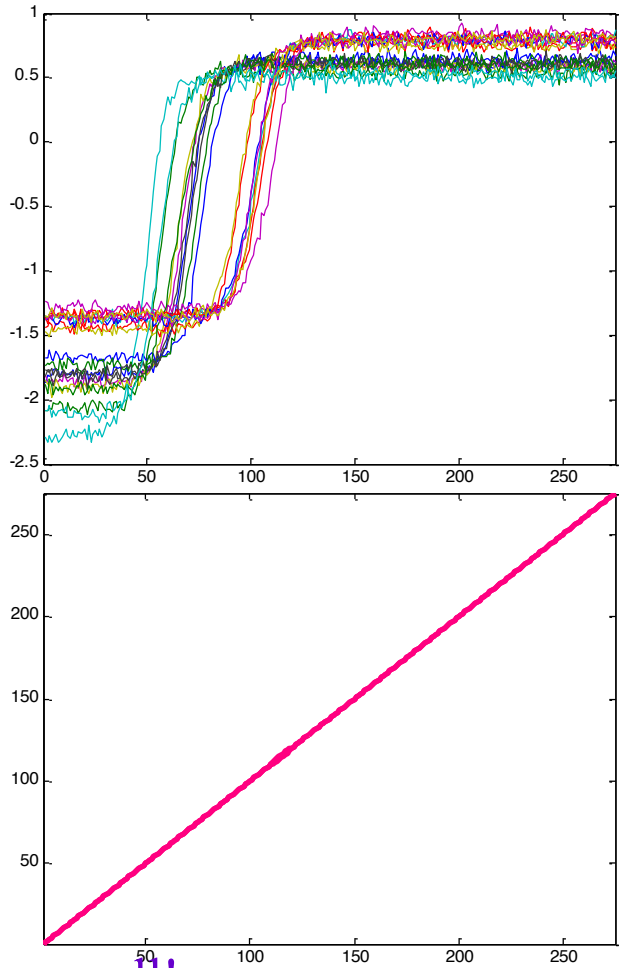
100 % Accuracy!

Eamonn Keogh

Learning an R-K Band Automatically

Our heuristic search algorithm automatically learns the bands from the data.
(sometimes, we can even get an unintuitive shape that give a good result.)

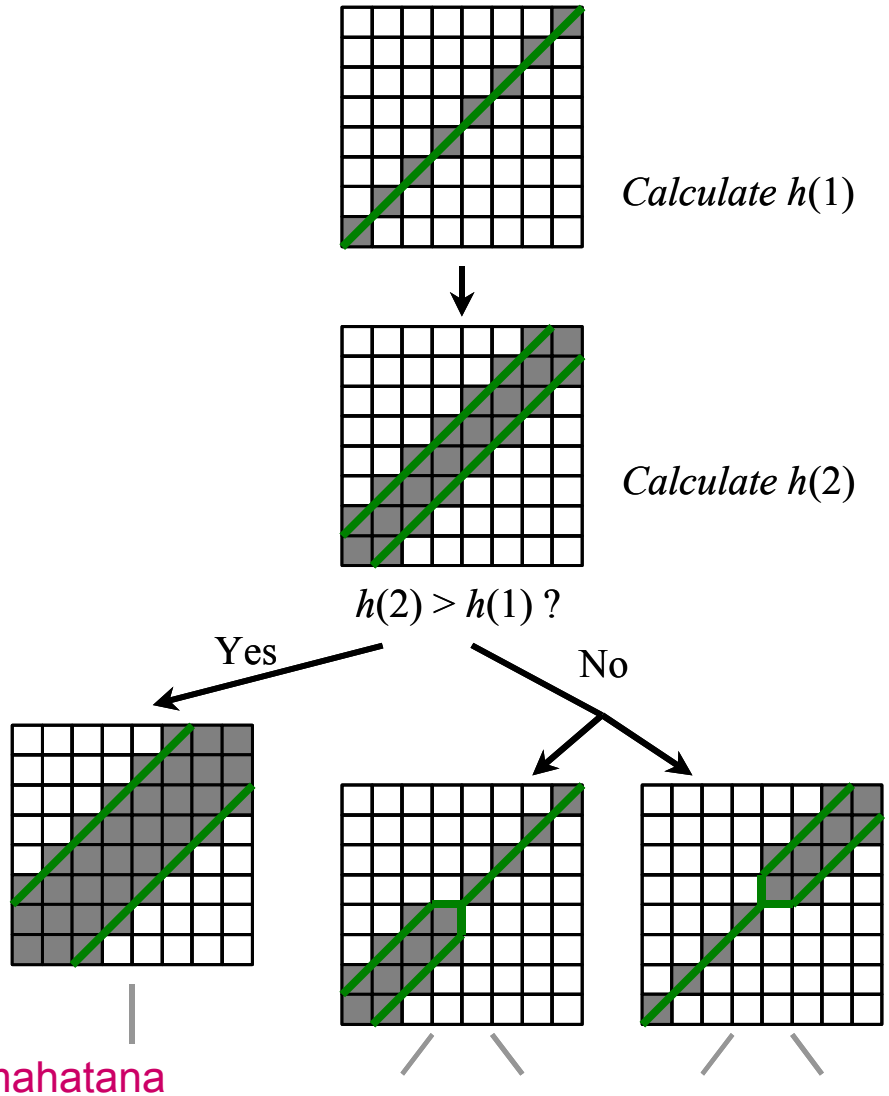




100 % Accuracy as well!

Eamonn Keogh

R-K Band Learning With Heuristic Search



Clustering on Time Series

- Similarly, a lot of work are about:
  - Choosing the right representation, and/or
  - Choosing the right distance measure
  - Then use existing clustering algorithms such as k-means or hierarchical clustering algorithms
- Some ad-hoc time series clustering algorithms have been proposed

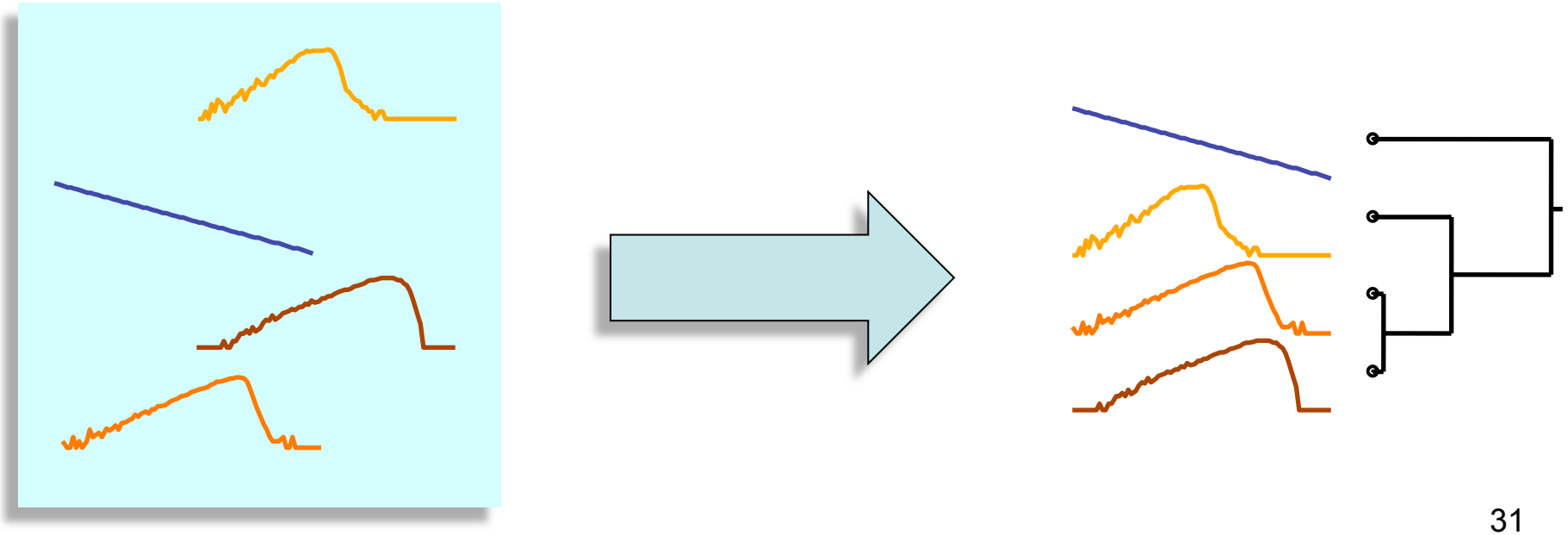
Time Series Clustering

Whole Clustering: The notion of clustering here is similar to that of conventional clustering of discrete objects. Given a set of individual time series data, the objective is to group similar time series into the same cluster.

Subsequence Clustering: Given a single time series, individual time series (subsequences) are extracted with a sliding window. Clustering is then performed on the extracted time series.

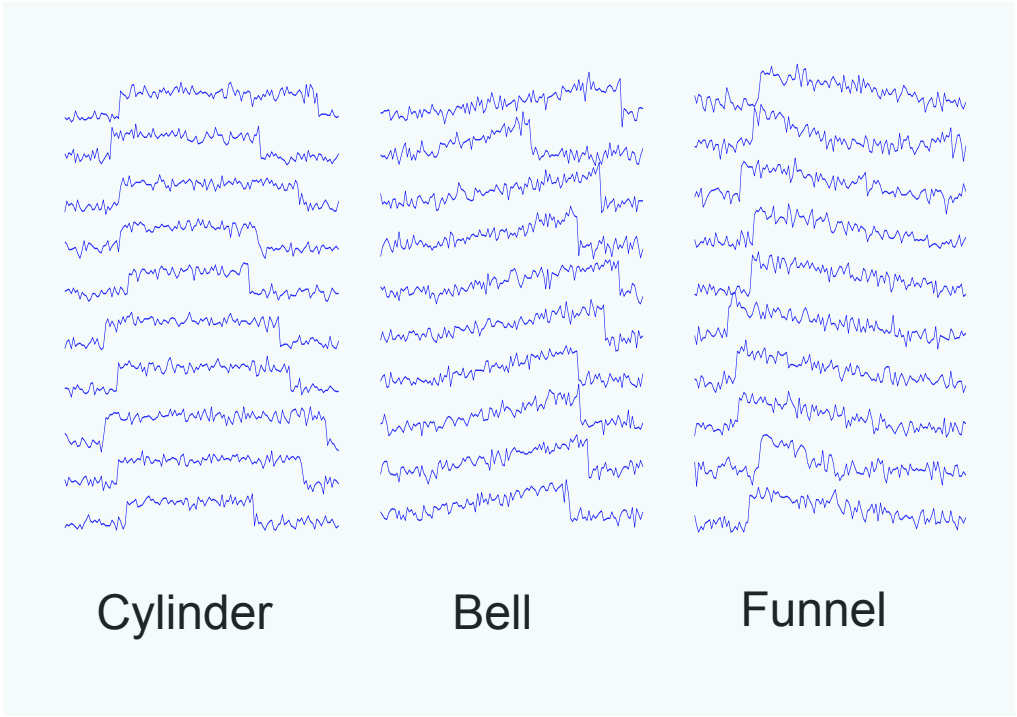
Whole Clustering

Whole Clustering: The notion of clustering here is similar to that of conventional clustering of discrete objects. Given a set of individual time series data, the objective is to group similar time series into the same cluster.

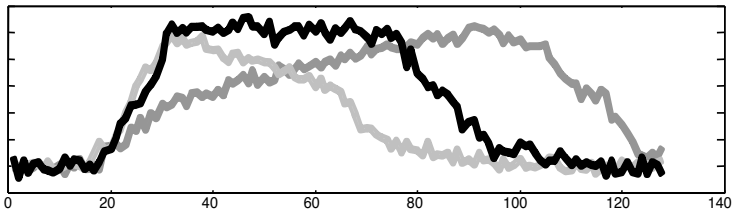


Whole Clustering

Clustering Data:



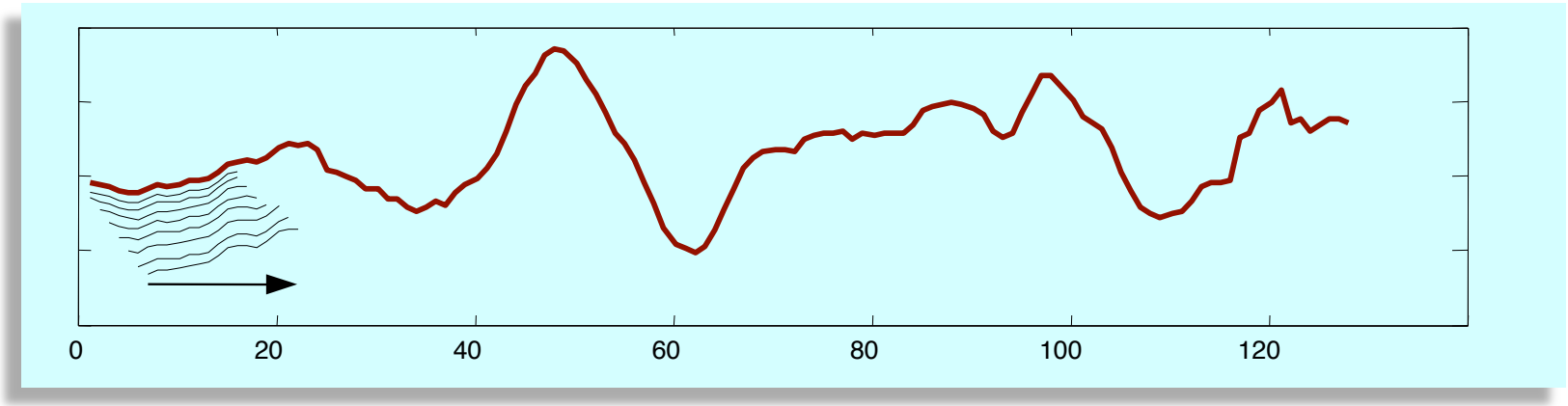
Clustering Algorithm



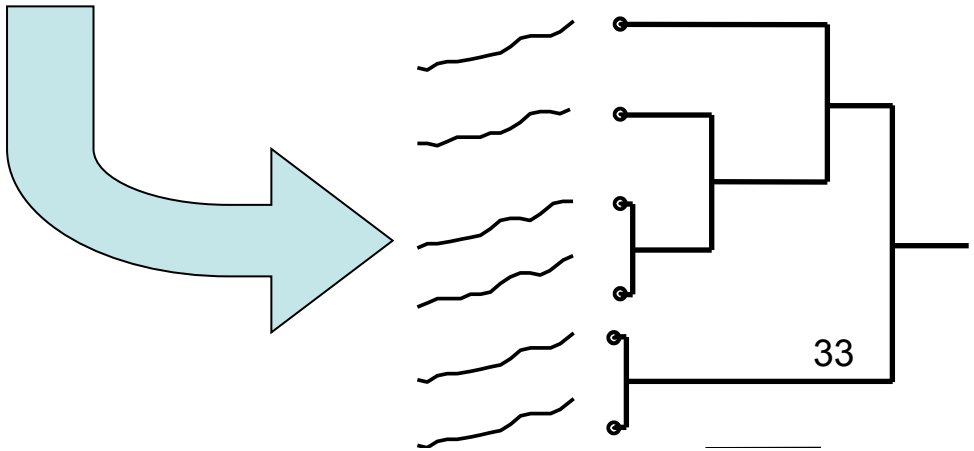
Final centers found by K-Means

Subsequence Clustering (STS)

Subsequence Clustering: Given a single time series, individual time series (subsequences) are extracted with a sliding window. Clustering is then performed on the extracted time series.



Note: There may be other ways to define subsequence clustering, we are making **no** claim about any such definitions.



Why do Subsequence Clustering?

- Finding association rules in time series
- Anomaly detection in time series
- Indexing of time series
- Classifying time series
- Clustering of streaming time series has also been proposed as a knowledge discovery tool in its own right.

However..

Subsequence clustering is meaningless!

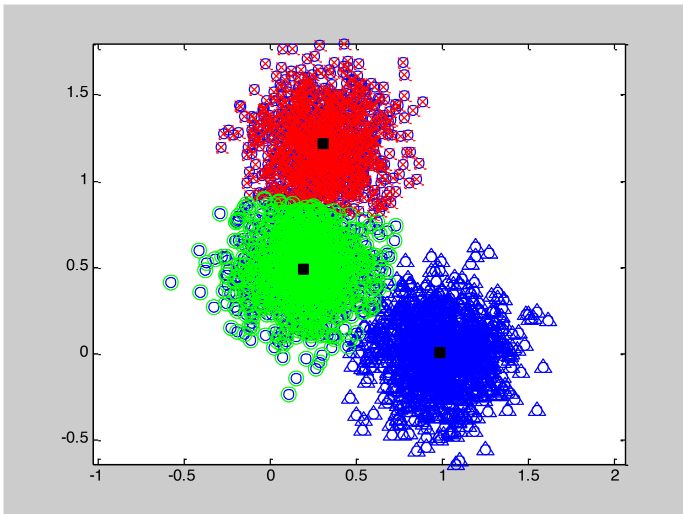
Keogh, E. and Lin, J. 2005. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowl. Inf. Syst. 8, 2 (Aug. 2005), 154-177.*

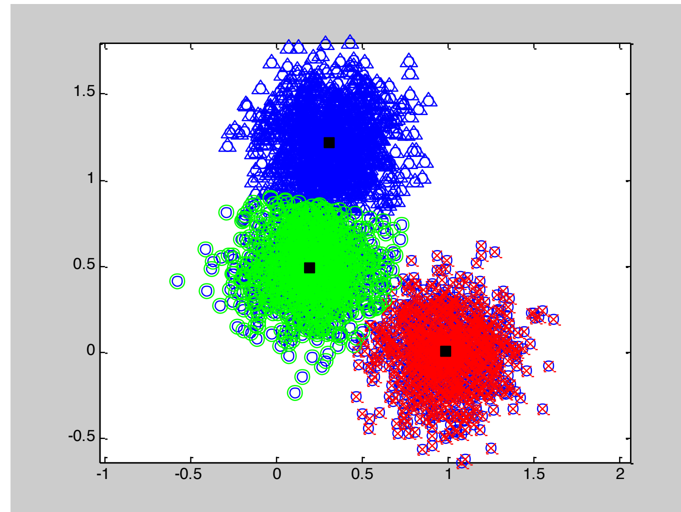
What Does it Mean to be Meaningless?

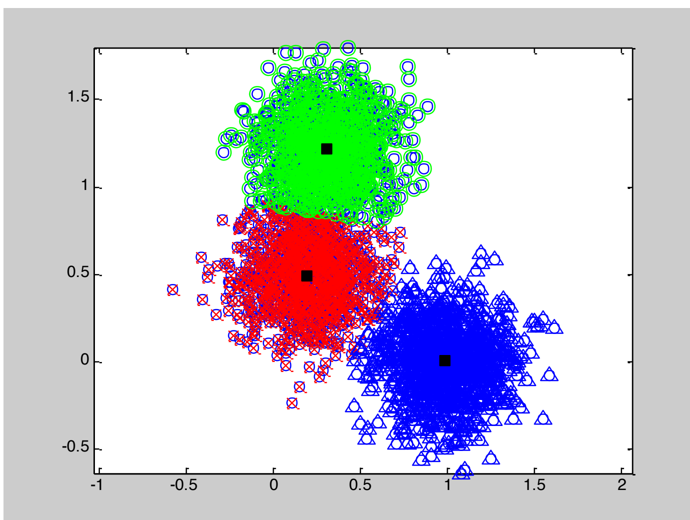
- An algorithm is meaningless if its output is independent of its input.

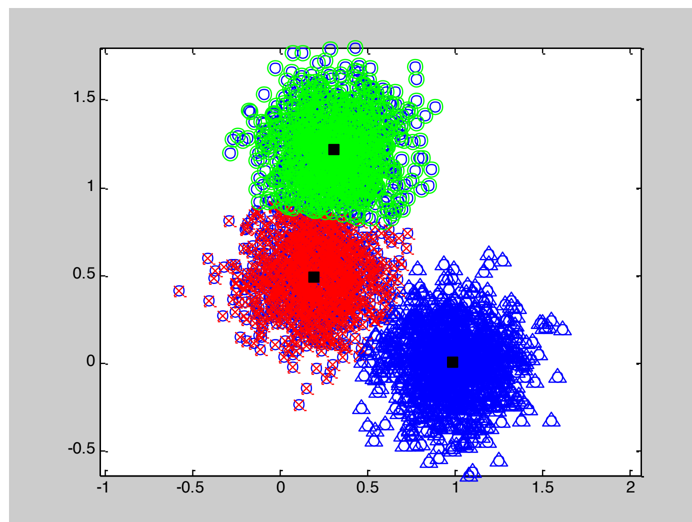
- With the exception of random number generators, meaningless algorithms are useless.

Example of Meaningful Clustering

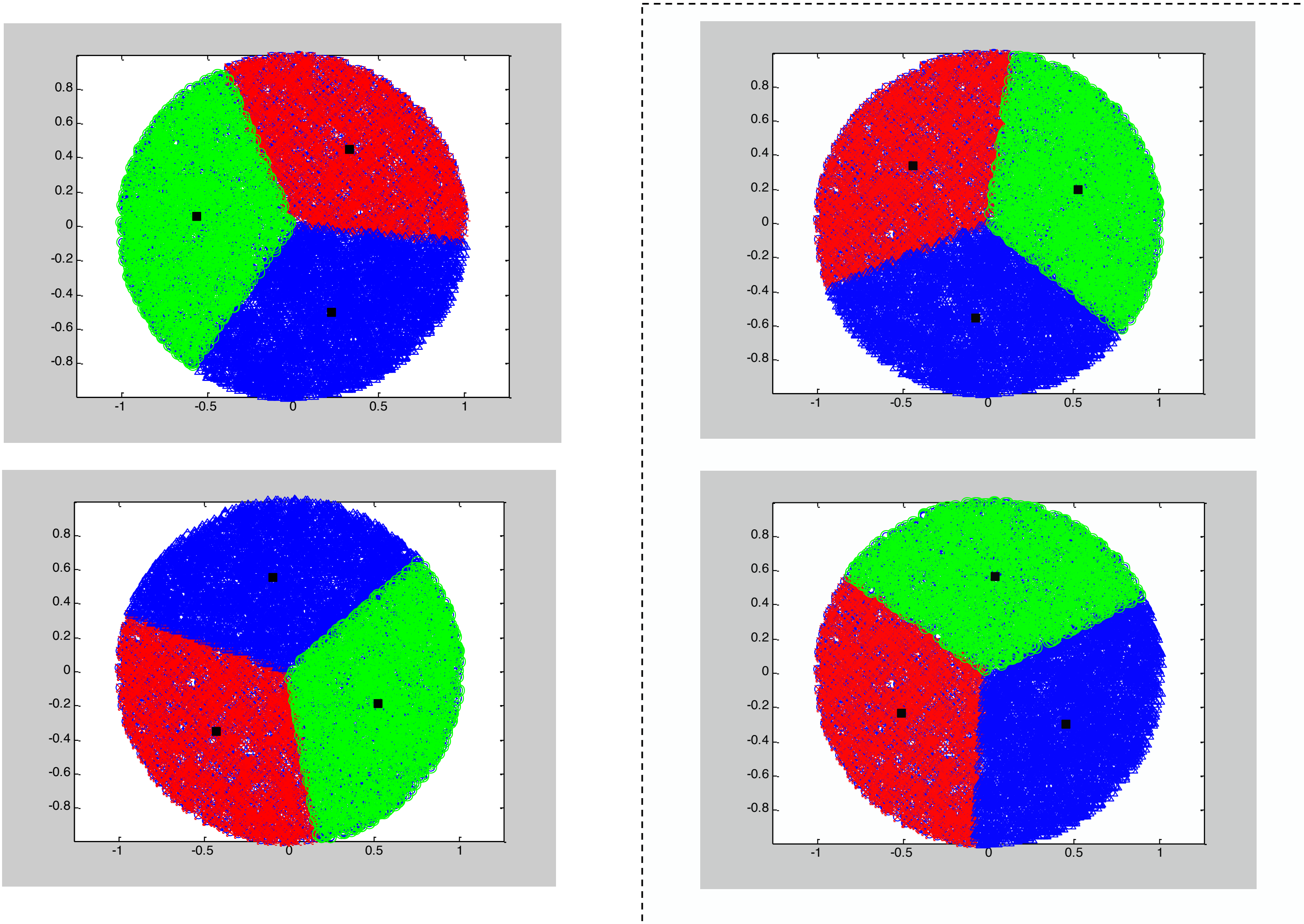




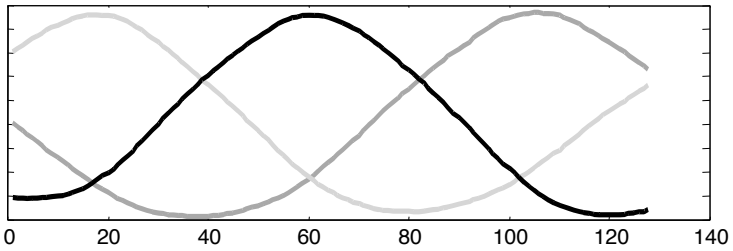




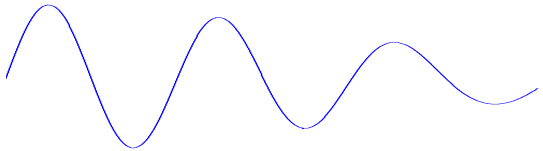
Example of Meaningless Clustering

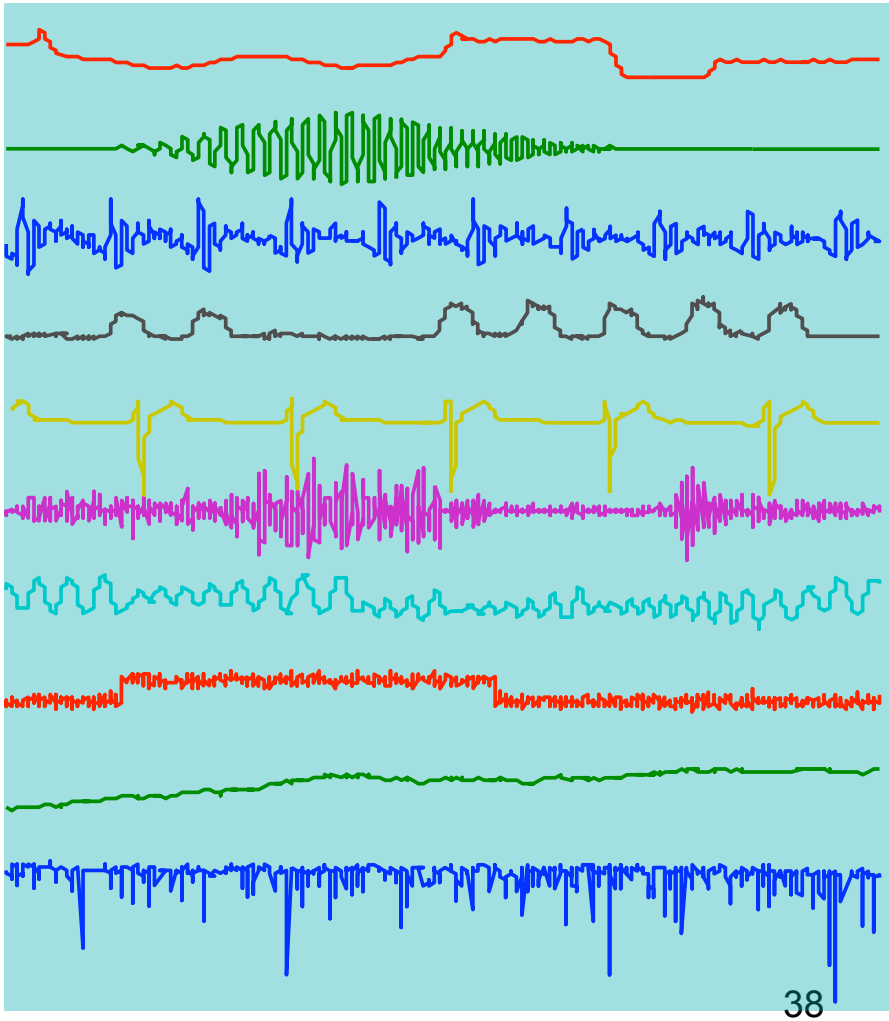


Let's take a look at the cluster centers created by subsequence clustering:

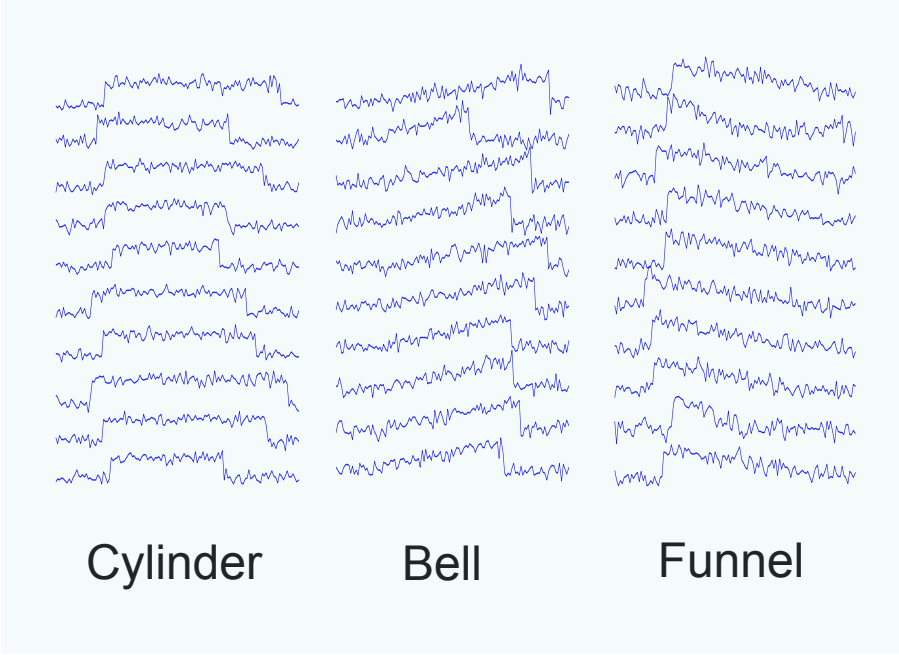


For subsequence clustering, no matter what the input, the output is a set of (out of phase) sine waves!

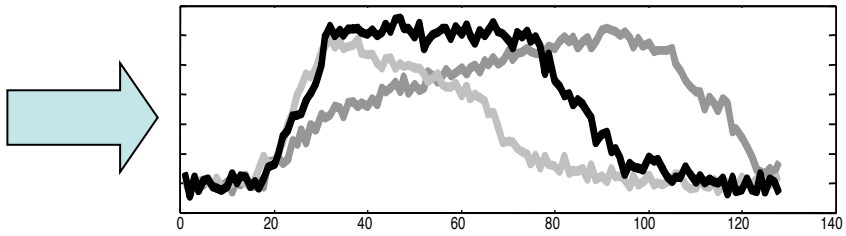




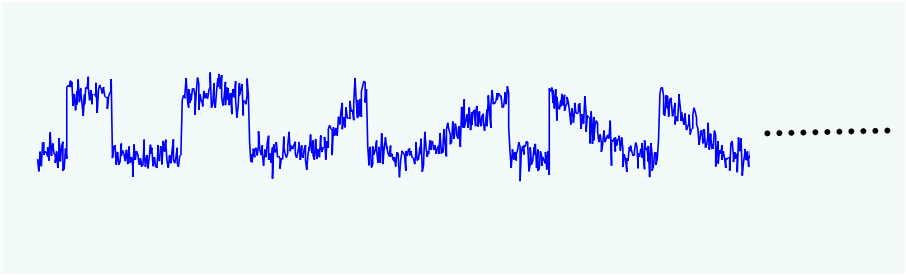
Whole Clustering:

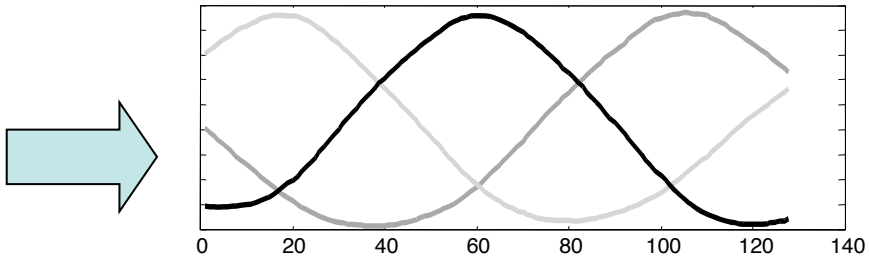


Cluster centers found by K-Means

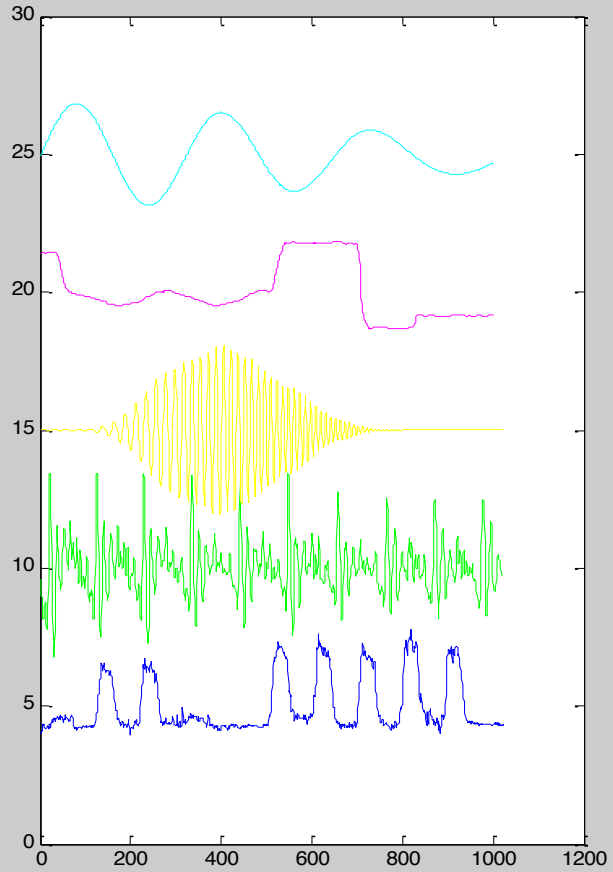


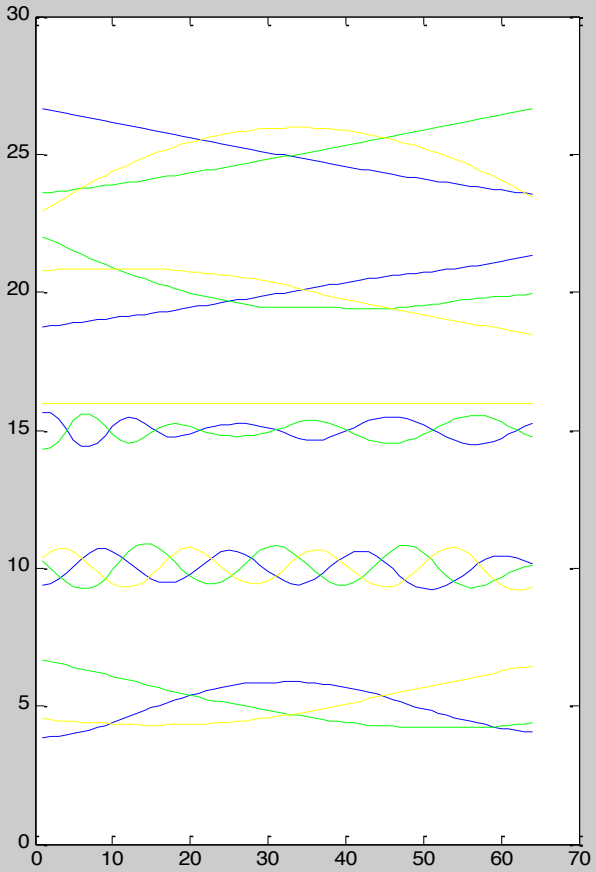
Subsequence Clustering:



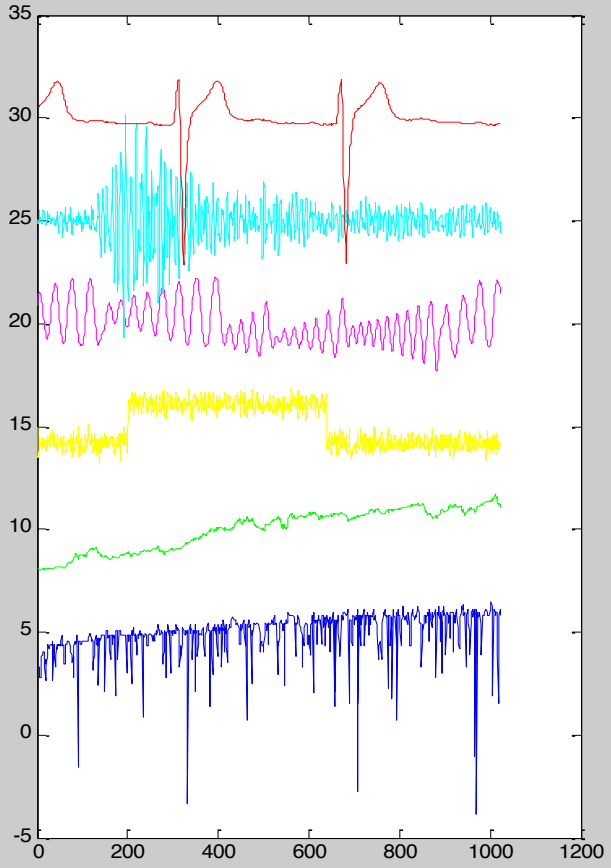


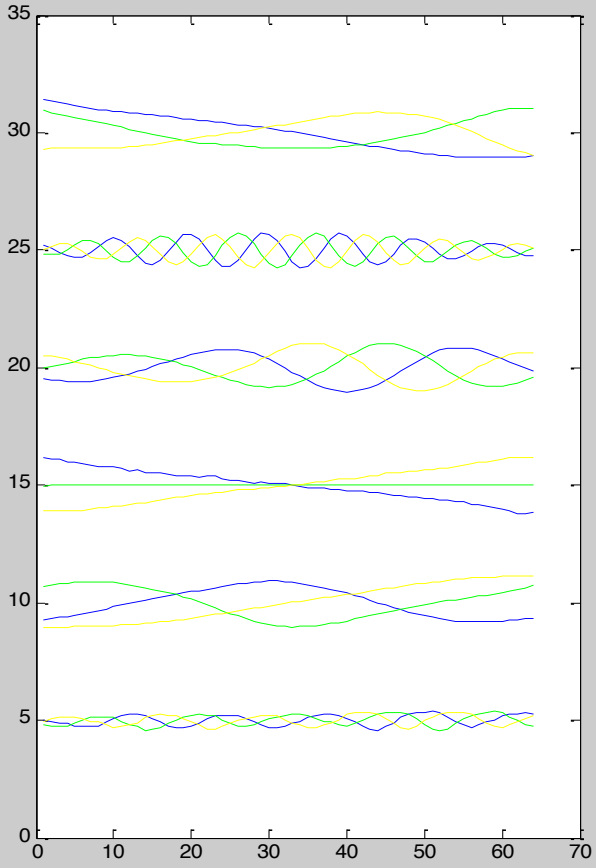
Cluster Centers





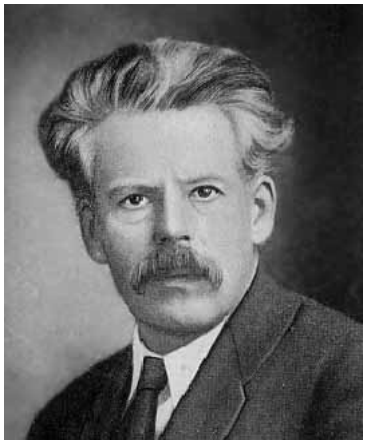
Cluster Centers



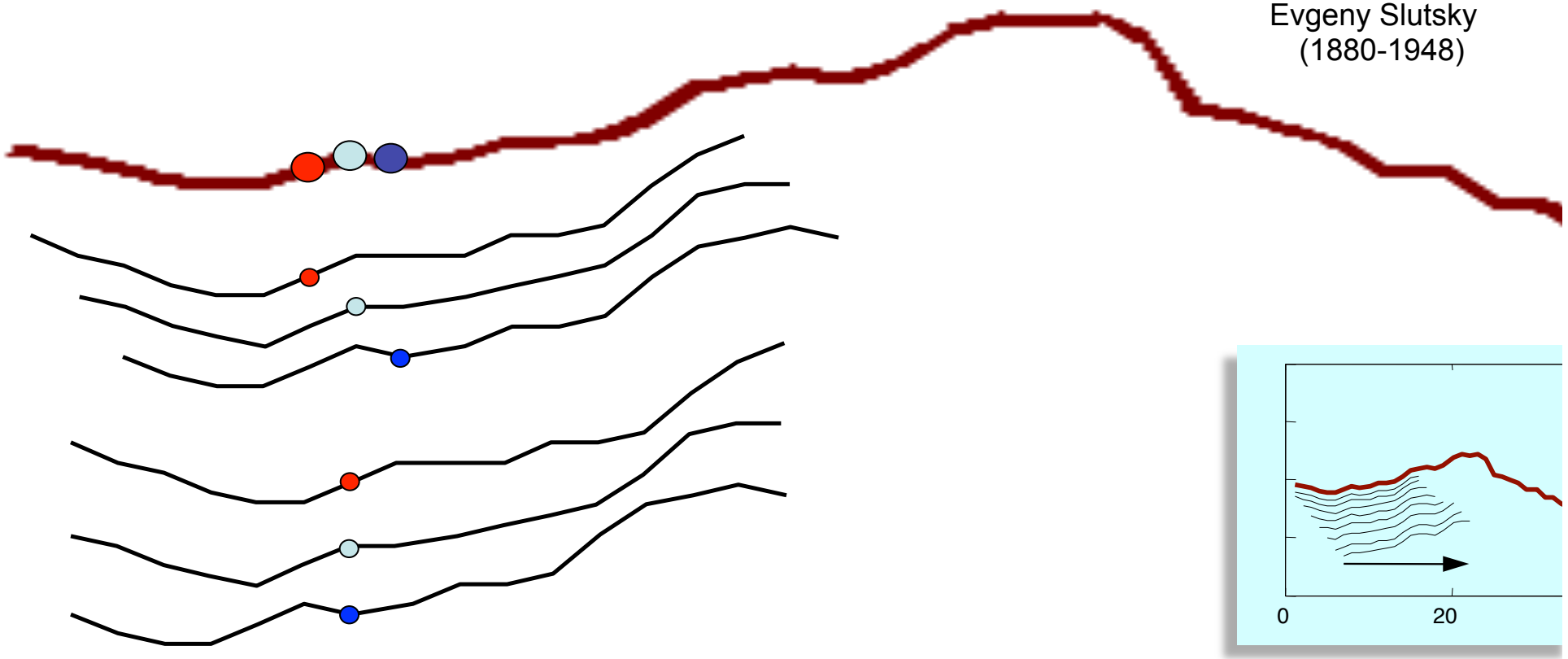


Why Sine Waves?

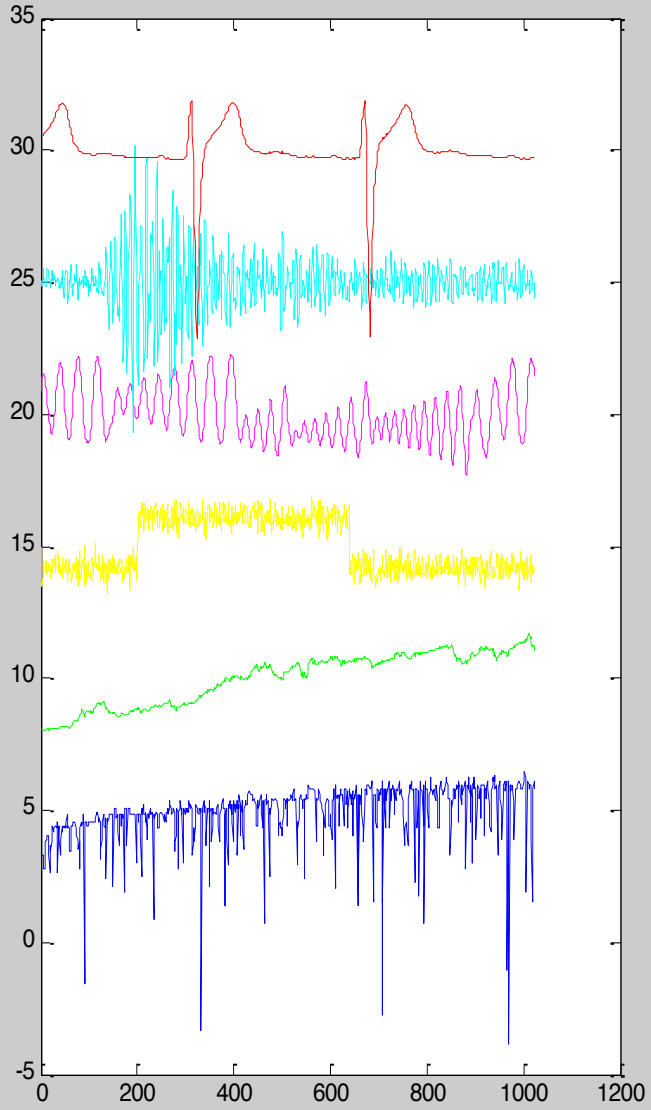
Slutsky's Theorem (informally stated)
Any time series will converge to a sine wave after repeated applications of moving window smoothing

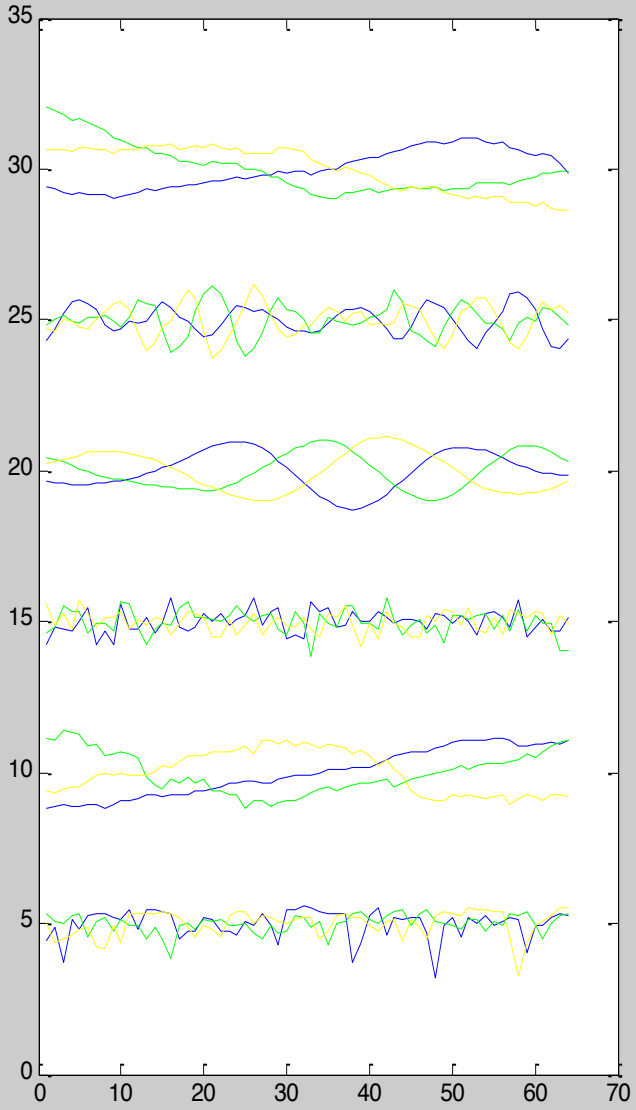


Evgeny Slutsky
(1880-1948)

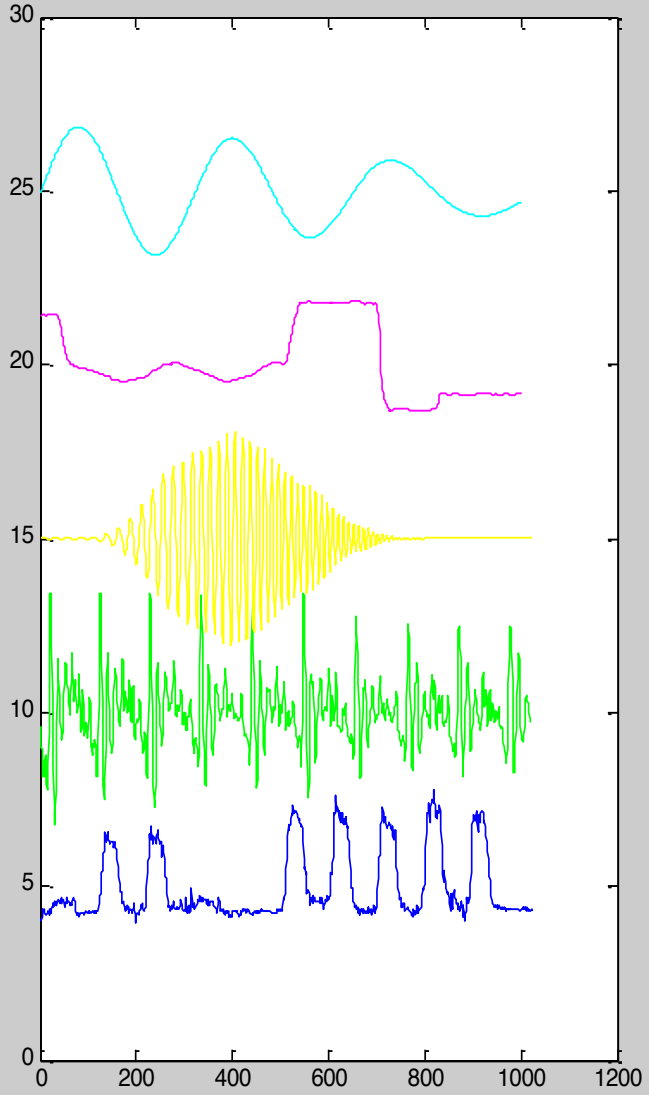


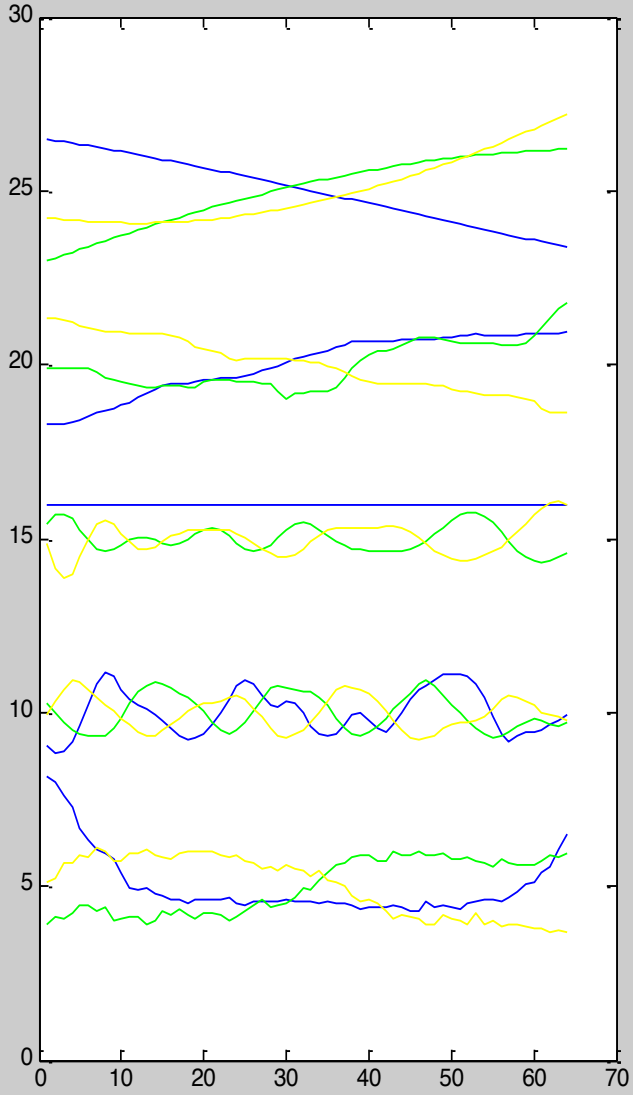
What If We Increase the Step Size?





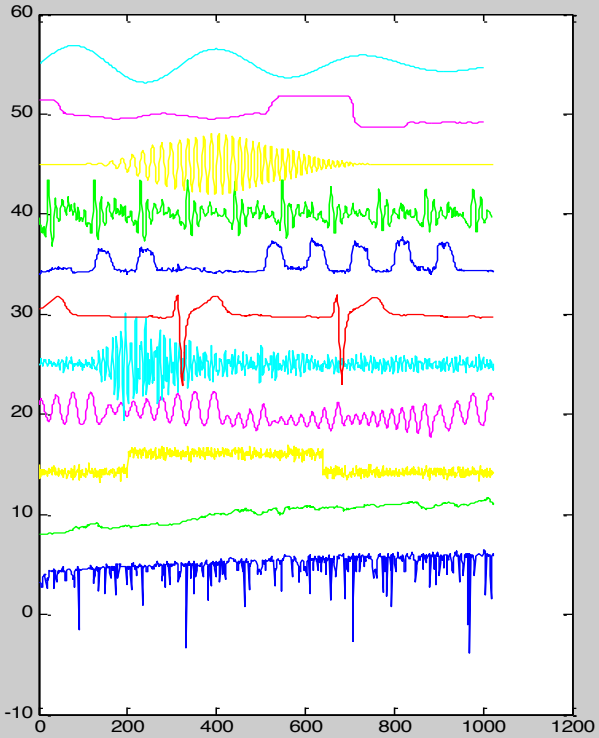
What If We Increase the Step Size?

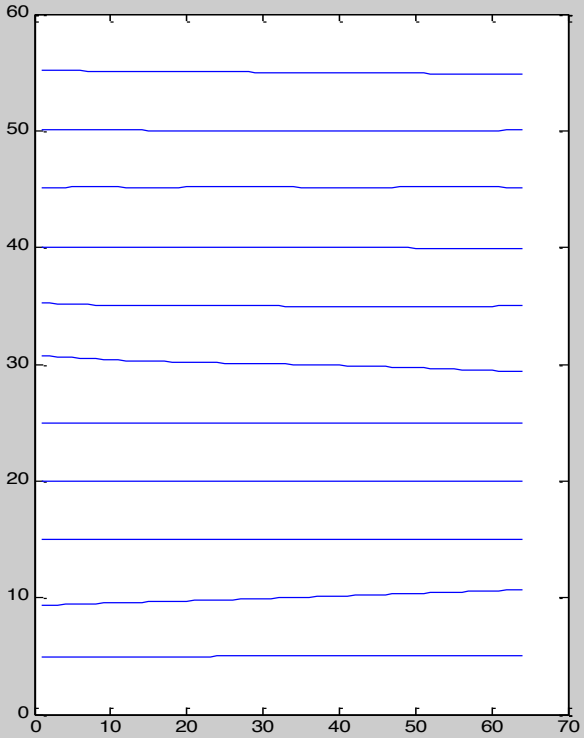




A Hidden Constraint

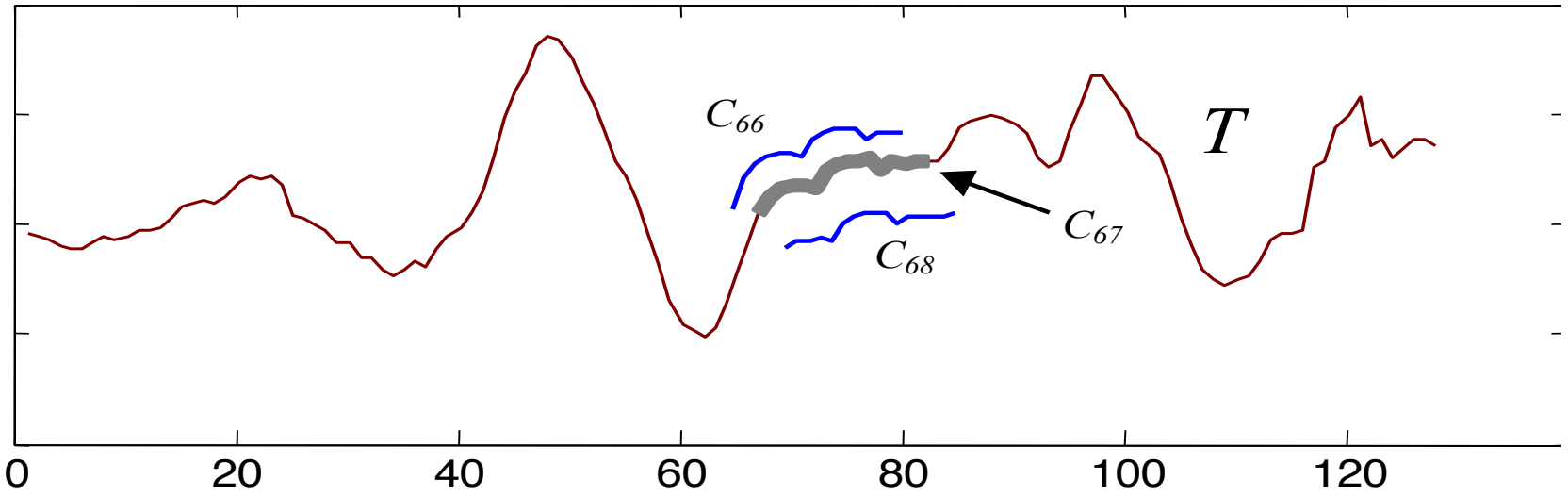
Fact: For any dataset, the weighted (by cluster membership) average of *k* clusters must sum up to the global mean.





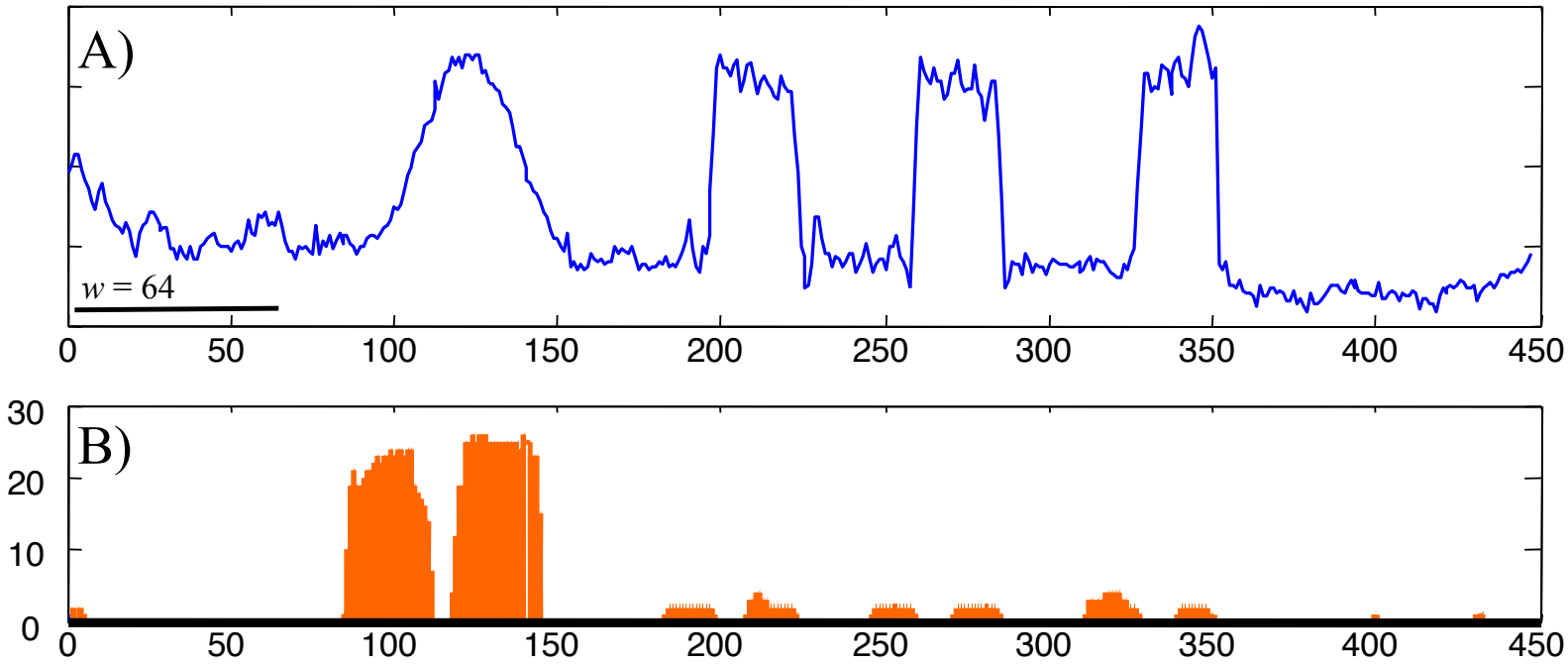
Trivial Matches

- *Trivial Match*: Given a subsequence *C* beginning at position *p*, a matching subsequence *M* beginning at *q*, and a distance *R*, we say that *M* is a *trivial match* to *C* of order *R*, if either *p* = *q* or there does not exist a subsequence *M'* beginning at *q'* such that *D*(*C*, *M'*) > *R*, and either *q* < *q'* < *p* or *p* < *q'* < *q*.



Trivial Matches

- Different subsequences have different numbers of trivial matches



Necessary Conditions

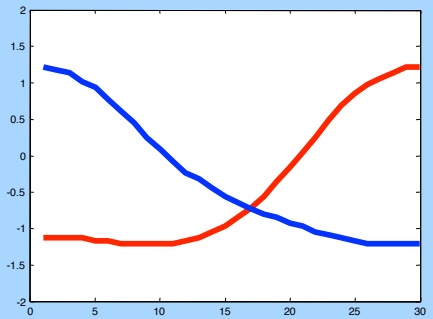
- For a STS clustering algorithm to discover k patterns:
  - The weighted mean of the patterns must sum to a horizontal line
  - Each of the k patterns must have approximately equal numbers of trivial matches

(Not) Finding rules in time series

The basic idea:

- Do STS clustering on a single time series.
- Give the cluster centers discrete labels (pattern 1, pattern 2 etc).
- Run a classic association rule algorithm on the discrete labels, with some temporal constraints.

"***if*** *we see* ***pattern 17***, ***then*** *we can expect to see within 20 time units,* ***pattern 27***."



Das et. al.
Rule discovery from time series. (1998). In Proc. of the 4th KDD

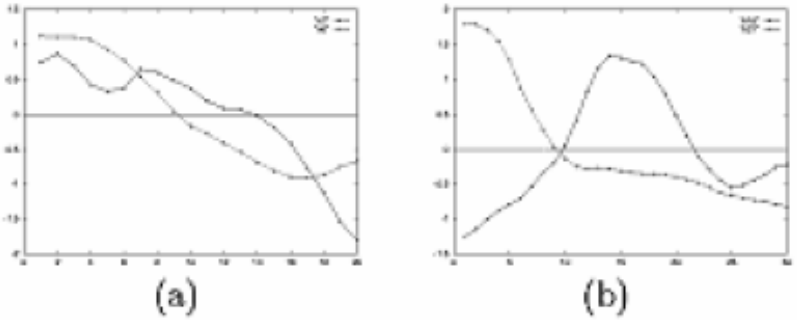
(Not) Finding rules in time series

G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. (1998). In Proc. of the 4th KDD

Extended by:

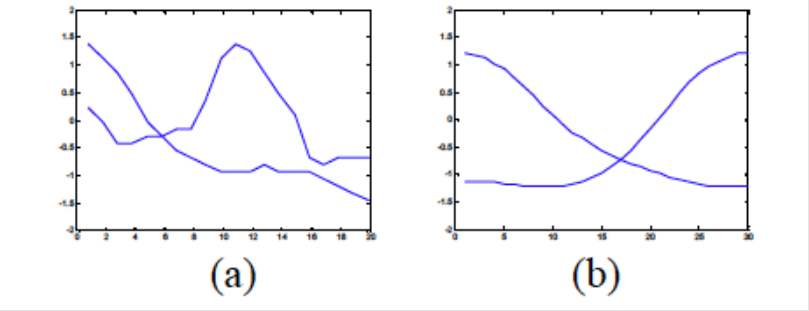
- Mori, T. & Uehara, K. (2001). Extraction of Primitive Motion and Discovery of Association Rules from Human Motion.
- Cotofrei, P. & Stoffel, K (2002). Classification Rules + Time = Temporal Rules.
- Fu, T. C., Chung, F. L., Ng, V. & Luk, R. (2001). Pattern Discovery from Stock Time Series Using Self-Organizing Maps.
- Harms, S. K., Deogun, J. & Tadesse, T. (2002). Discovering Sequential Association Rules with Constraints and Time Lags in Multiple Sequences.
- Hetland, M. L. & Sætrom, P. (2002). Temporal Rules Discovery Using Genetic Programming and Specialized Hardware.
- Jin, X., Lu, Y. & Shi, C. (2002). Distribution Discovery: Local Analysis of Temporal Rules.
- Yairi, T., Kato, Y. & Hori, K. (2001). Fault Detection by Mining Association Rules in House-keeping Data.
- Tino, P., Schittenkopf, C. & Dorffner, G. (2000). Temporal Pattern Recognition in Noisy Non-stationary Time Series Based on Quantization into Symbolic Streams.
- and many more

A Simple Experiment...



w	d	Rule	Sup %	Conf %	J-Mea.	Fig
20	5.5	$7 \Rightarrow^{15} 8$	8.3	73.0	0.0036	(a)
30	5.5	$18 \Rightarrow^{20} 21$	1.3	62.7	0.0039	(b)

*"**if** stock rises then falls greatly, follow a smaller rise, **then** we can expect to see within 20 time units, a pattern of rapid decrease followed by a leveling out."*



w	d	Rule	Sup %	Conf %	J-Mea	Fig
20	5.5	$11 \Rightarrow^{15} 3$	6.9	71.2	0.0042	(a)
30	5.5	$24 \Rightarrow^{20} 19$	2.1	74.7	0.0035	(b)

Our reimplementation
The punch line is…

Finding order in randomness?!

What we are NOT Claiming

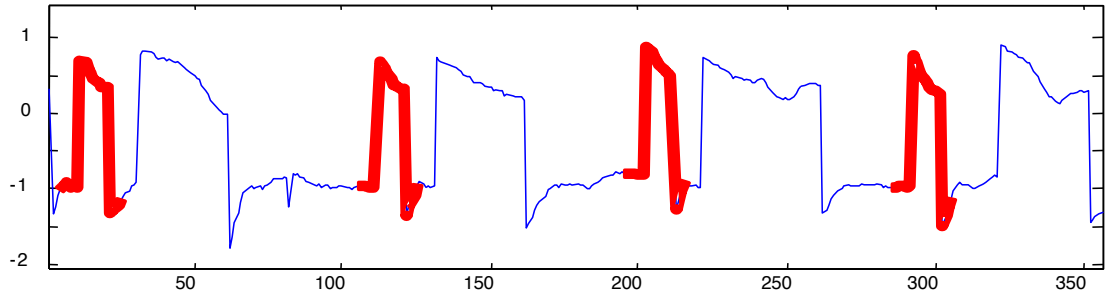
- Clustering of time series is meaningless
- Sliding windows is always a bad thing
- Clustering of *discrete* sequences with sliding windows is flawed
- People are deliberately publishing results that they know are meaningless

Is There Another Way?

- The problem with STS clustering is that every subsequence is considered.
- If we want to find true patterns, we need to consider only the subsequences that matter.
  - Chicken & Egg problem?!

A Tentative Solution: Motif-based Clustering

- Time Series Motifs!!
  - Frequently re-occurring patterns.
  - Find m-motifs (m >> k)



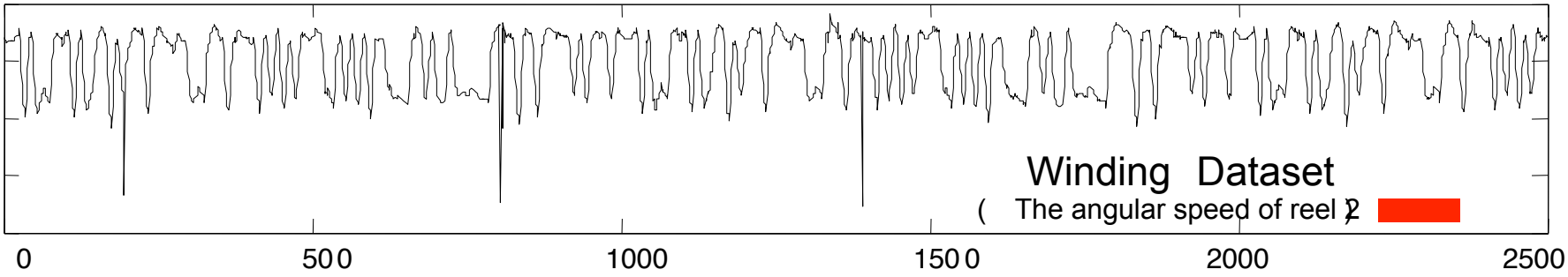
Lin, J., Keogh, E., Patel, P. & Lonardi, S. (2002). Finding Motifs in Time Series. In the *2nd Workshop on Temporal Data Mining,* at the *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Edmonton, Alberta, Canada. July 23-26, 2002.

Chiu, B. Keogh, E., & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. In the *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. August 24 - 27, 2003. Washington, DC, USA.

Time Series Motif Discovery

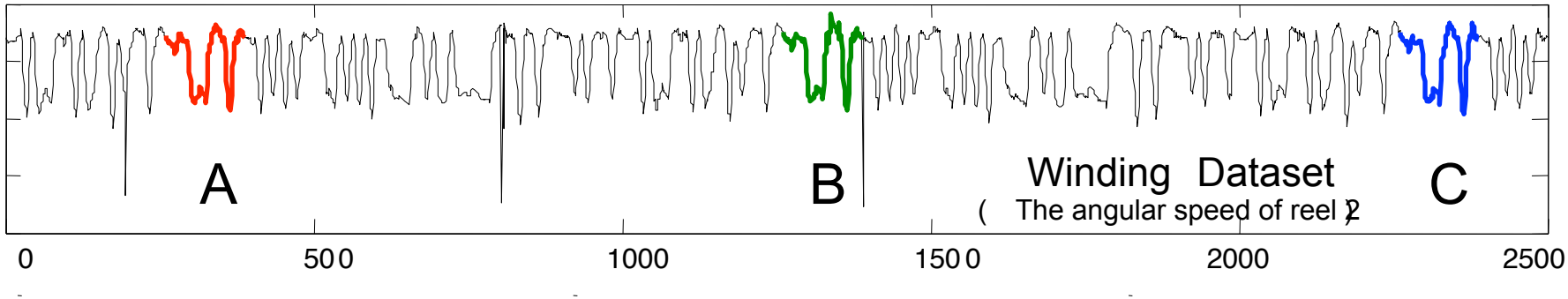
Time Series Motif Discovery

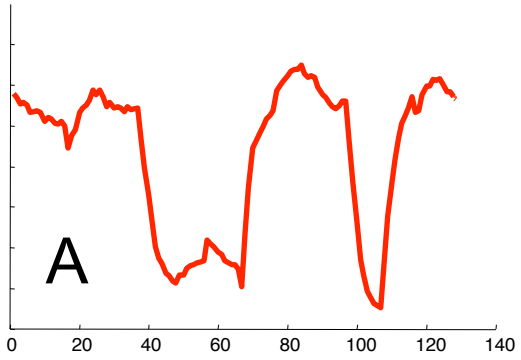
(finding repeated patterns)

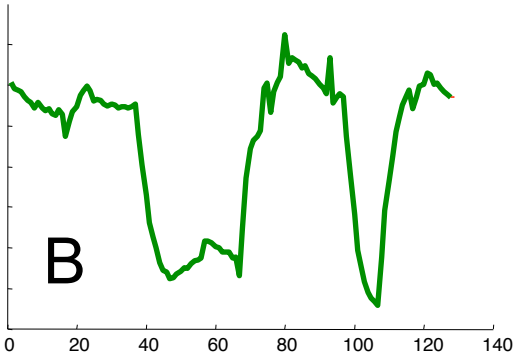


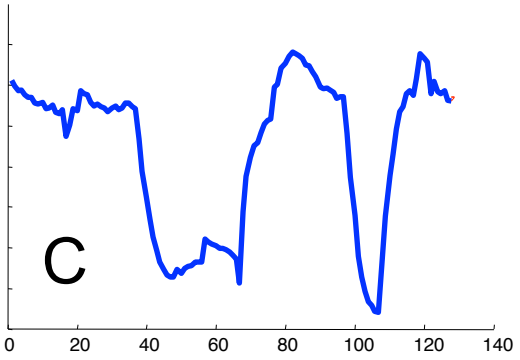
Time Series Motif Discovery

(finding repeated patterns)





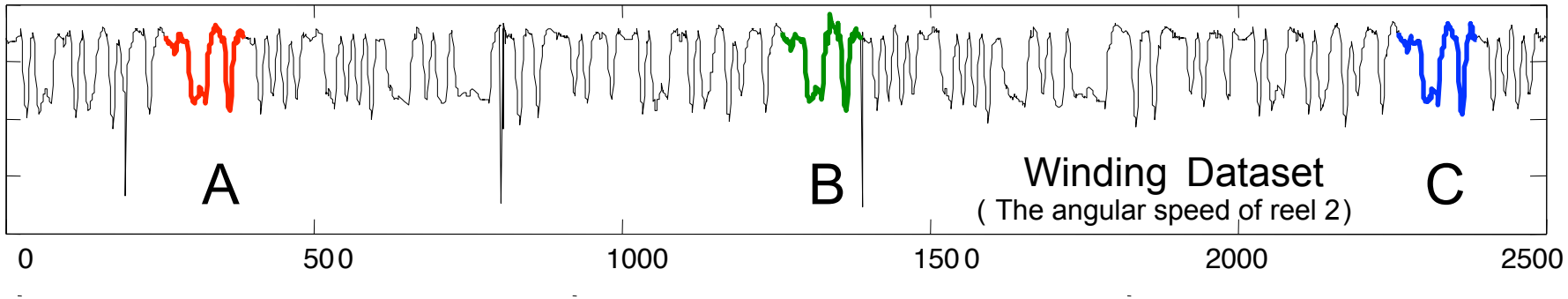


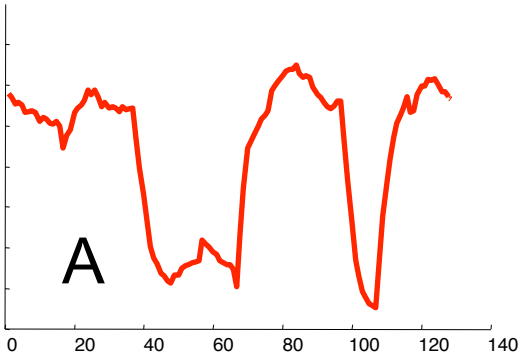


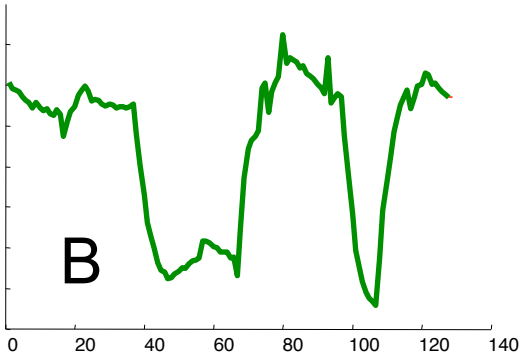
Time Series Motif Discovery

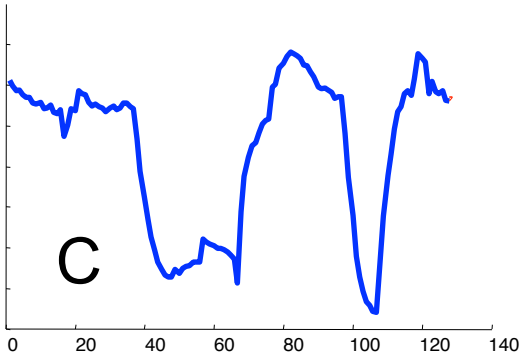
(finding repeated patterns)

To find these 3 motifs would require about 6,250,000 calls to the Euclidean Distance function!









Why Find Motifs?

· Mining **association rules** in time series requires the discovery of motifs. These are referred to as *primitive shapes* and *frequent patterns*.

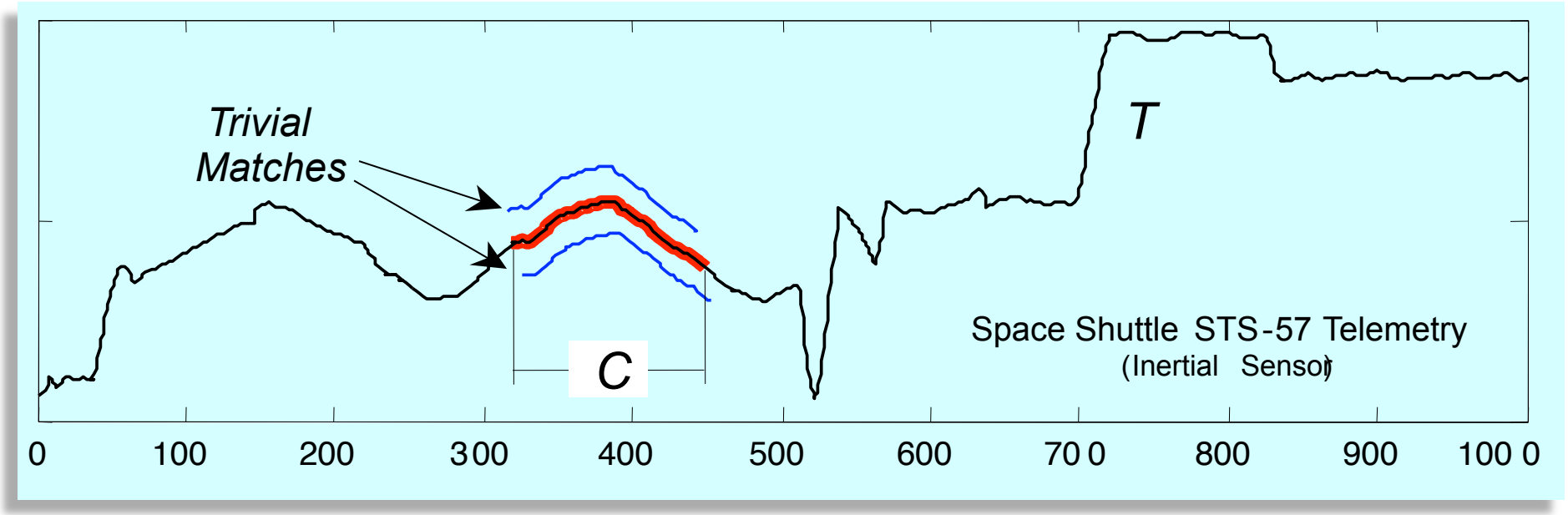
· Several time series **classification algorithms** work by constructing typical prototypes of each class. These prototypes may be considered motifs.

· Many time series **anomaly/interestingness detection** algorithms essentially consist of modeling normal behavior with a set of typical shapes (which we see as motifs), and detecting future patterns that are dissimilar to all typical shapes.

· In **robotics**, Oates et al., have introduced a method to allow an autonomous agent to generalize from a set of qualitatively different *experiences* gleaned from sensors. We see these "*experiences*" as motifs.

· In **medical data mining**, Caraca-Valente and Lopez-Chavarrias have introduced a method for characterizing a physiotherapy patient's recovery based of the discovery of *similar patterns*. Once again, we see these "*similar patterns*" as motifs.

• **Animation and video capture…** (Tanaka and Uehara, Zordan and Celly)

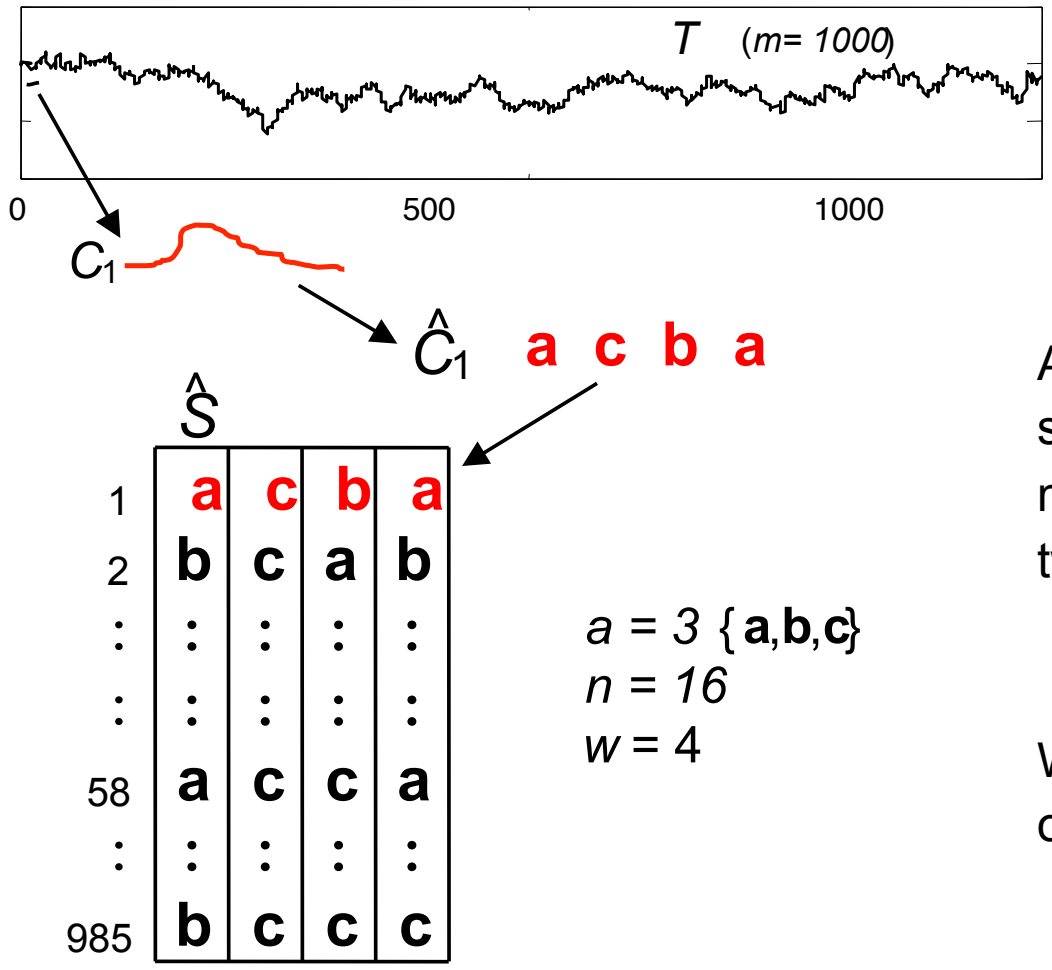


Definition 1. *Match*: Given a positive real number R (called *range*) and a time series T containing a subsequence C beginning at position p and a subsequence M beginning at q, if $D(C, M) \leq R$, then M is called a *matching* subsequence of C.

Definition 2. *Trivial Match*: Given a time series T, containing a subsequence C beginning at position p and a matching subsequence M beginning at q, we say that M is a *trivial match* to C if either $p = q$ or there does not exist a subsequence M' beginning at q' such that $D(C, M') > R$, and either $q < q' < p$ or $p < q' < q$.

Definition 3. *K-Motif*(n,R): Given a time series T, a subsequence length n and a range R, the most significant motif in T (hereafter called the *1-Motif*(n,R)) is the subsequence C_1 that has highest count of non-trivial matches (ties are broken by choosing the motif whose matches have the lower variance). The K^{th} most significant motif in T (hereafter called the *K-Motif*(n,R)) is the subsequence C_K that has the highest count of non-trivial matches, and satisfies $D(C_K, C_i) > 2R$, for all $1 \leq i < K$.

A simple worked example of the motif discovery algorithm

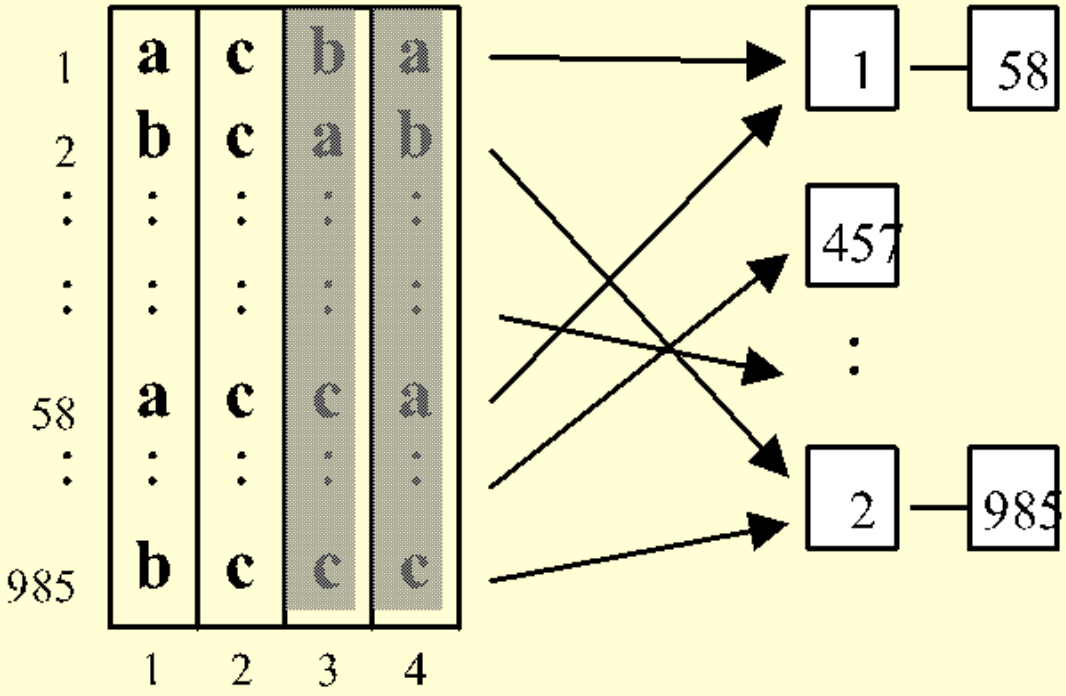


a = 3 {**a**,**b**,**c**}
n = 16
w = 4

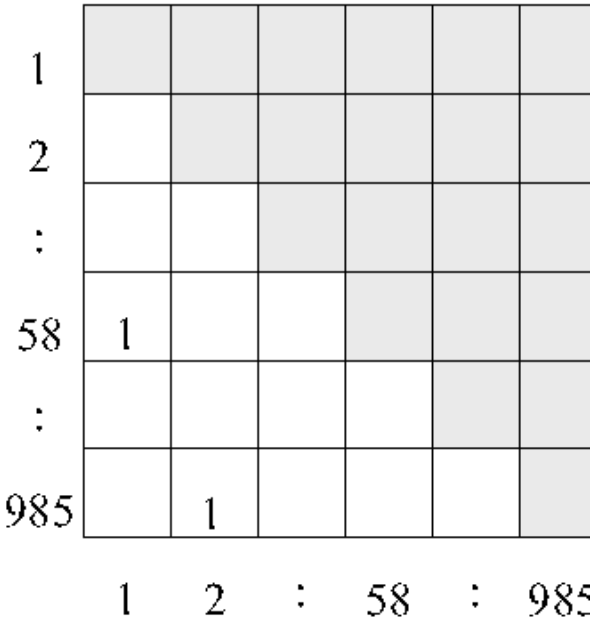
Assume that we have a time series *T* of length 1,000, and a motif of length 16, which occurs twice, at time T_1 and time T_{58}.

We can build a hash table, keyed on the SAX words

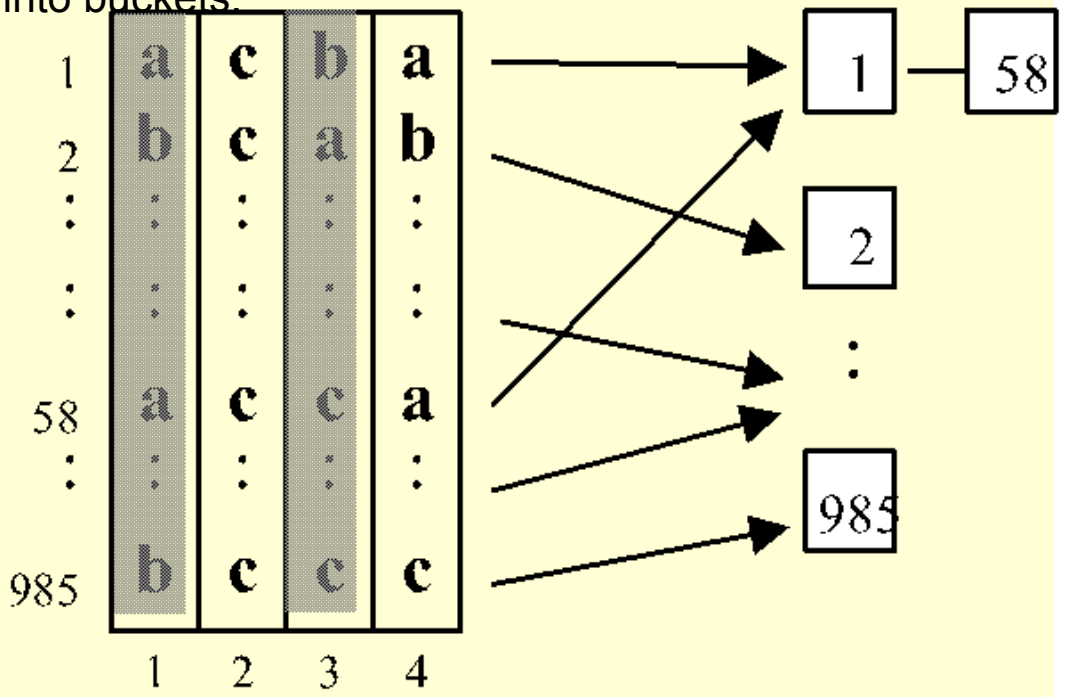
A mask {1,2} was randomly chosen, so the values in columns {1,2} were used to project matrix into buckets.



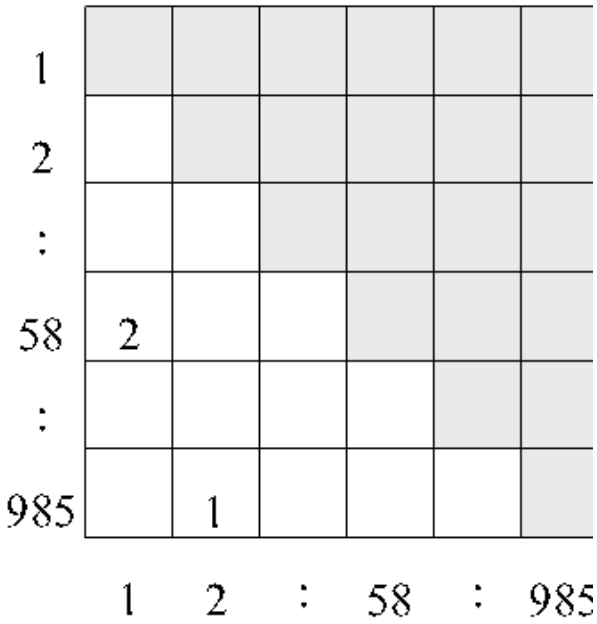
Collisions are recorded by incrementing the appropriate location in the collision matrix



A mask {2,4} was randomly chosen, so the values in columns {2,4} were used to project matrix into buckets.

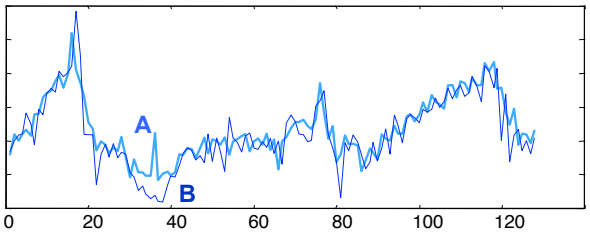


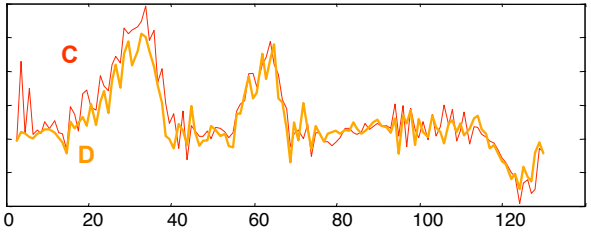
Once again, collisions are recorded by incrementing the appropriate location in the collision matrix

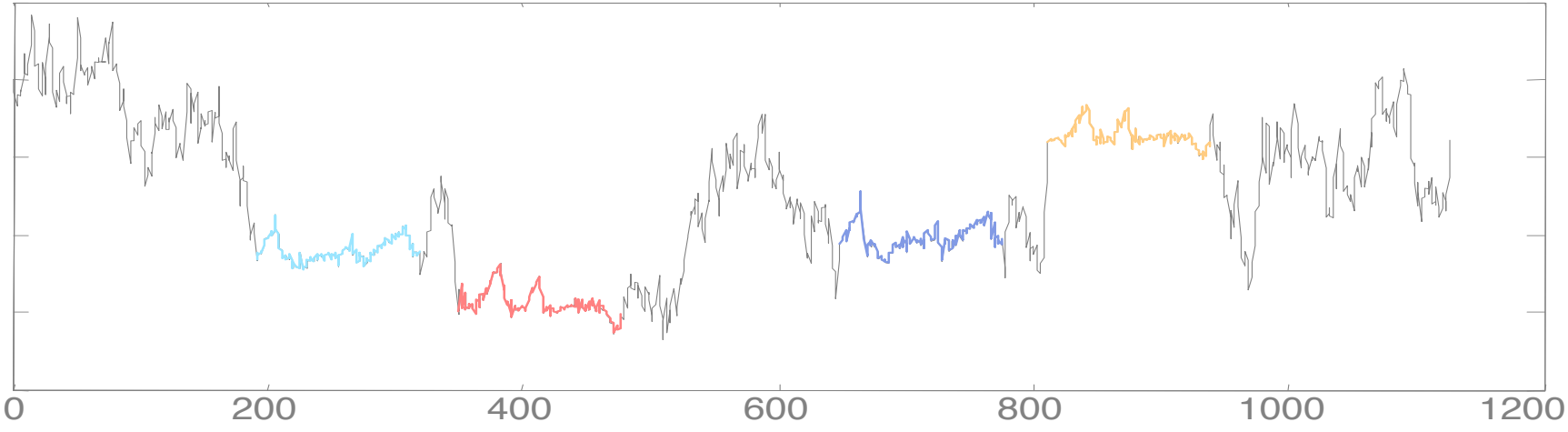


A Simple Experiment

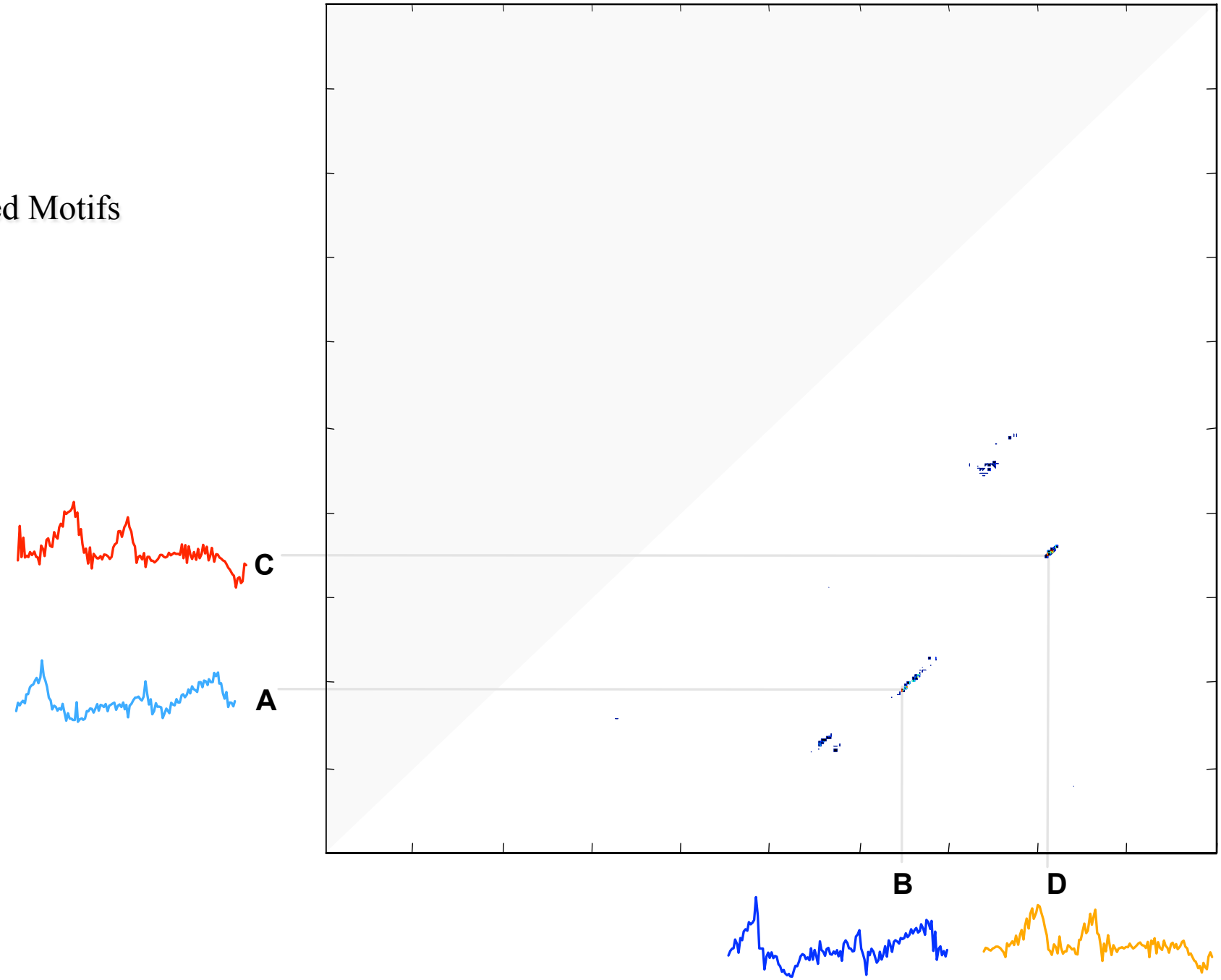
Let us imbed two motifs into a random walk time series, and see if we can recover them



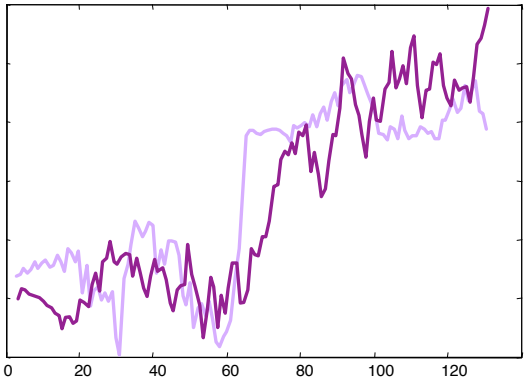


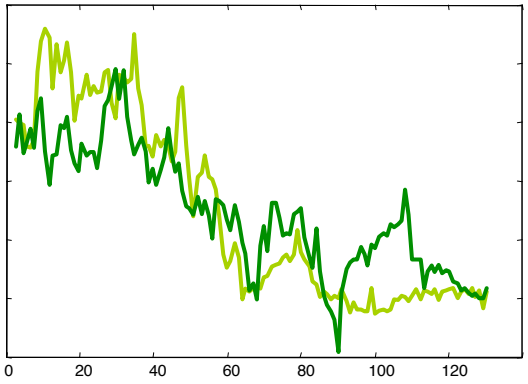


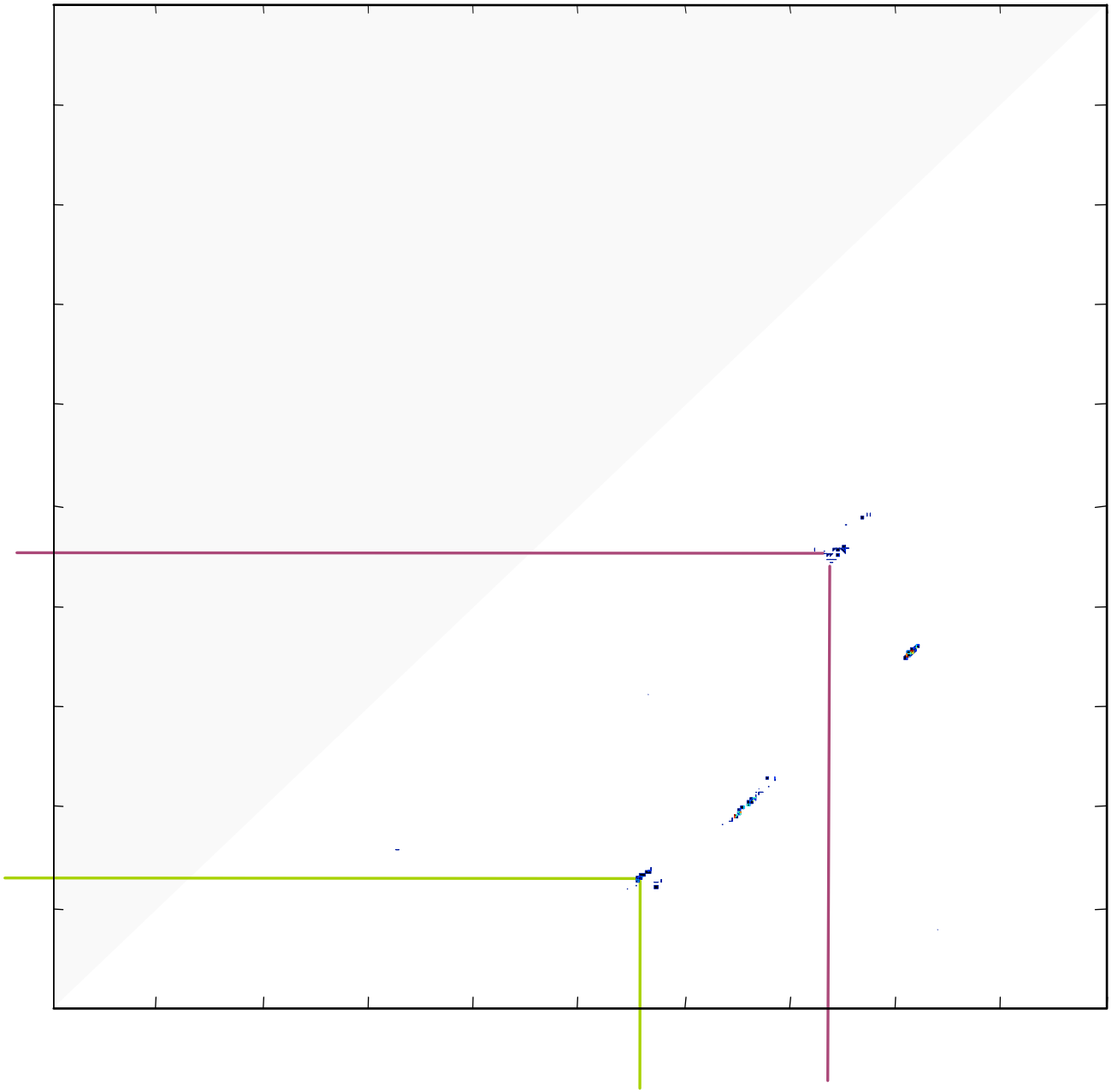
Planted Motifs



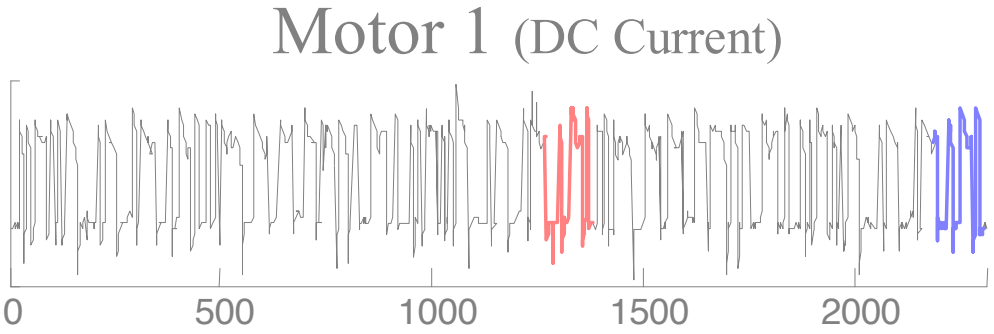
“Real” Motifs

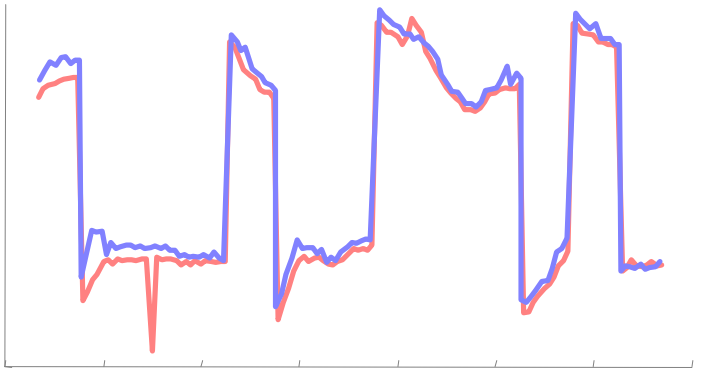


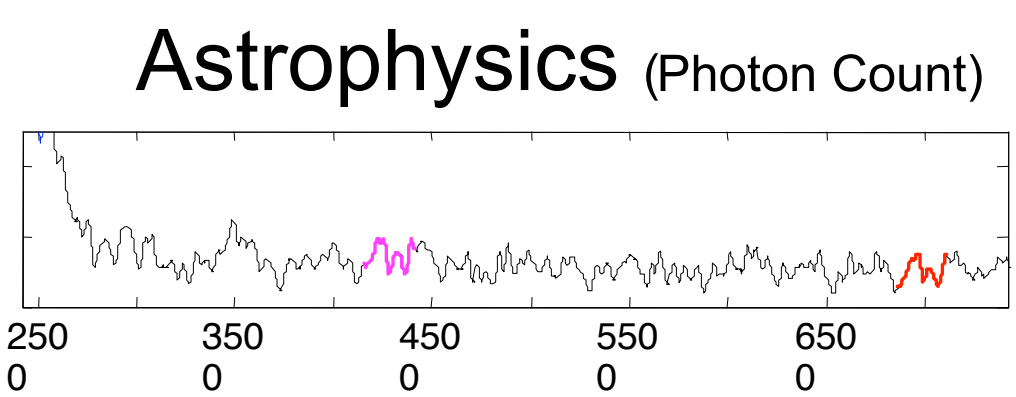


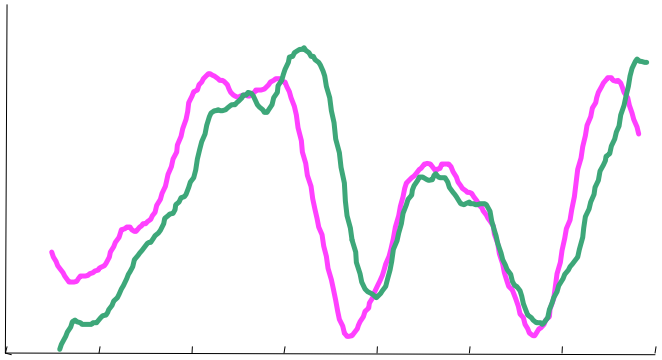


Some Examples of Real Motifs





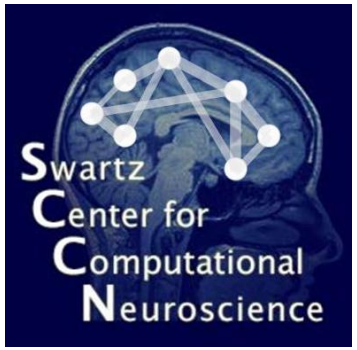




Finding Time Series Motifs on Disk-Resident Data

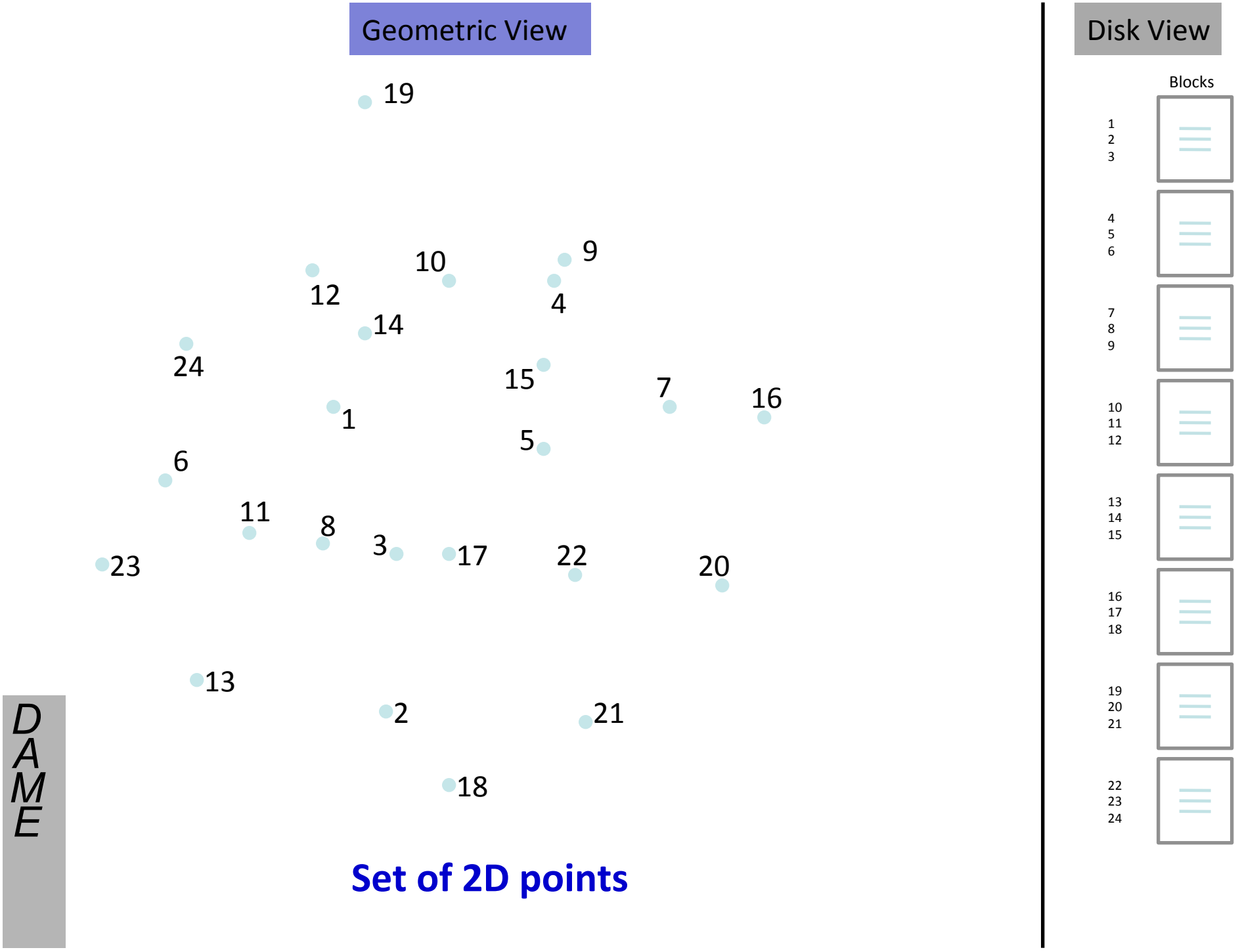
Abdullah Mueen, Dr. Eamonn Keogh
UC Riverside

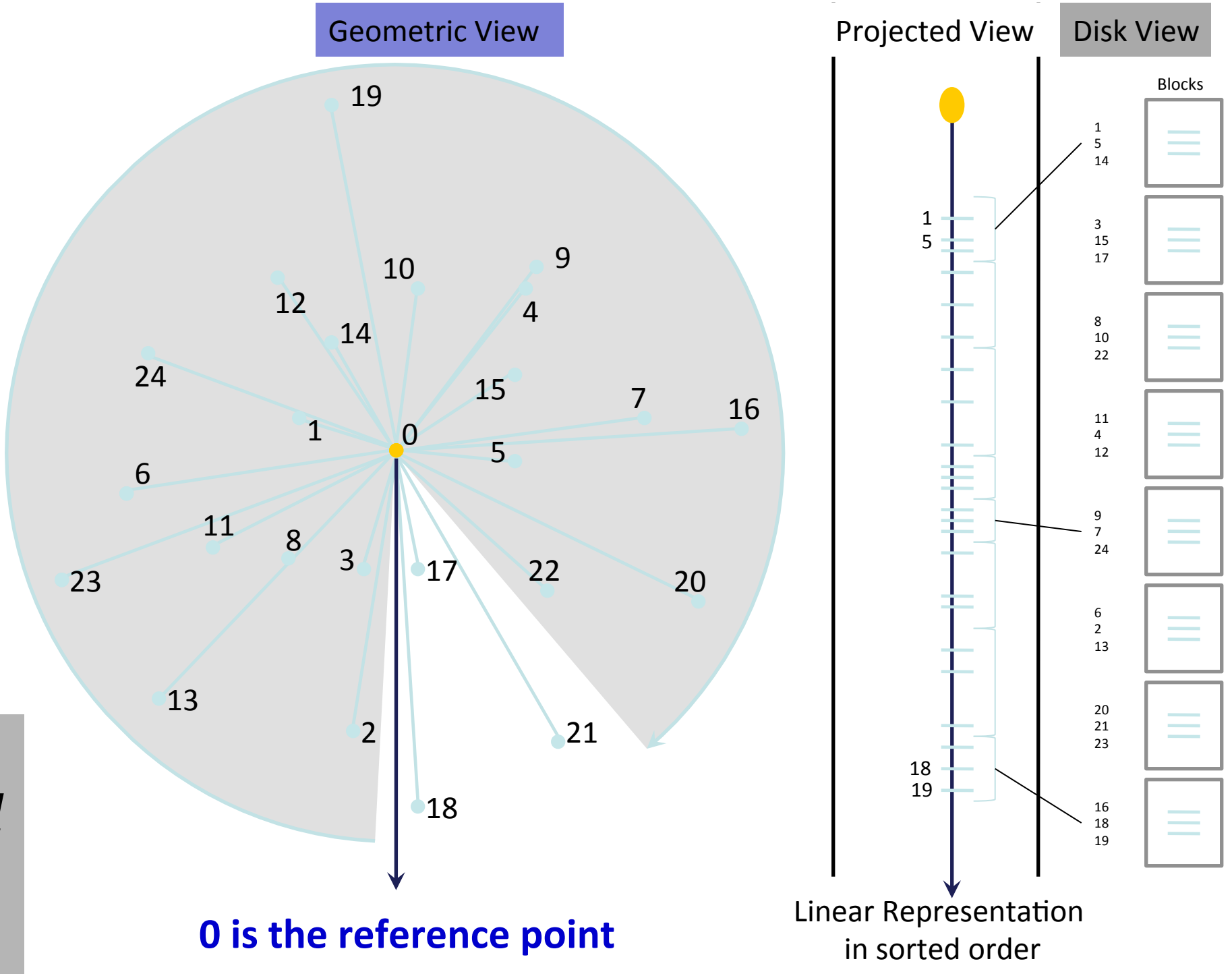
Nima Bigdely-Shamlo
Swartz Center for Computational Neuroscience, UCSD

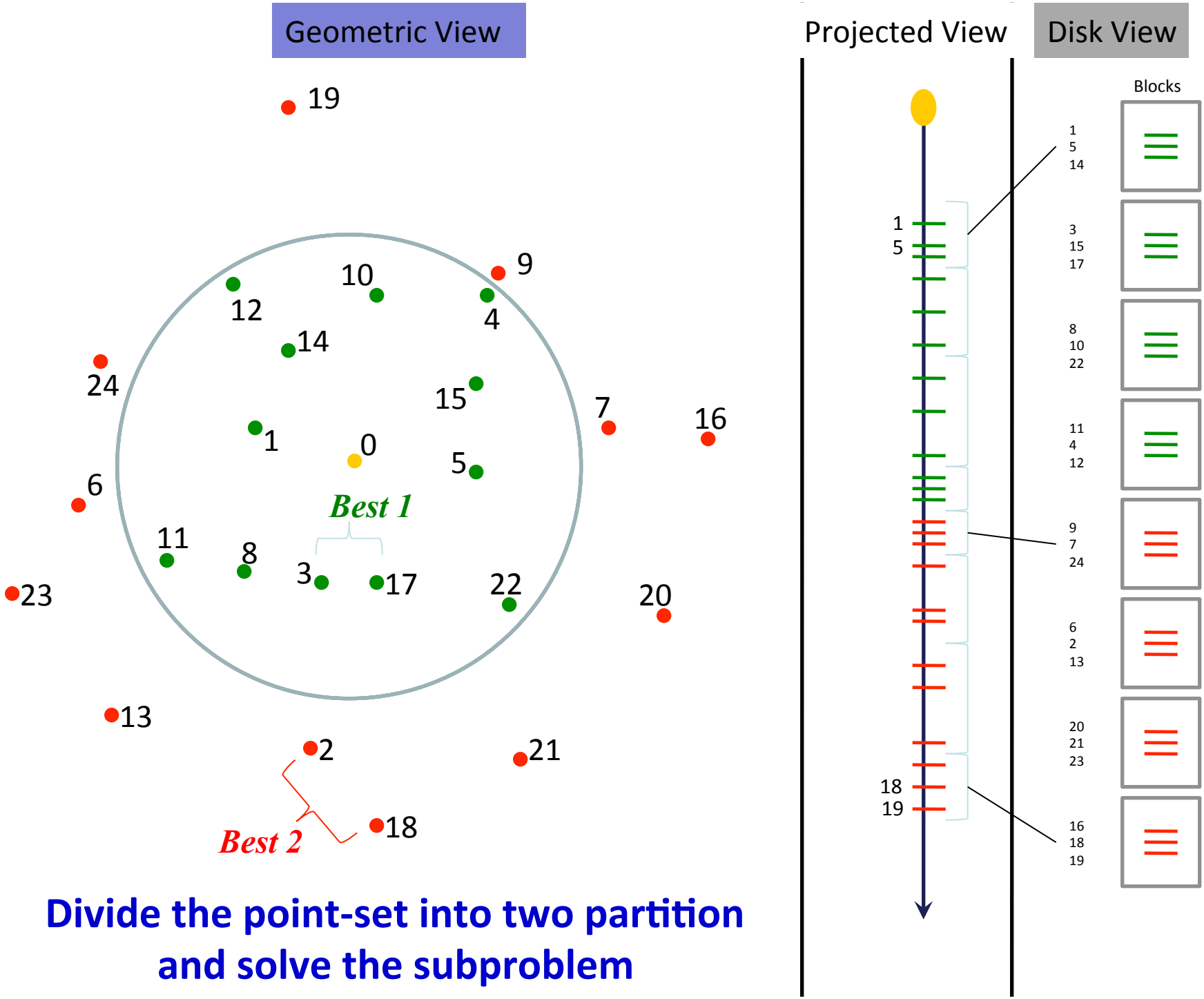


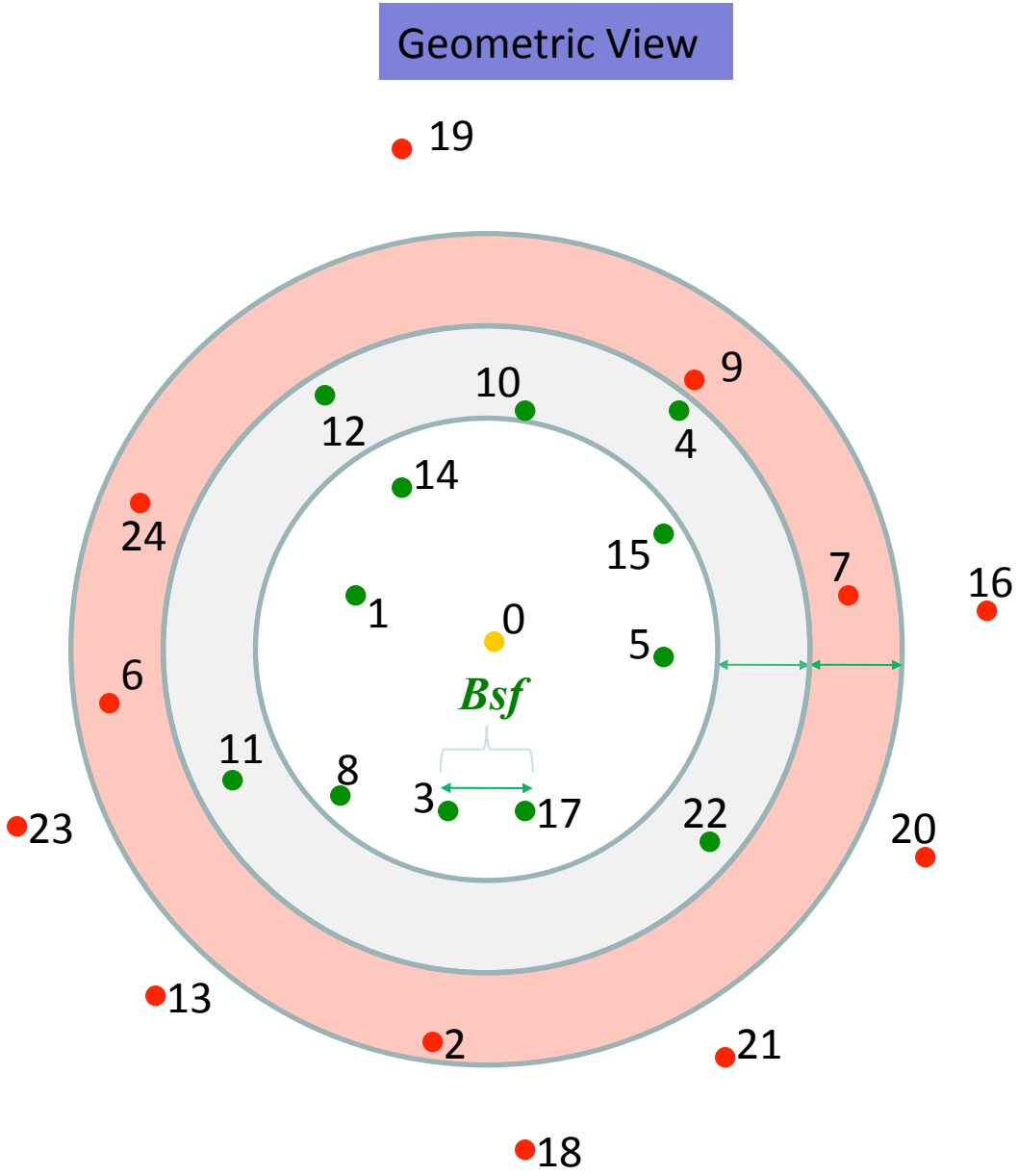
Motif Discovery in Disk-Resident Datasets

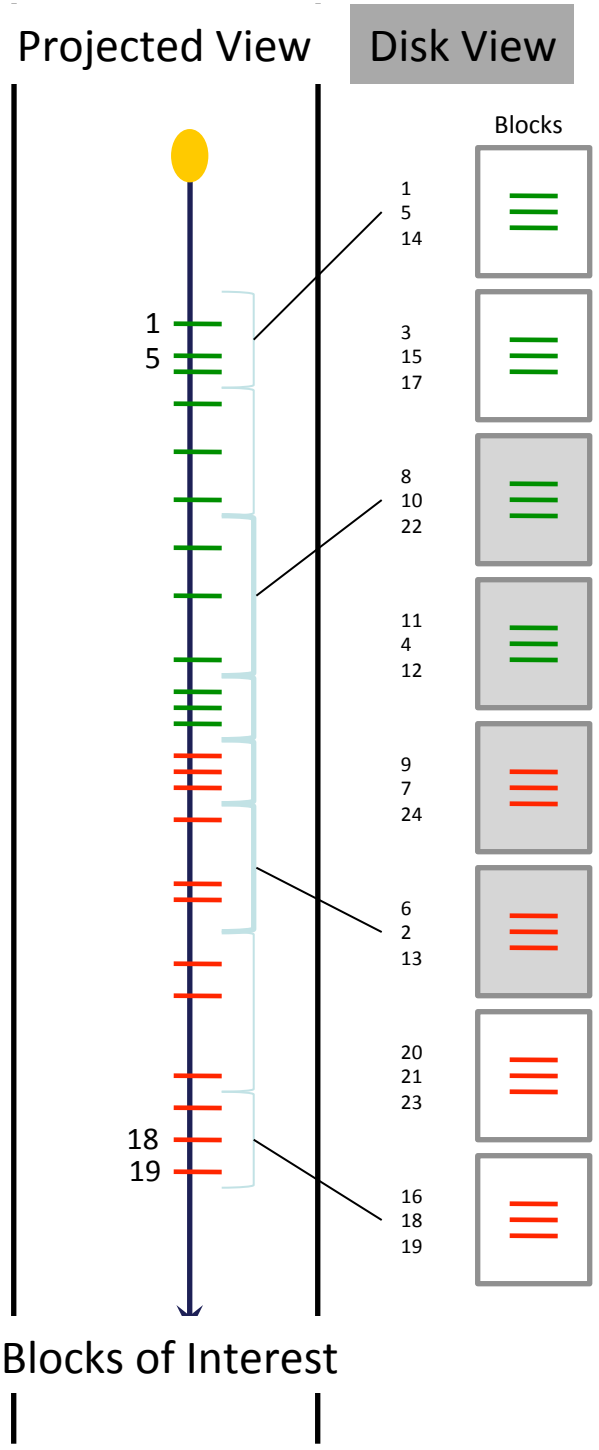
- Large datasets
  - Light Curves of Stars.
  - Performance Counters of Data Centers.
- Pseudo time series dataset
  - "80 million Tiny Images"
- Database of normalized subsequences
  - An hour long trace of EEG generates over one million normalized subsequences.


Geometric View
19
9
10
12
4
14
24
15
7
16
1
5
6
11
8
3
17
22
20
23
13
2
21
18
Set of 2D points
DAME
Disk View
Blocks
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24


Geometric View
Projected View
Disk View
Blocks
19
10
9
12
4
14
24
15
7
16
1
0
5
6
11
8
3
17
22
20
23
13
2
21
18
1
5
14
3
15
17
8
10
22
11
4
12
9
7
24
6
2
13
20
21
23
18
19
16
18
19
0 is the reference point
Linear Representation
in sorted order
DAME


Geometric View
Projected View
Disk View
Blocks
19
9
10
12
4
14
24
15
7
16
1
0
5
6
Best 1
11
8
3
17
22
20
23
13
2
21
18
Best 2
1
5
14
3
15
17
8
10
22
11
4
12
9
7
24
6
2
13
20
21
23
18
19
16
18
19
DAME
Divide the point-set into two partition
and solve the subproblem

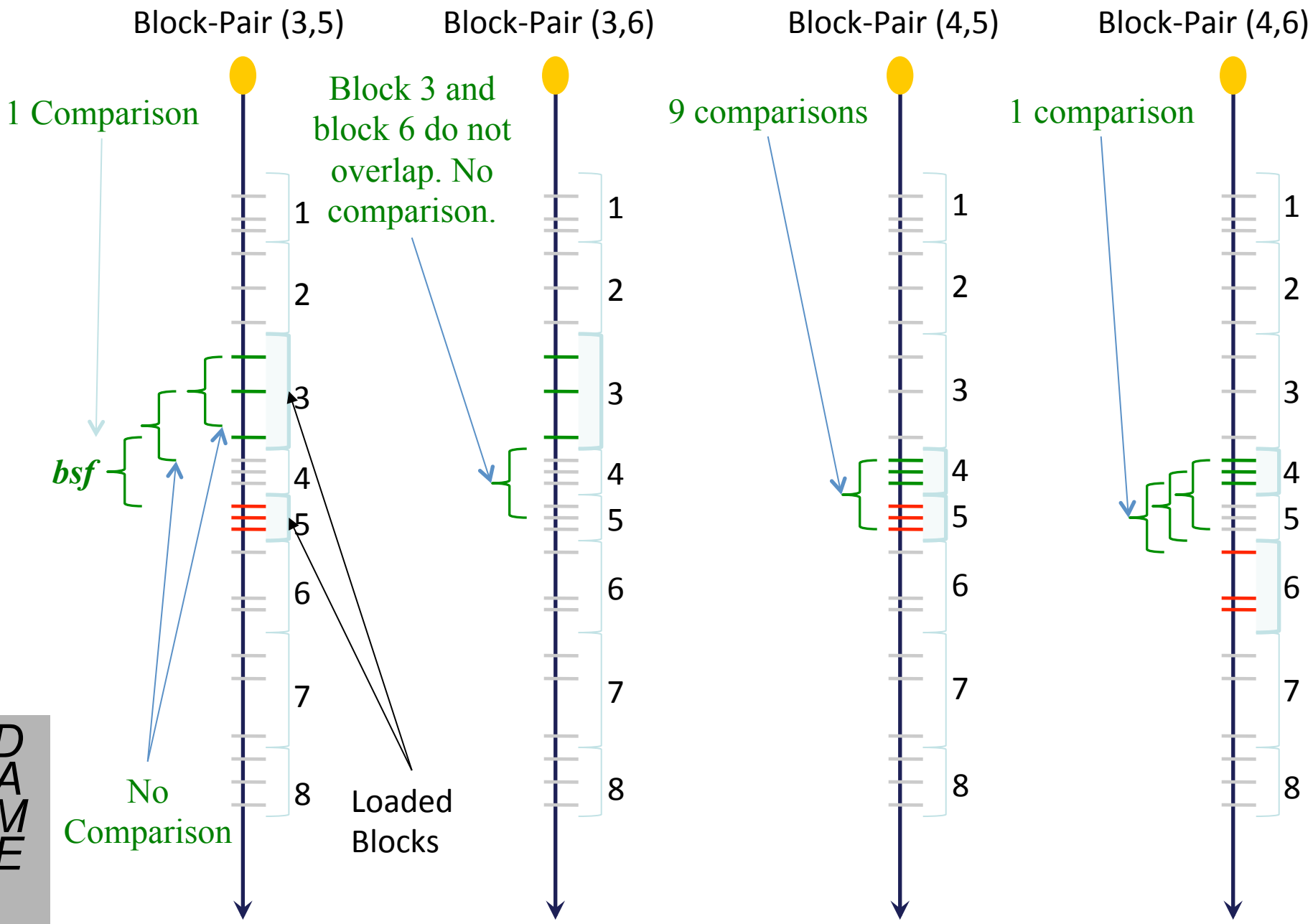




DAME

The inner ring is the region for blocks 5 and 6

The outer ring is the region for blocks 3 and 4



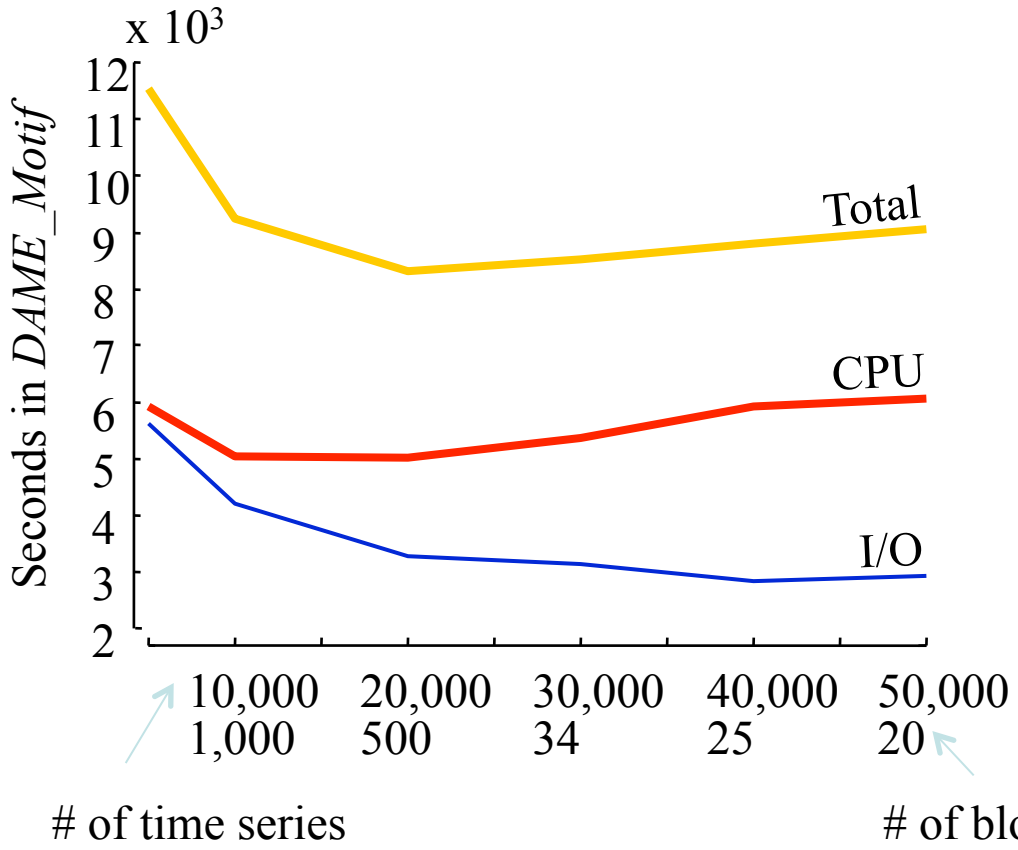
DAME

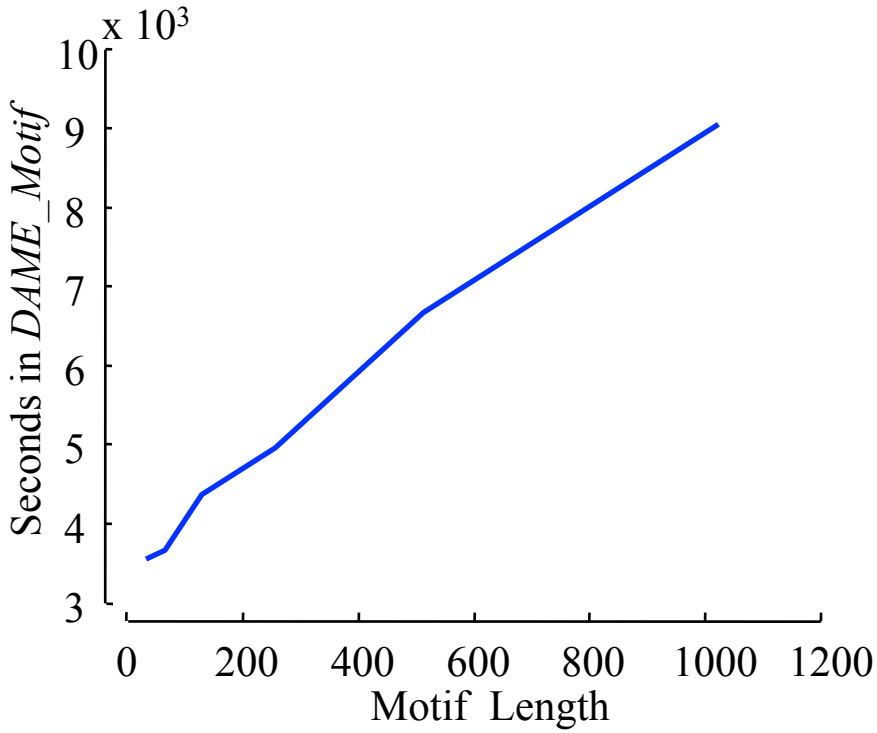
11 comparisons are made instead of 9*16=144

Speedup

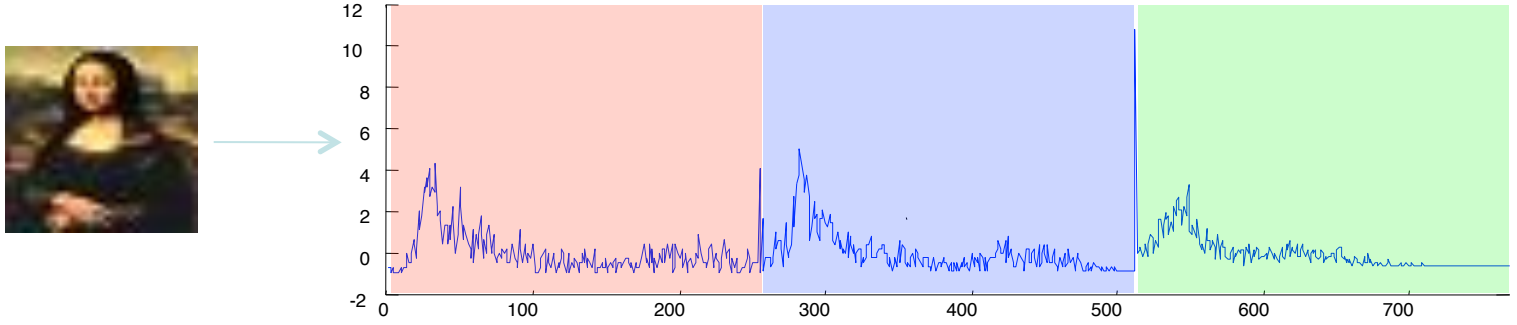
Memory	Disk	Algorithm	Largest Dataset Tested (thousands)	Time for the Largest Dataset	Estimated Time for 4.0 million
√	X	*CompletelyInMemory*	100	35 minutes	37.8 days
X	√	*CompletelyInDisk*	200	1.50 days	1.65 years
√	√	*DAME*	4,000	1.35 days	1.35 days
√	X	*NoAdditionalStorage* *(normalization done in memory)*	200	4.82 days	5.28 years

Performance Evaluation

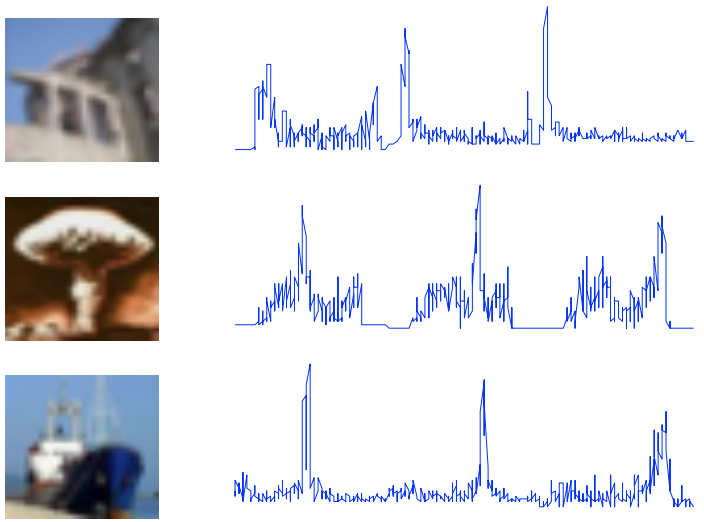




Case Study: Image Motifs



- Concatenated color histogram is considered as pseudo time series.
- Each time series is of length 256*3 = 768.
- 80 million tiny images of 32X32 resolution.



80 million tiny images : collected by *Antonio Torralba, Rob Fergus, William T. Freeman* at MIT.

Case Study: Image Motifs

Duplicate Image

Near Duplicate Image

- *DAME* worked on the first 40 million time series in ~6.5 days
- *DAME* found 3,836,902 images which have at least one duplicate.
  - 1,719,443 unique images.
- 542,603 images have near duplicates with distance less than 0.1.

Conclusion

- *DAME*: The first exact-motif discovery algorithm that finds motif in disk-resident data.
- *DAME* is scalable to massive datasets of the order of millions of time series.
- *DAME* successfully finds motif in EEG traces and image databases.

VizTree - Motivation

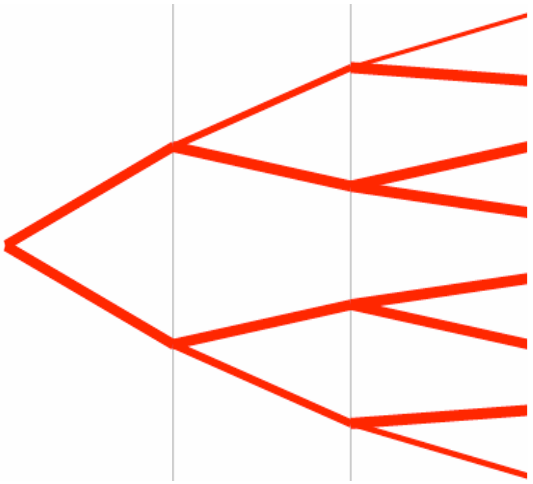
0101100101111001101001000010001000
1010011011010111000010101011101000
1111100011011011011111101001100101
1001000110100011110011011010001010
1011110001011010011011001101000100
0000100110001001110000011101000111
1100101100001010010

1000100010100100010101010000101010
0101000101011101111010110100100101
1110100101010011101010101001010101
0010101011101010100101010101100101
0101010010110010111011110100010101
1100001010000100111010100011101010
0001010101100101110101

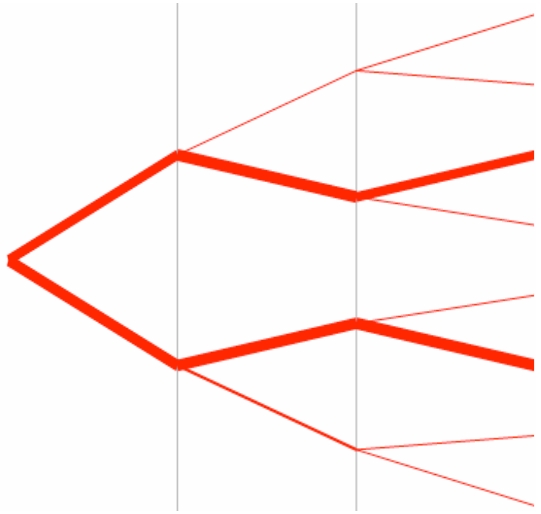
Here are two sets of bit strings. Which set is generated by a human and which one is generated by a computer?

VizTree

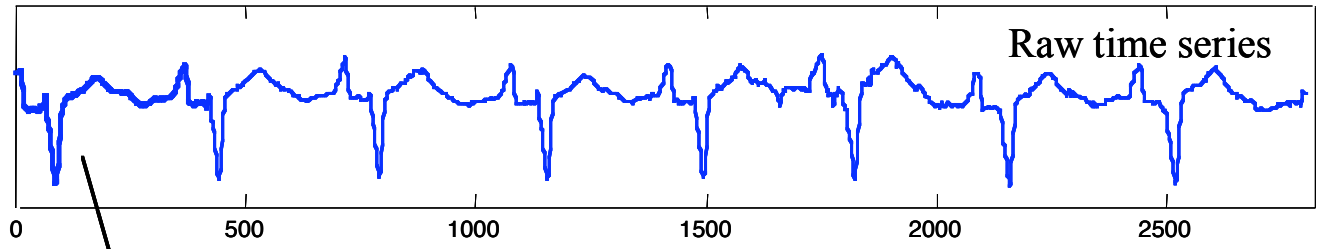
0101100101111001101001000010001010
0011011010111000010101011101111100
0011011011011111101001100100100011
1010001111001101101000101111000102
1101001101100110100000010011000102
0111000001110100110010110000101001
010



1000100010100100010101010000101001
0001010111011110101101001011101001
0101010011101010101001010010101011
1101010100101010101101010100101101
0101110111101000111000010100001001
1110101000111000010101011001011101
01



Lets put the sequences into a depth limited suffix tree, such that the frequencies of all triplets are encoded in the thickness of branches. "*humans usually try to fake randomness by alternating patterns*"



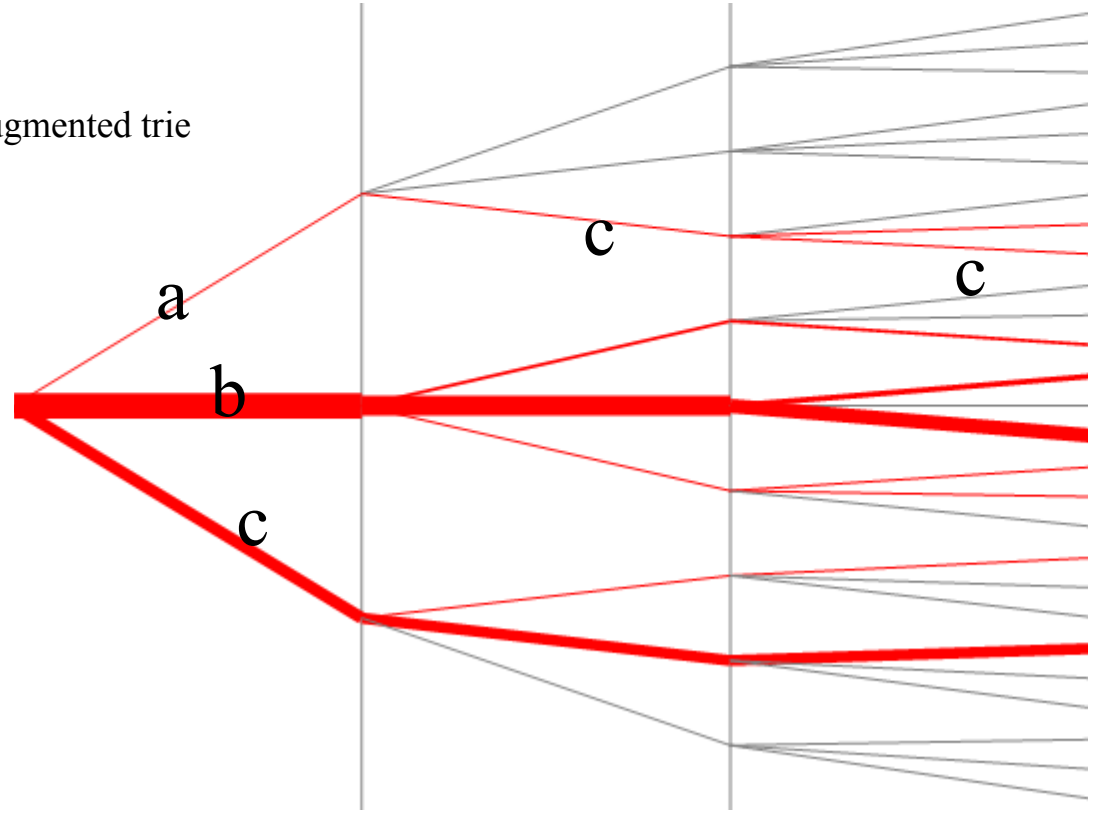
Subsequence extracted

C_1

Converted to SAX

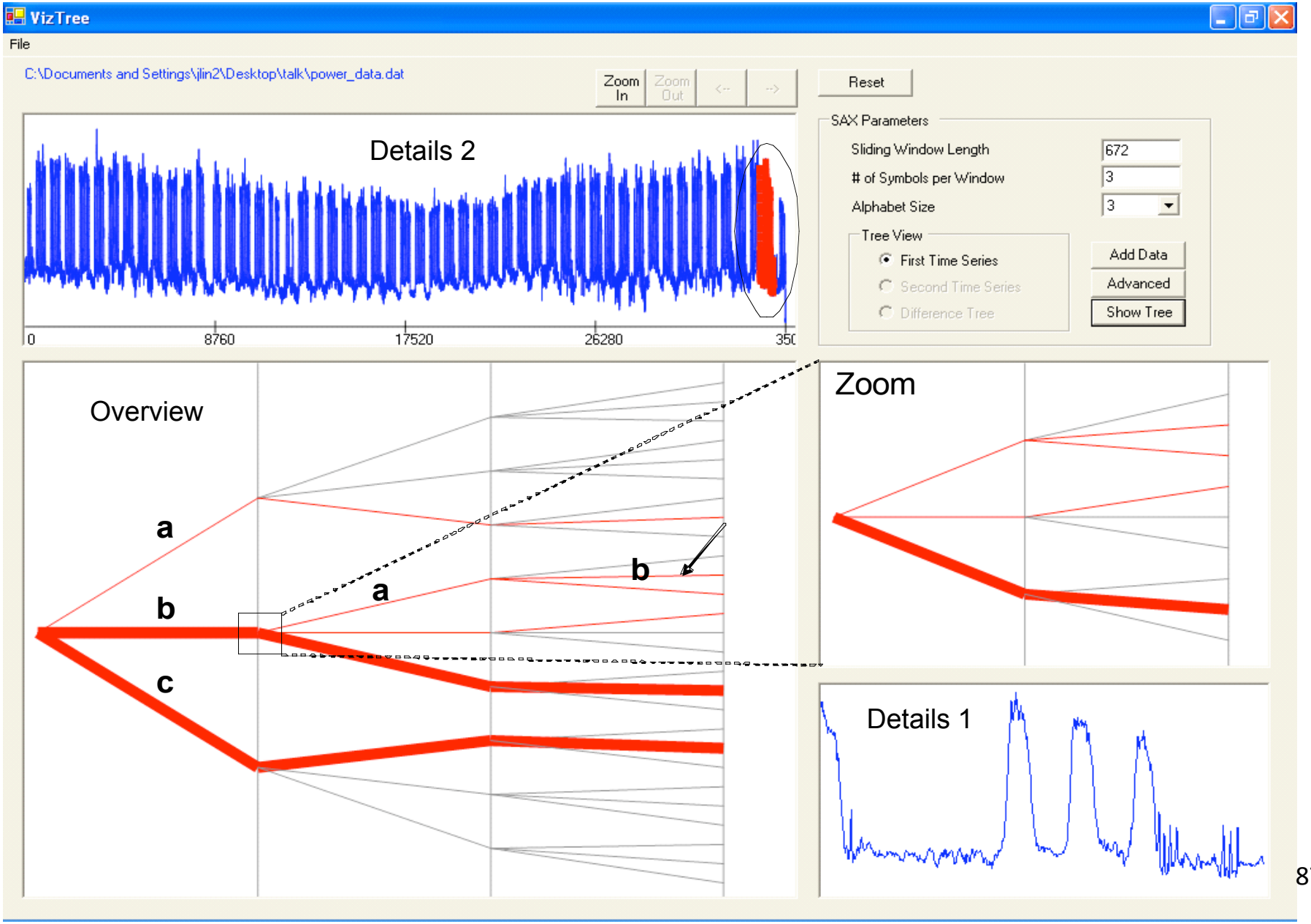
$\hat{C}_1$ **c a a**

Insert into augmented trie



VizTree

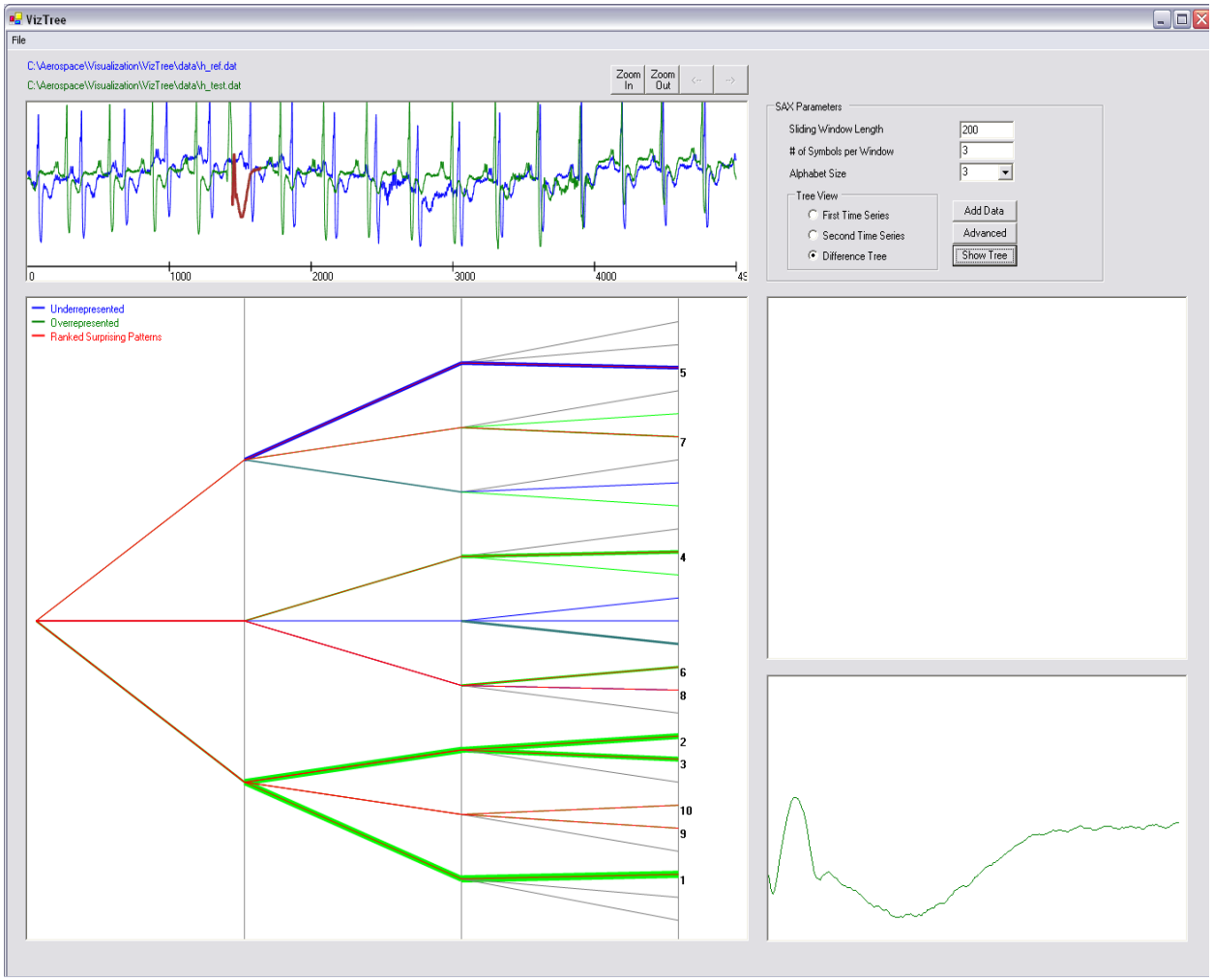
(http://www.cs.gmu.edu/~jessica/viztree.htm)



VizTree/ DiffTree

DiffTree

- Convert the two time series to SAX
- Push the data in a depth-limited suffix tree
- Encode the frequencies as the line thickness
- Encode the *difference* of frequencies as the line *color*



Blue lines - pattern is more common in A
Green lines - pattern is more common in B
Red lines - pattern is equi-frequent in A and B

Grammar-Based Motif Discovery

Grammar Induction

- Most algorithms still suffer a limitation: the length of motif needs to be given.
- Grammar = repeated patterns, repeated structure

Simple example of grammar from a string:

Input string:

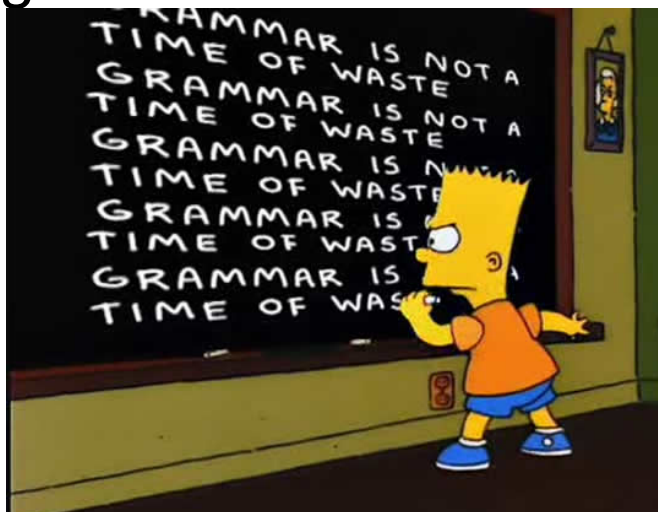
1 1 2 2 1 1 1 2 2 1 1 1 2 2 1 1 1 2 2 1

(Desired) Output Grammar:

R0 → R1 R1

R1 → R2 R2

R2 → 1 1 2 2 1



SEQUITUR

What is SEQUITUR?

- Introduced by Nevill-Manning and Witten, 1996
- Online, linear-time, grammar-based compression algorithm
- Infers a context-free grammar from a sequence of symbols
- Works by compressing repeated patterns of input string

Policies of SEQUITUR

- Digram uniqueness
- Rule utility

Example

Input string: abcdabc

Grammar
S → a

Digrams

Example

Input string: abcdabc

Grammar
S → ab

Digrams
ab

Example

Input string: abcdabc

Grammar
S → abc

Digrams
ab
bc

Example

Input string: abcdabc

Grammar
S → abcd

Digrams
ab
bc
cd

Example

Input string: abcdabc

Grammar
S → abcda

Digrams
ab
bc
cd
da

Example

Input string: abcdabc

Grammar
S → abcdab

Digrams
ab
bc
cd
da

Enforcing digram uniqueness

ab occurs twice

Creating new rule A → ab

Example

Input string: abcdabc

Grammar
S → AcdA
A → ab

Digrams
ab
Ac
cd
dA

Enforcing digram uniqueness

ab occurs twice

Creating new rule A → ab

Example

Input string: abcdabc

Grammar
S → AcdAc
A → ab

Digrams
ab
Ac
cd
dA

Enforcing digram uniqueness

Ac occurs twice

Creating new rule B → Ac

Example

Input string: abcdabc

Grammar
S → BdB
A → ab
B → Ac

Digrams
ab
Ac
Bd
dB

Enforcing digram uniqueness

Ac occurs twice

Creating new rule B → Ac

Example

Input string: abcdabc

Grammar
S → BdB
B → abc

Digrams
ab
Ac
Bd
dB

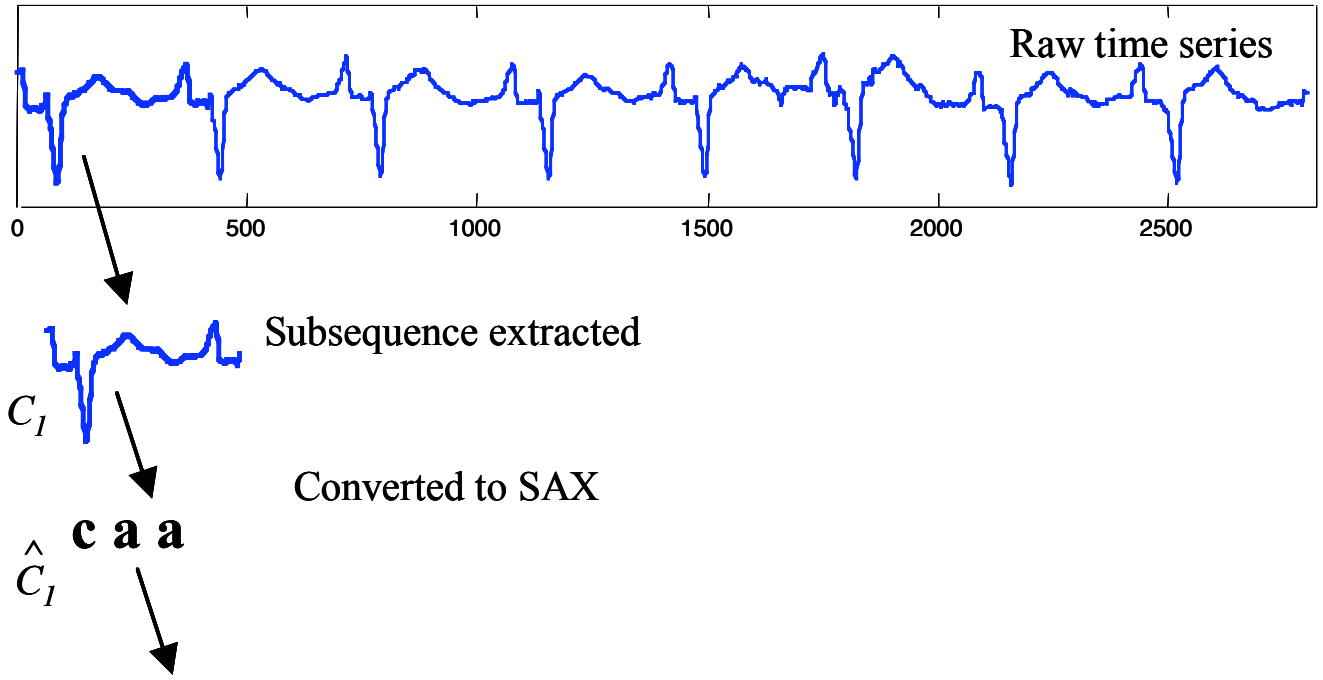
Enforcing rule utility

A occurs only once

Removing A → ab

There is a problem, however: time series are real-valued!

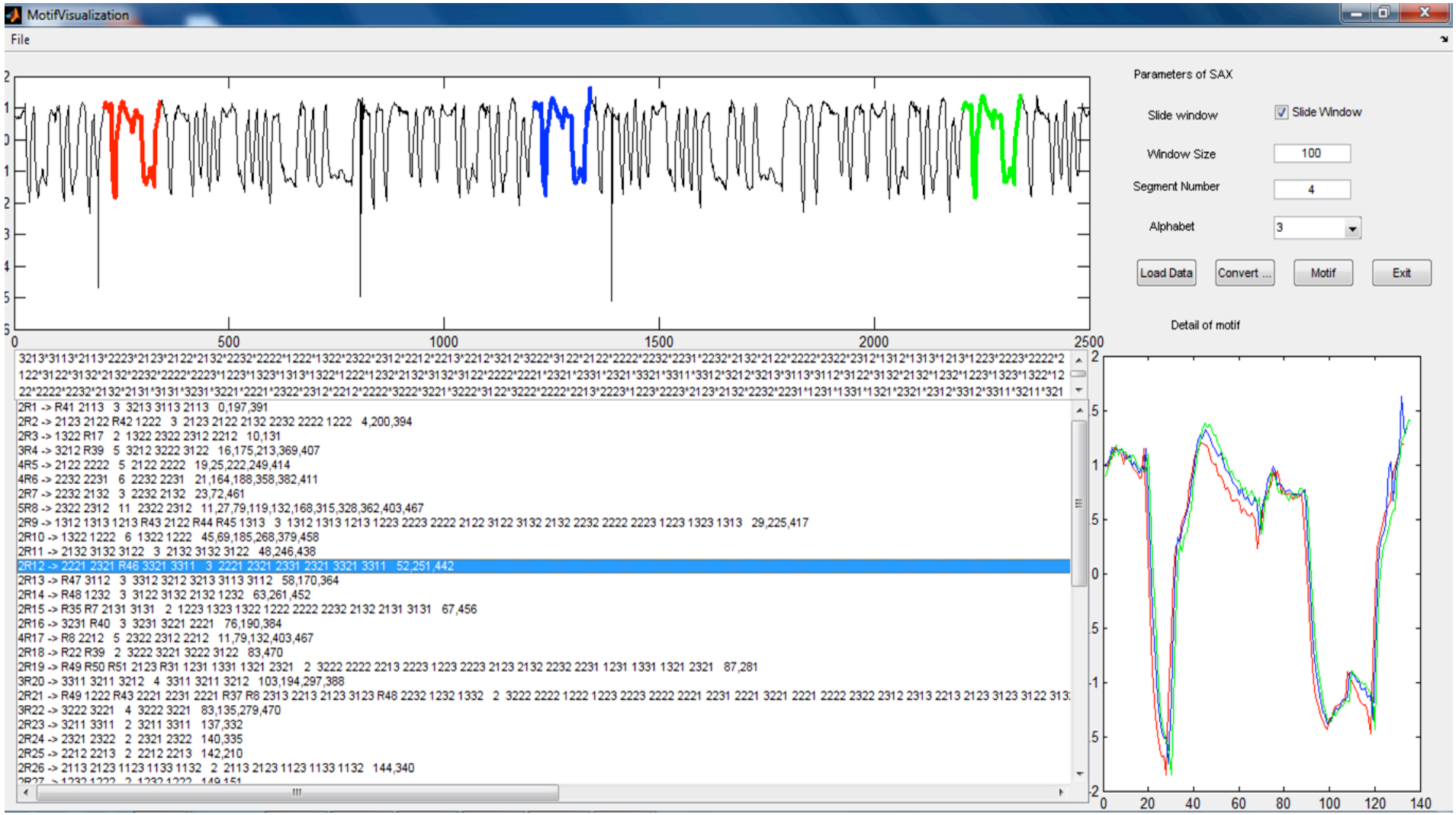
Solution: *SAXify* the data!



S = caa caa cab cac ccc caa caa caa cab…
= caa_1 cab_3 cac_4 ccc_5 caa_6 cab_9

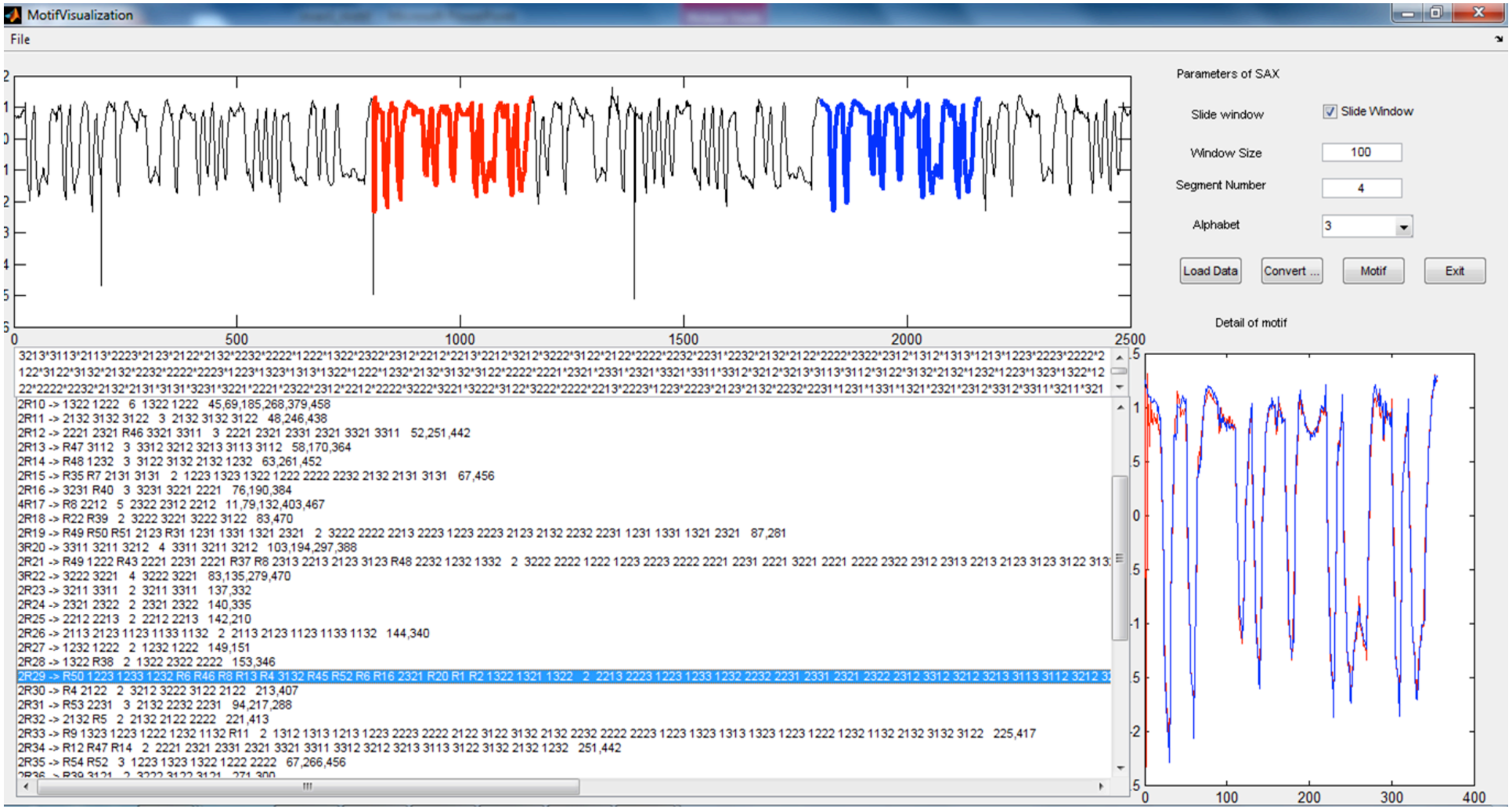
numerosity reduction

GrammarViz: Variable-Length Motif Discovery



Yuan Li, Jessica Lin, and Tim Oates. 2012. Visualizing variable-length time series motifs. In Proceedings of the 2012 SIAM International Conference on Data Mining. Anaheim, CA. Apr 26-28. Pages 895-906

GrammarViz: Variable-Length Motif Discovery



Multivariate Motifs

Limitations

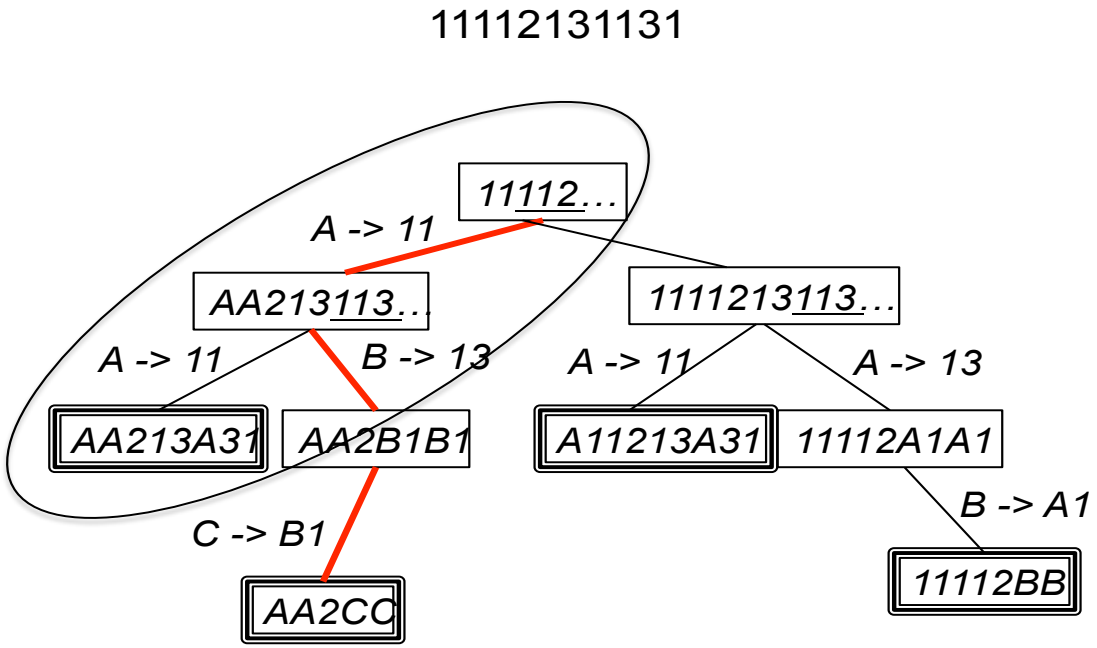
- Greedy algorithm
- Grammar found is not minimal!

Input string: $(11112131131)^4$

Grammar Rule	Expanded Grammar Rule
R0 –> R1 R1 R2 R2 R2 R3 3 1	1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1
R1 –> 1 1	1 1
R2 –> R3 R4 R1 1	2 1 3 1 1 3 1 1 1 1 1
R3 –> 2 1 R 4	2 1 3 1 1
R4 –> 3 R1	3 1 1

Tree Search for Grammar Induction

- Trigrams: any three adjacent symbols
- A trigram is made of 2 digrams (e.g., “abc” can be parsed as a(bc) or (ab)c
- Keep track of all digrams, and (selectively) substitute repeating digrams with new symbols



Tree Search for Grammar Induction

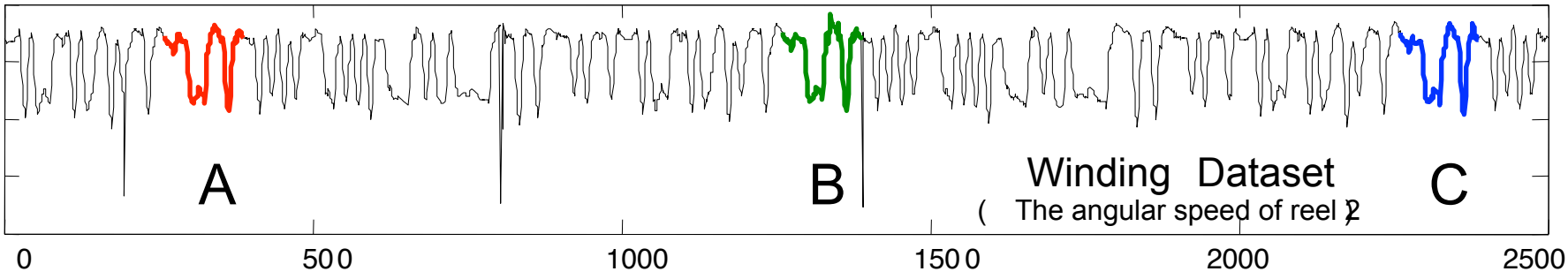
Input string: $(11112131131)^4$

Grammar Rule	Expanded Grammar Rule
R0 –> R1 R1	1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1
R1 –> R2 R2	1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1
R2 –> R3 R3 2 1 3 R4	1 1 1 1 2 1 3 1 1 3 1
R3 –> 1 1	1 1
R4 –> R3 3 1	1 1 3 1

Motifs Discovery Challenges

How can we find motifs…

- Without having to specify the length/other parameters
- In massive datasets
- In streaming data
- While ignoring "background" motifs (ECG example)
- Under time warping, or uniform scaling
- While assessing their significance



Finding these 3 motifs requires about 6,250,000 calls to the Euclidean distance function

Anomaly (interestingness) detection

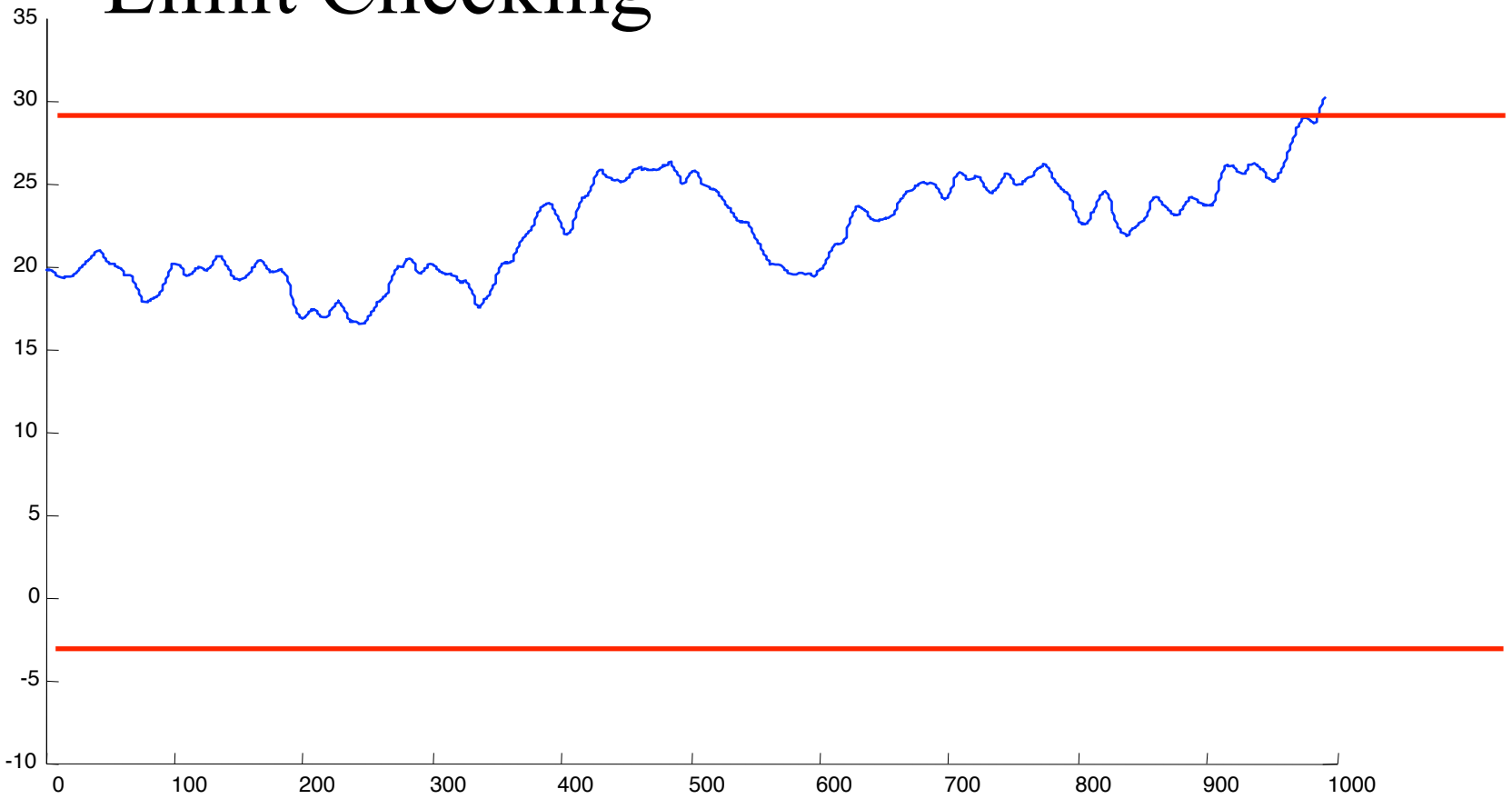
We would like to be able to discover surprising (unusual, interesting, anomalous) patterns in time series.

Note that we don't know in advance in what way the time series might be surprising

Also note that "surprising" is very context dependent, application dependent, subjective etc.

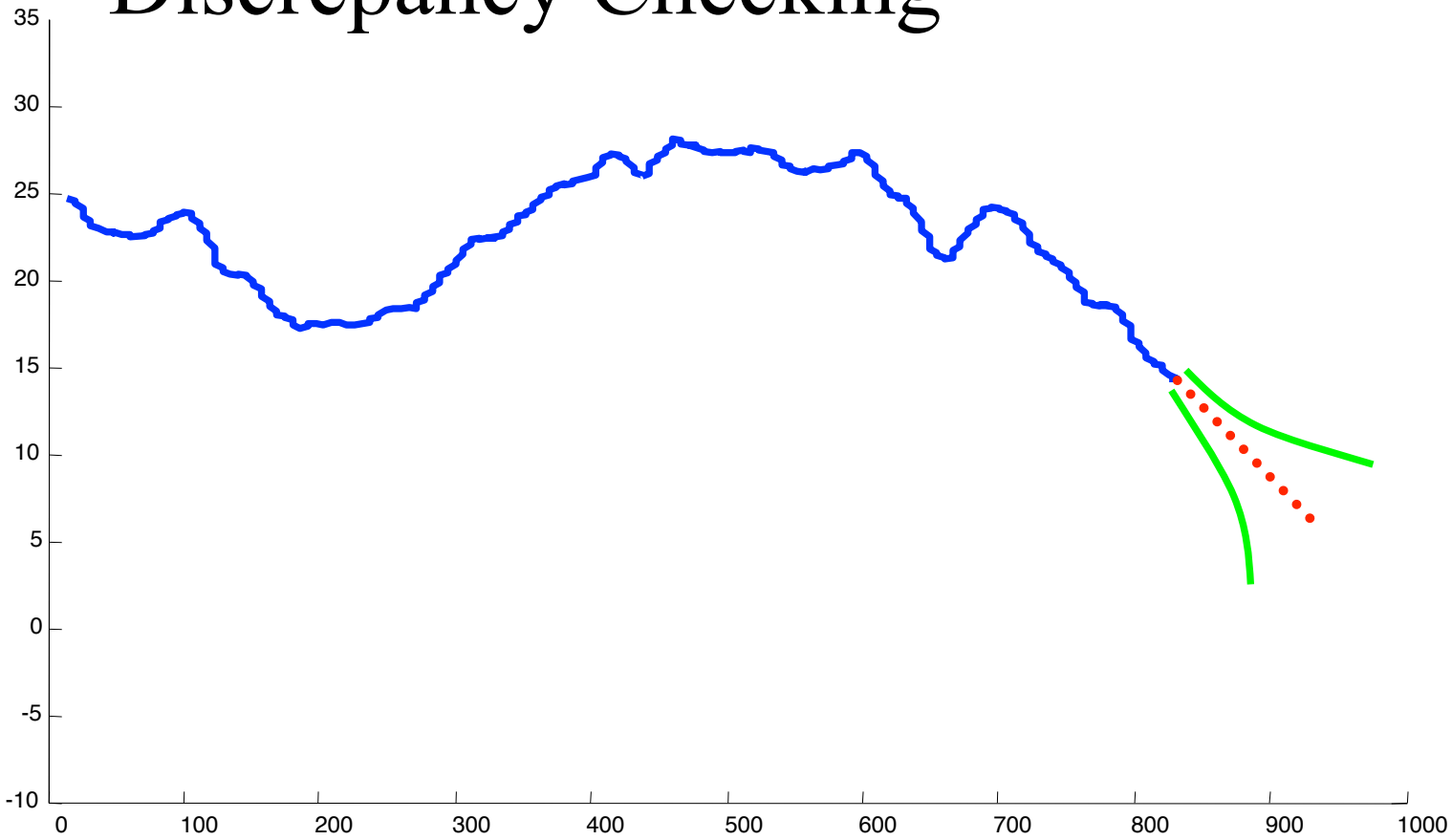
Simple Approaches I

Limit Checking

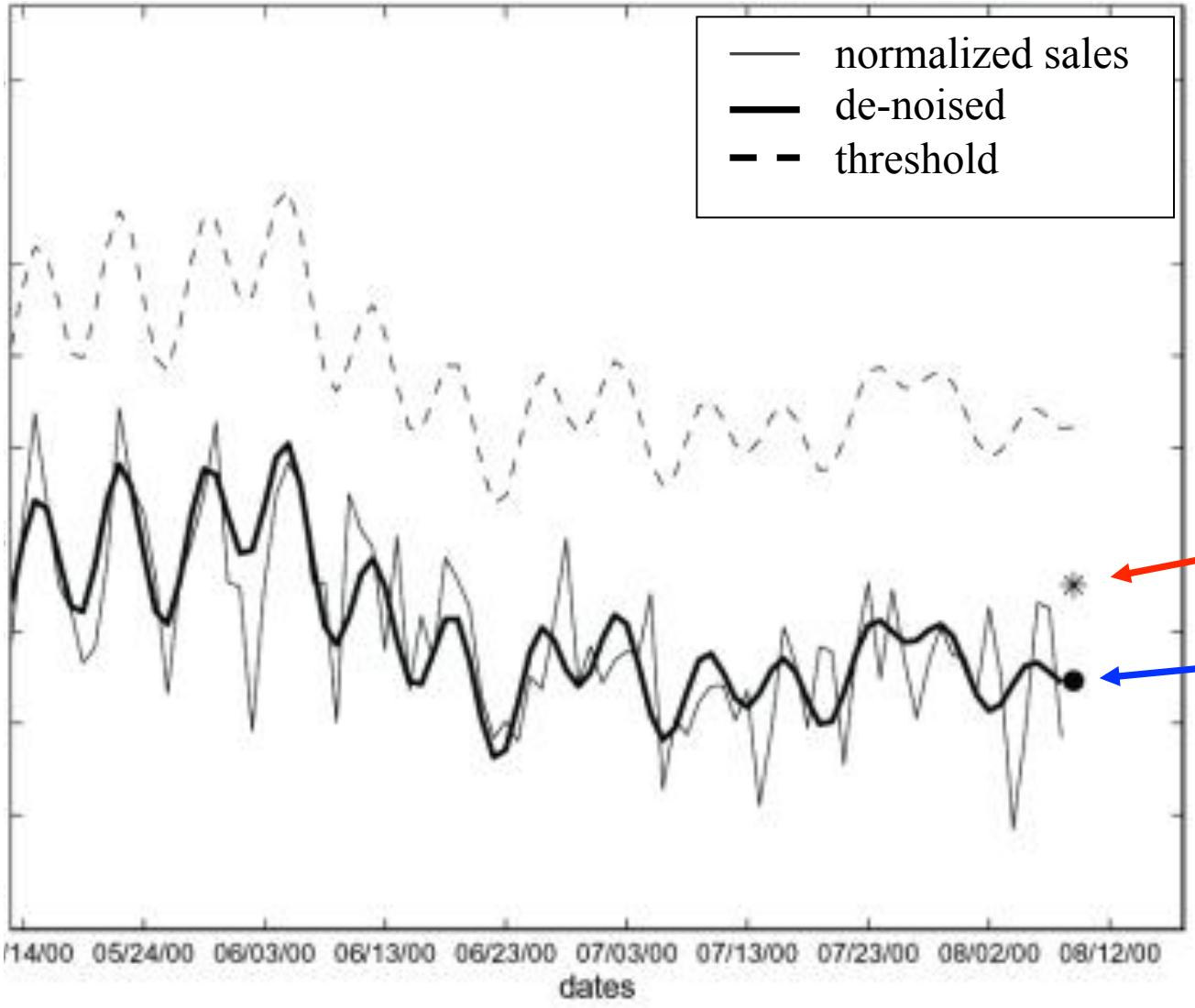


Simple Approaches II

Discrepancy Checking



Discrepancy Checking: Example



Early statistical detection of anthrax outbreaks by tracking over-the-counter medication sales

Goldenberg, Shmueli, Caruana, and Fienberg

Actual value

Predicted value

The actual value is greater than the predicted value, but still less than the threshold, so no alarm is sounded.

Time Series Discord

(Keogh and Lin, 2005)

- Discord: subsequence that is *least* similar to other subsequences
- Applications:
  - Anomaly detection
  - Clustering
  - Data cleaning

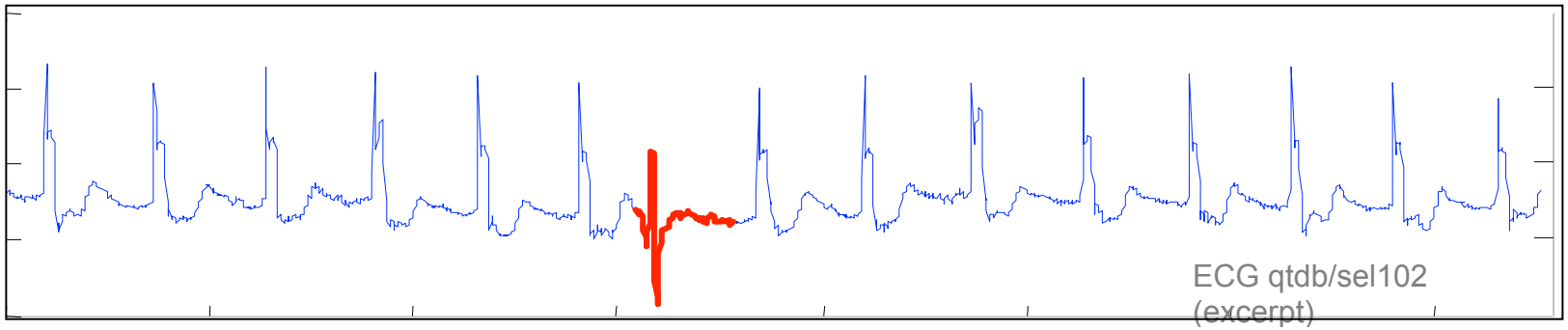


Image Discords

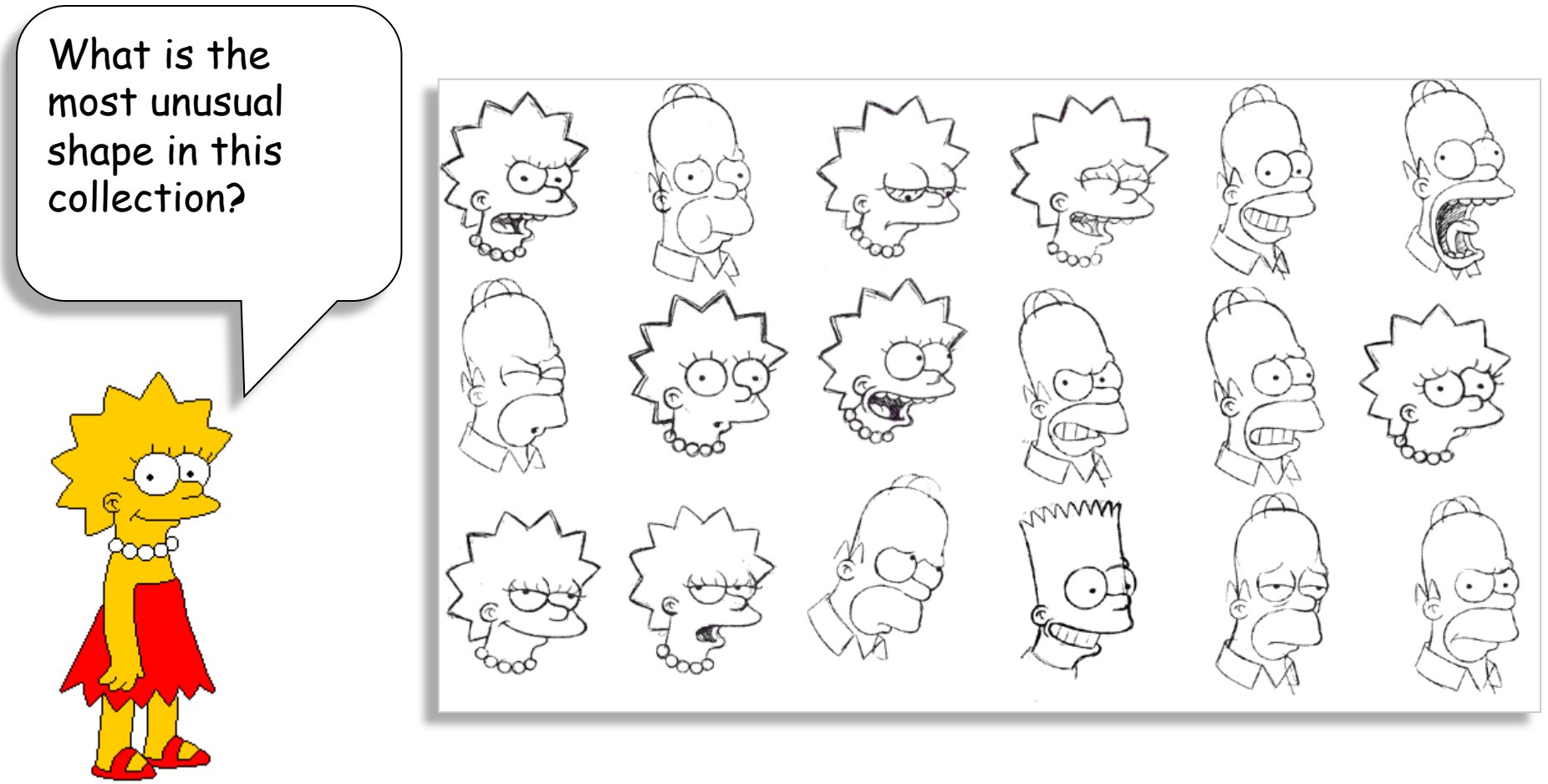
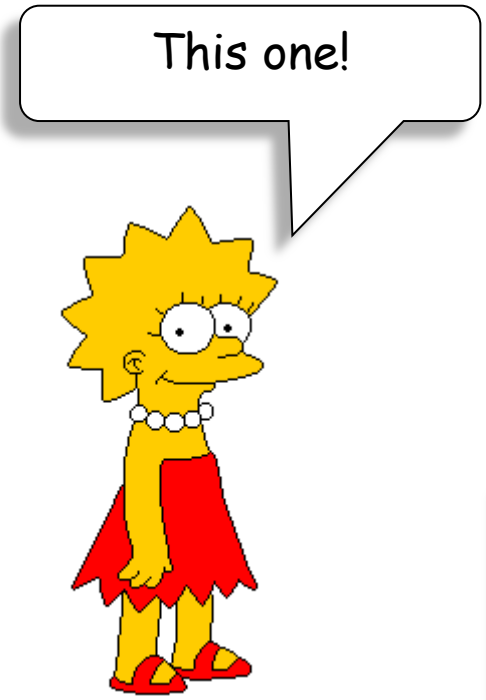
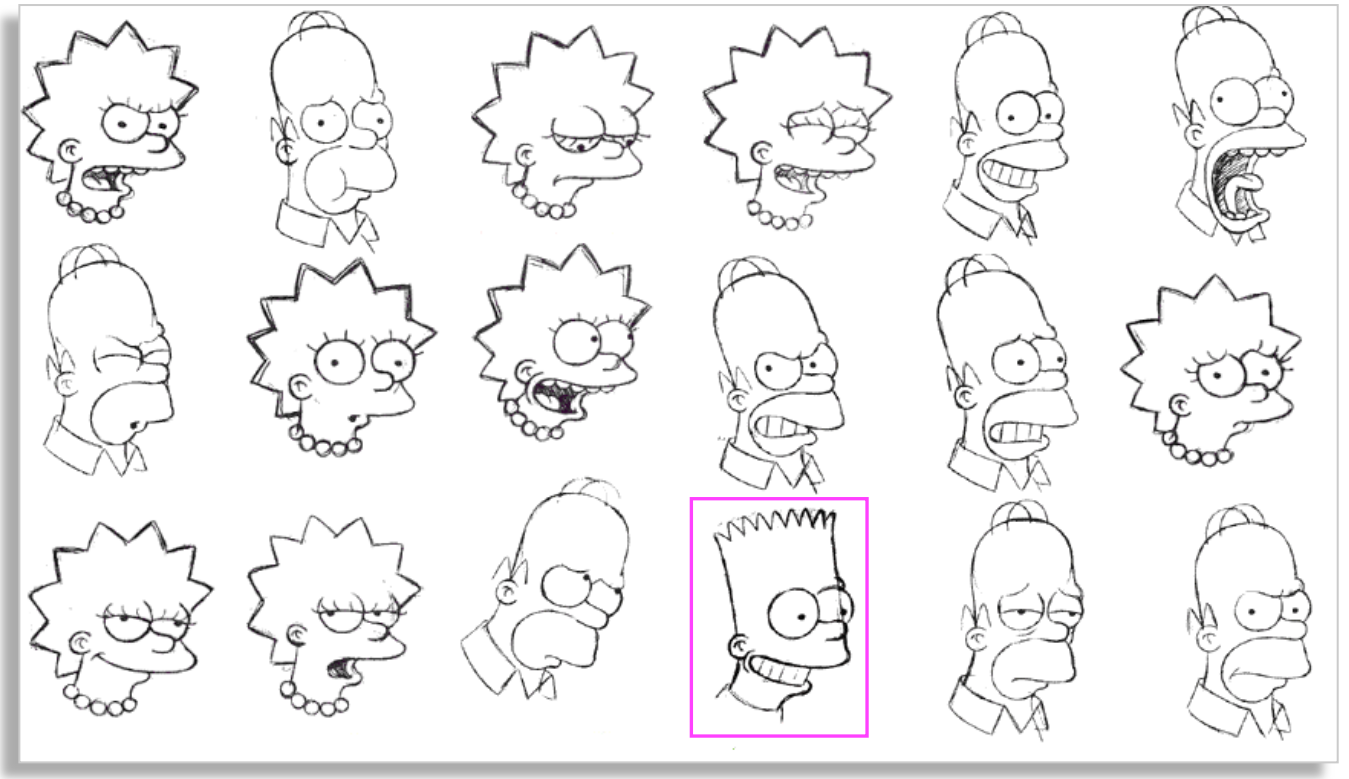


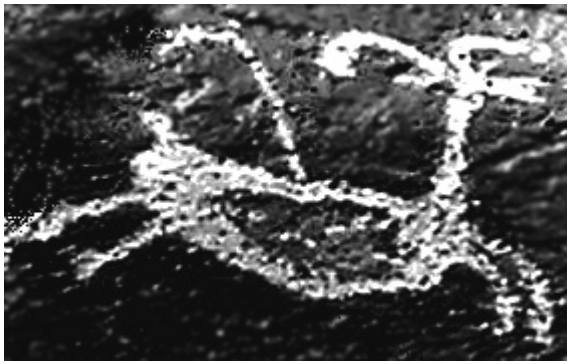
Image Discords

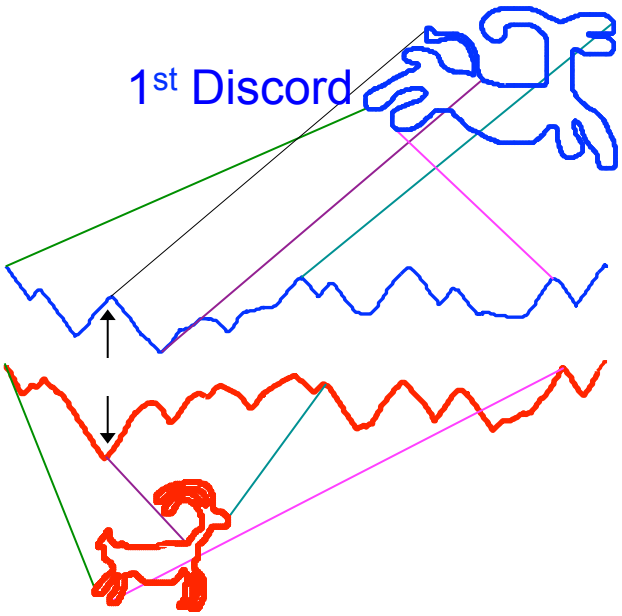


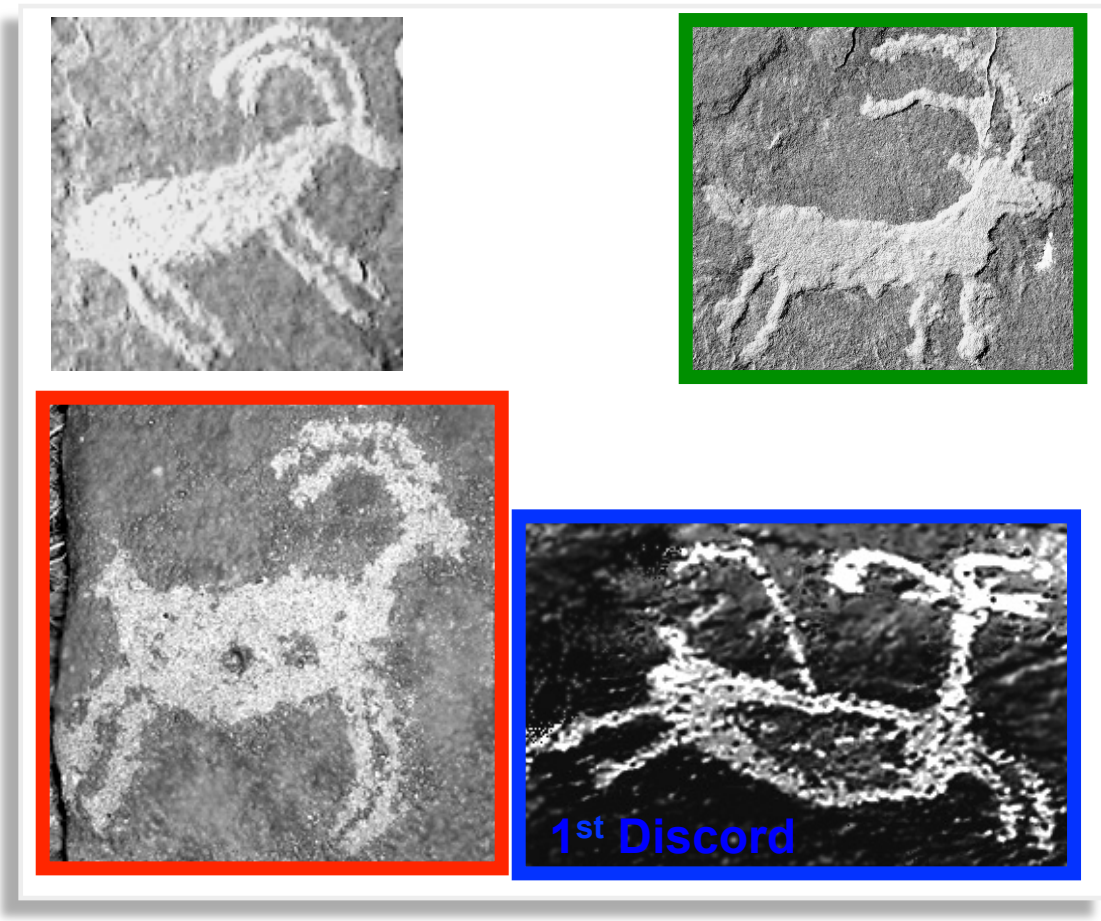


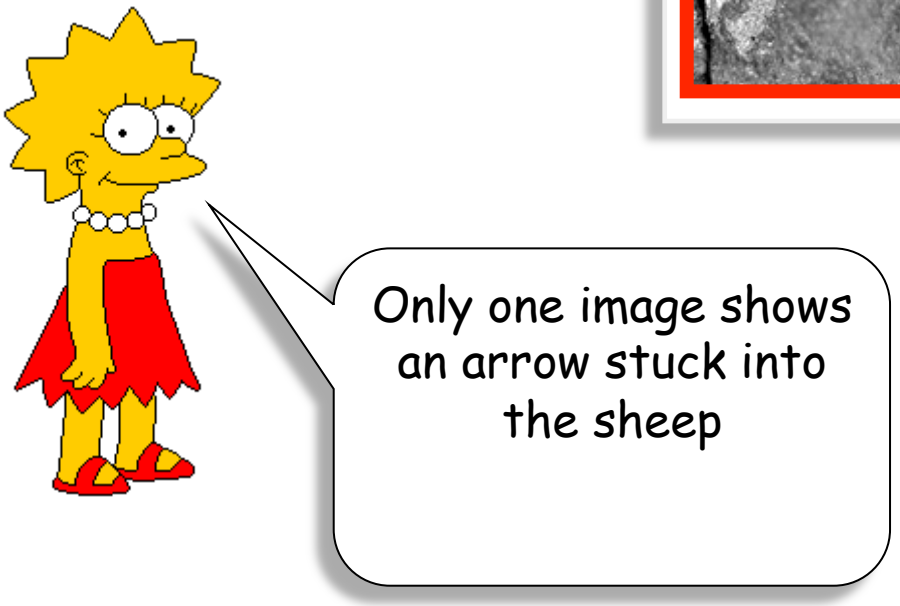
Shape Discord: Given a collection of shapes S, the shape D is the discord of S if D has the largest distance to its nearest match. That is, ∀ shape C in S, the nearest match M_C of C and the nearest match M_D of D, Dist(D, M_D) > Dist(C, M_C).

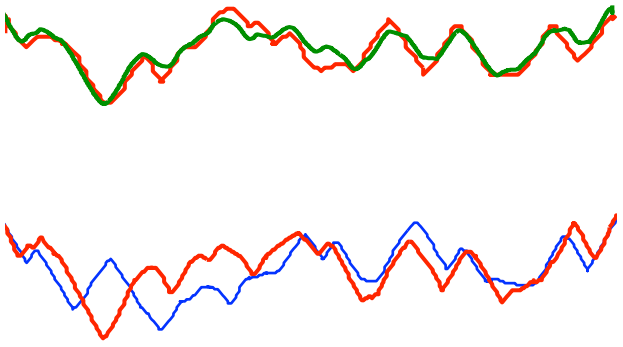
This one is even more subtle… Here is a subset of a large collection of petroglyphs



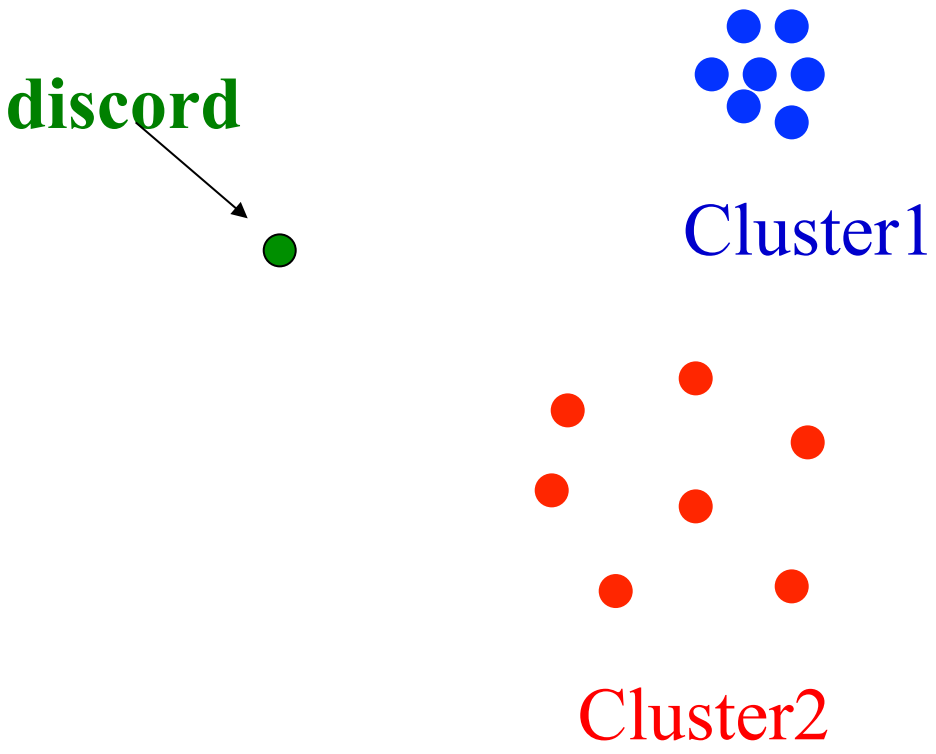

1st Discord


1st Discord


Only one image shows an arrow stuck into the sheep

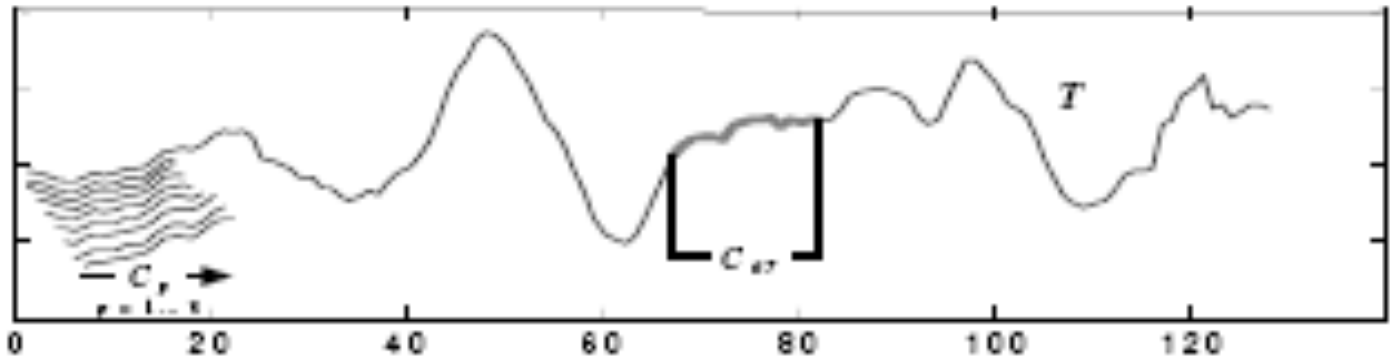


Discord Example



Background – Sliding Windows

- Use a sliding window to extract subsequences



Time Series Discords

- Subsequence *C* of length *n* is said to be the discord if *C* has the largest distance to its nearest non-self match.
- K^{th} Time Series Discord

Finding Discords: Brute-force

- [outer loop] For each subsequence in the time series, [inner loop] find the distance to its nearest match
- The subsequence that has the greatest such value is the discord (i.e. discord is the subsequence with the farthest nearest-neighbor)
- $O(m^2)$

Finding Discords

```
Function  [ dist, loc ] = Discord_Search(S)
best_so_far_dist = 0
best_so_far_loc = NaN
for p  = 1 to size (S)                                  // begin outer loop
  nearest_neighbor_dist = infinity
  for q  = 1 to size (S)                                // begin inner loop
    if p!= q                                            // Don't compare to self
          if  RD(Cp , Cq )  < nearest_neighbor_dist
             nearest_neighbor_dist = RD(Cp , Cq )
          end
    end
  end                                                   // end inner loop
   if nearest_neighbor_dist > best_so_far_dist
     best_so_far_dist = nearest_neighbor_dist
     best_so_far_loc  = p
   end
end                                                     // end outer loop
return [ best_so_far_dist, best_so_far_loc ]
```

0	**2**	4.2	1.1	2.3	8.5
2	0	3	3.2	3.5	8.2
4.2	3	0	1.2	9.2	9.7
1.1	3.2	**1.2**	0	**0.1**	**7.5**
2.3	3.5	9.2	**0.1**	0	7.6
8.5	8.8	9.7	7.5	7.6	0
1.1	2	1.2	0.1	0.1	**7.5**

The code says…
Find the smallest (non diagonal) value in each column, the largest of these is the discord

Finding Discords, Fast

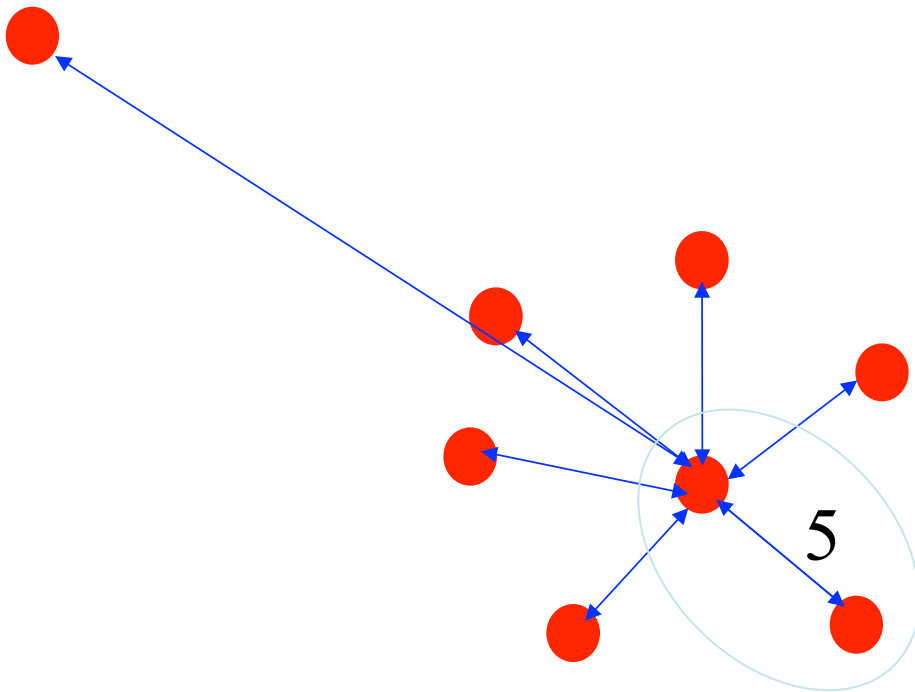
```
Function  [ dist, loc ] = Heuristic_Search(S, Outer, Inner )
best_so_far_dist = 0
best_so_far_loc = NaN
for each index p  given by heuristic Outer   // begin outer loop
   nearest_neighbor_dist = infinity
   for each index q  given  by heuristic Inner // begin inner loop
      if p!= q
         if  RD(C_p , C_q )  < best_so_far_dist
            break                                  // break out of inner loop
         end
         if  RD(C_p , C_q )  < nearest_neighbor_dist
            nearest_neighbor_dist = RD(C_p , C_q )
        end
      end
   end                                             // end inner loop
    if nearest_neighbor_dist > best_so_far_dist
       best_so_far_dist = nearest_neighbor_dist
       best_so_far_loc  = p
    end
end                                                // end outer loop
return [ best_so_far_dist, best_so_far_loc ]
```

0	**2**	4.2	1.1	2.3	8.5
2	0	3	3.2	3.5	8.2
4.2	3	0	1.2	9.2	9.7
1.1	3.2	**1.2**	0	**0.1**	**7.5**
2.3	3.5	9.2	**0.1**	0	7.6
8.5	8.8	9.7	7.5	7.6	0

The code now says… If while searching a given column, you find a distance less than nearest_neighbor_dist then that column cannot have the discord.

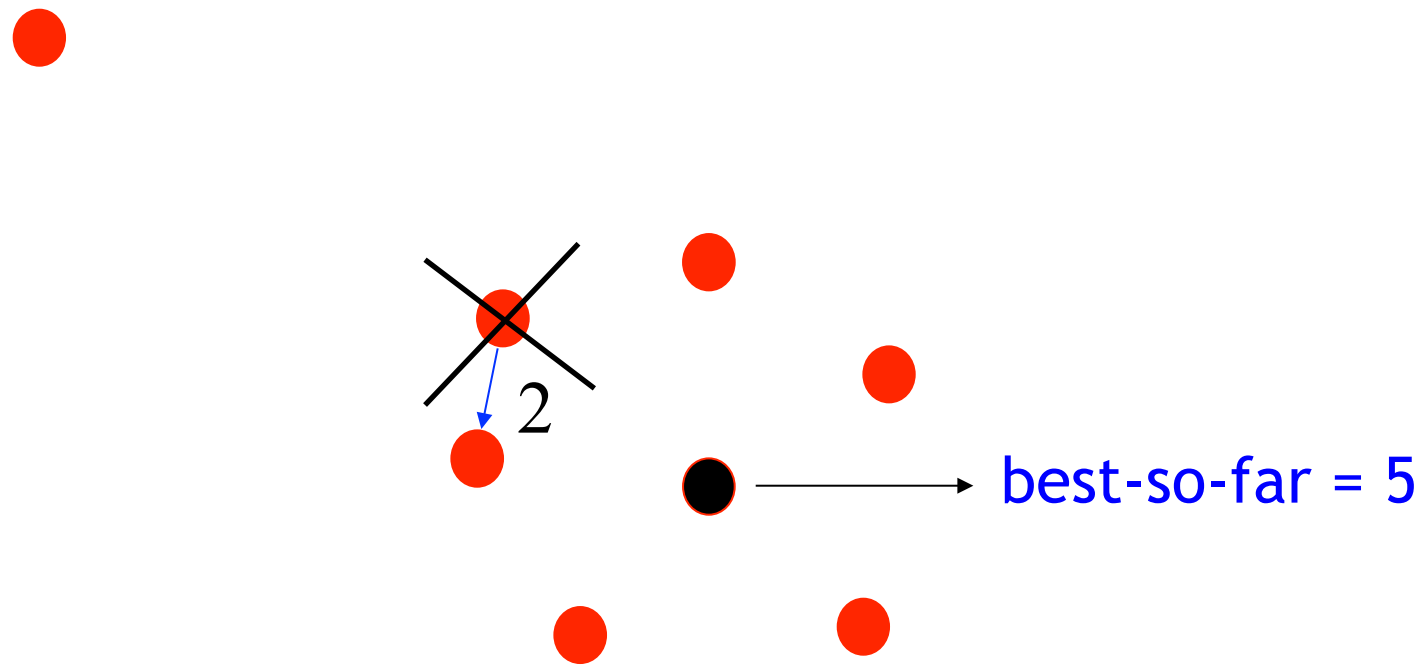
The code also uses heuristics to order the search…

Example

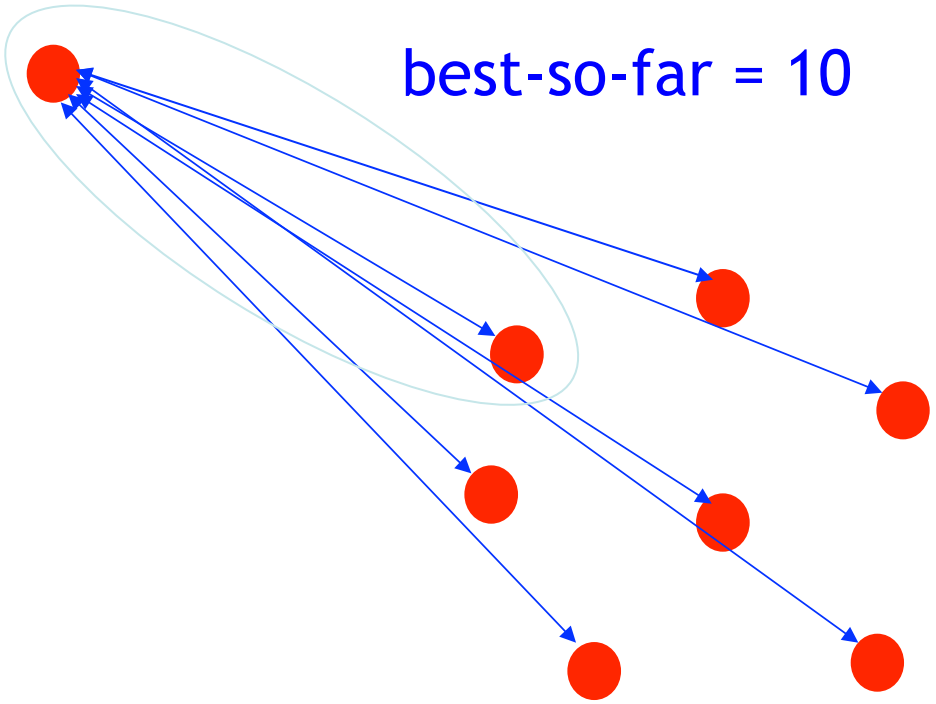


best-so-far = 5

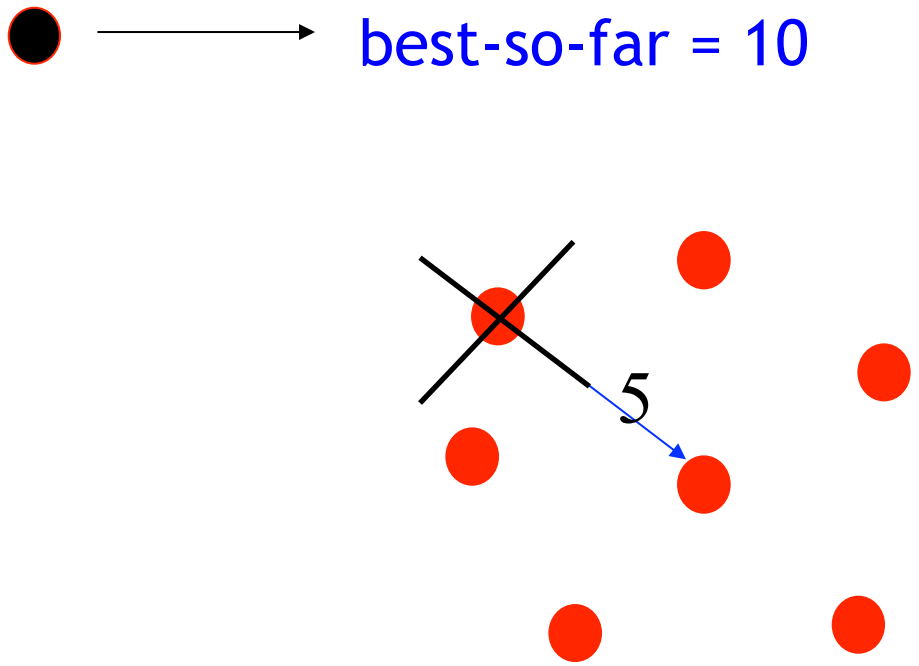
Example



Example – Optimal Ordering



Example – Optimal Ordering



The Magic Heuristics

- In the outer loop, visit the columns in order of the Discord score
- In the inner loop, visit the row cells in order of nearest neighbor first

0	**2**	4.2	1.1	2.3	8.5
2	0	3	3.2	3.5	8.2
4.2	3	0	1.2	9.2	9.7
1.1	3.2	**1.2**	0	**0.1**	**7.5**
2.3	3.5	9.2	**0.1**	0	7.6
8.5	8.8	9.7	7.5	7.6	0

The Magic Heuristics would reduce the time complexity from $O(n^2)$ algorithm to just $O(n)$!

The Magic Heuristics

- In the outer loop, visit the columns in order of the Discord score
- In the inner loop, visit the row cells in order of nearest neighbor first

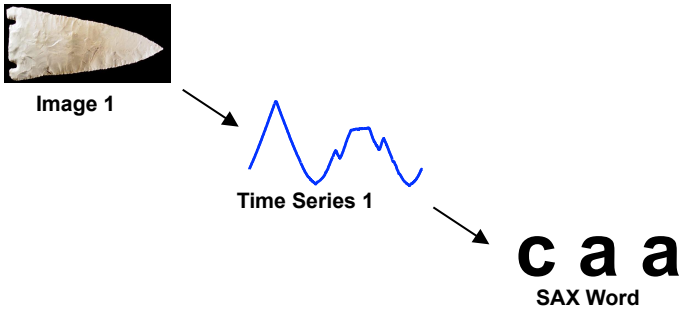
0	2	4.2	1.1	2.3	8.5
2	0	3	3.2	3.5	8.2
4.2	3	0	1.2	9.2	9.7
1.1	3.2	1.2	0	0.1	7.5
2.3	3.5	9.2	0.1	0	7.6
8.5	8.8	9.7	7.5	7.6	0

Observations

- Visiting the columns in approximately order of the Discord score is still very helpful
- For the inner loop, we don't really need visit the rows in order of nearest neighbor first, so long as we find a "near enough" neighbor early on

Approximately Magic Heuristics

0	2	4.2	1.1	2.3	8.5
2	0	3	3.2	3.5	8.2
4.2	3	0	1.2	9.2	9.7
1.1	3.2	1.2	0	0.1	7.5
2.3	3.5	9.2	0.1	0	7.6
8.5	8.8	9.7	7.5	7.6	0

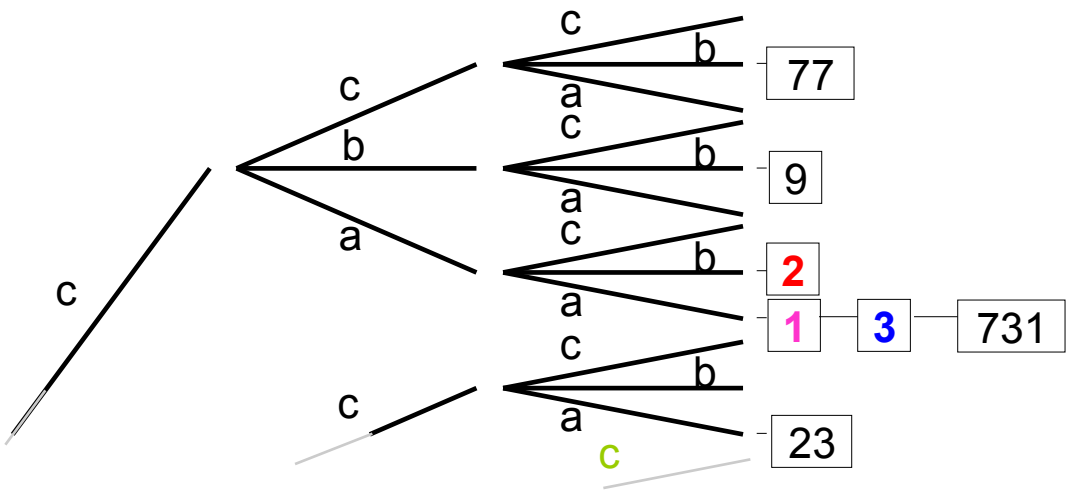


Rotation invariance ignored here

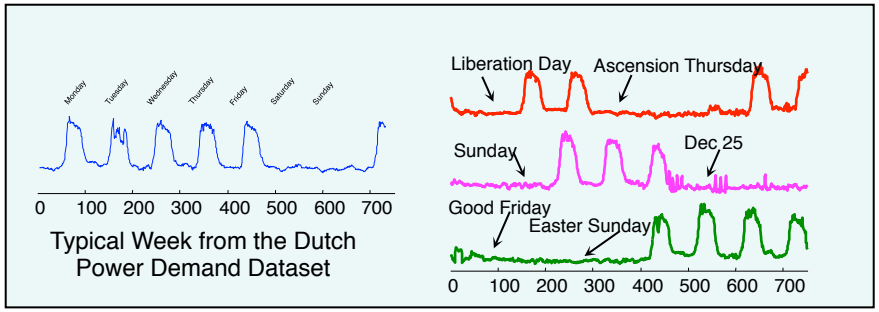
Inserted into array

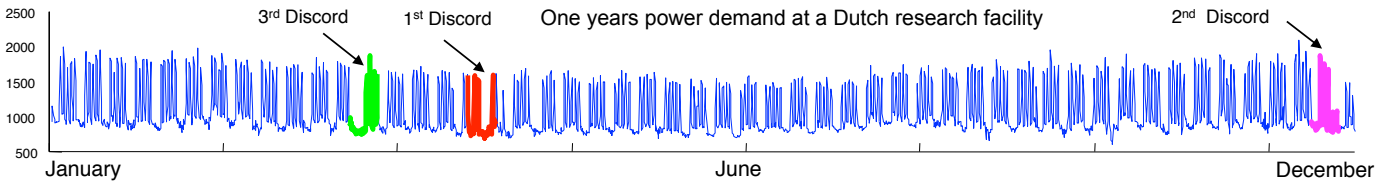
1	c	a	a	3
2	c	a	b	1
3	c	a	a	3
::	::	::	::	::
::	::	::	::	::
	c	b	b	2
m-1	a	c	b	1
m	b	c	a	2

Augmented Trie

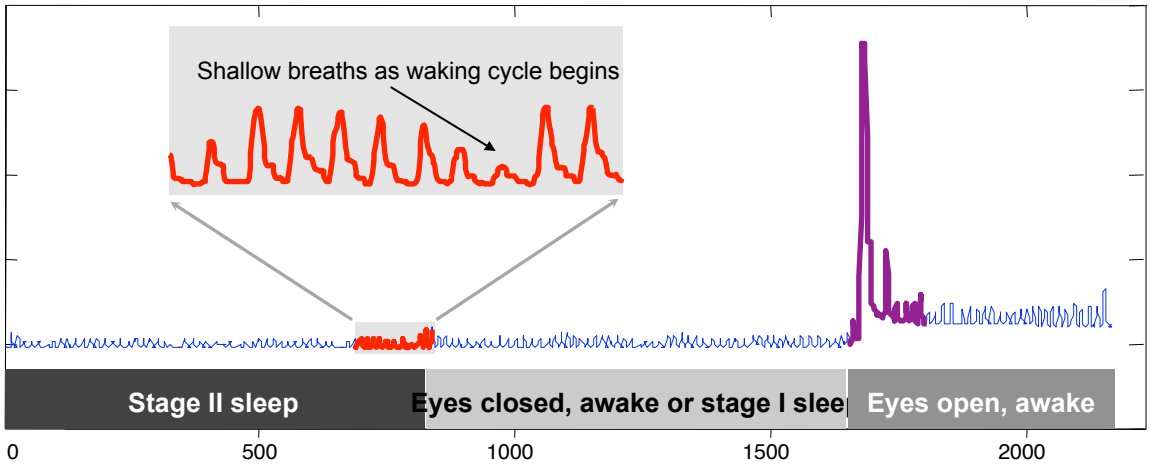


Time Series Discords





Power Demand



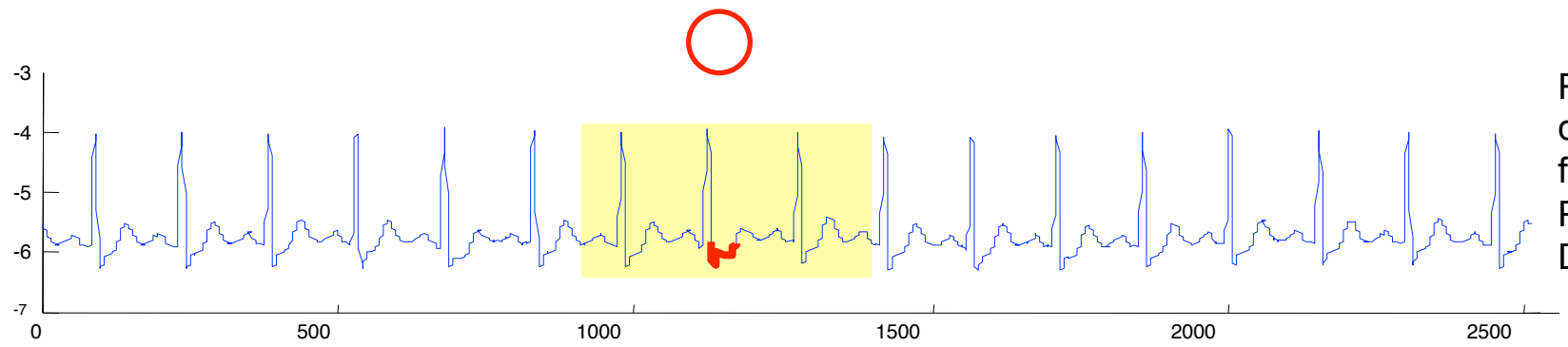
Sleep Cycles

A time series showing a patients respiration (measured by thorax extension), as they wake up. A medical expert, Dr. J. Rittweger, manually segmented the data. The 1-discord is a very obvious deep breath taken as the patient opened their eyes. The 2-discord is much more subtle and impossible to see at this scale. A zoom-in suggests that Dr. J. Rittweger noticed a few shallow breaths that indicated the transition of sleeping stages.

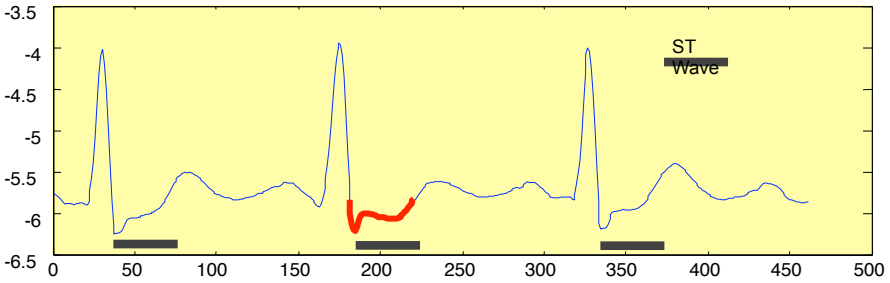
Institute for Physiology. Free University of Berlin. Data shows respiration (thorax extension), sampling rate 10 Hz.

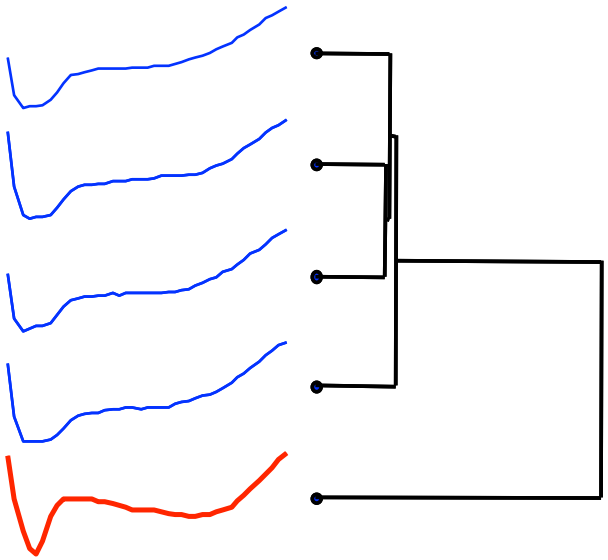
Discords in Medical Data

A cardiologist noted subtle anomalies in this dataset. Let us see if the discord algorithm can find them.

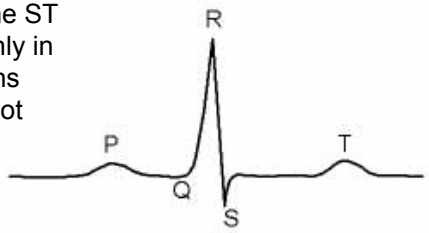


Record qtdbsele0606 from the PhysioBank QT Database (qtdb



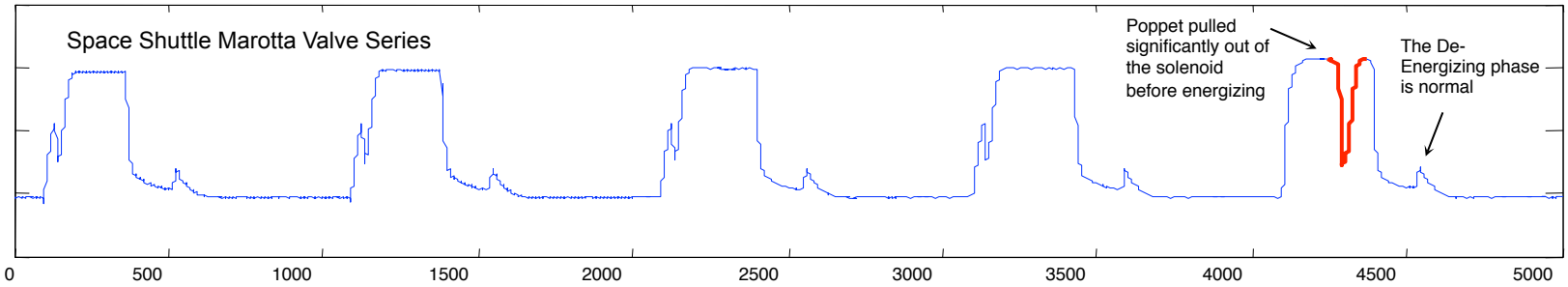


How was the discord able to find this very subtle Premature ventricular contraction? Note that in the normal heartbeats, the ST wave increases monotonically, it is only in the Premature ventricular contractions that there is an inflection.NB, this is not necessary true for all ECGS

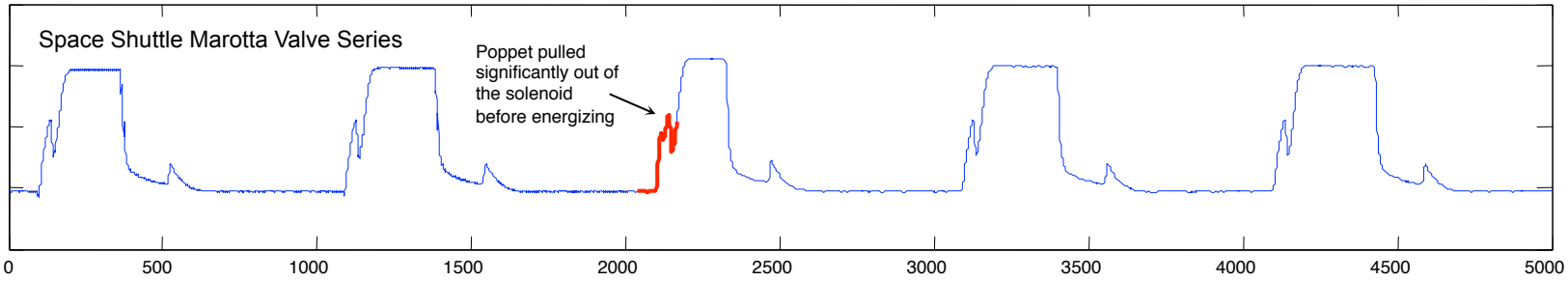


Discords in Space Shuttle Marotta Valve Series

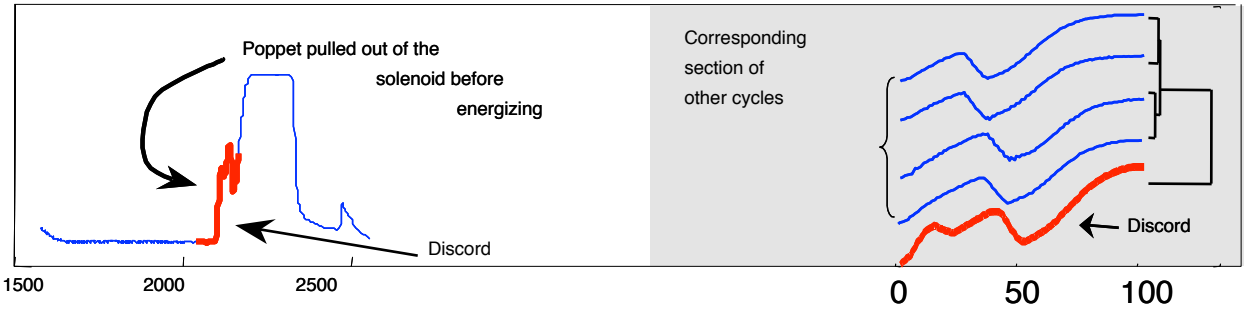
Example One

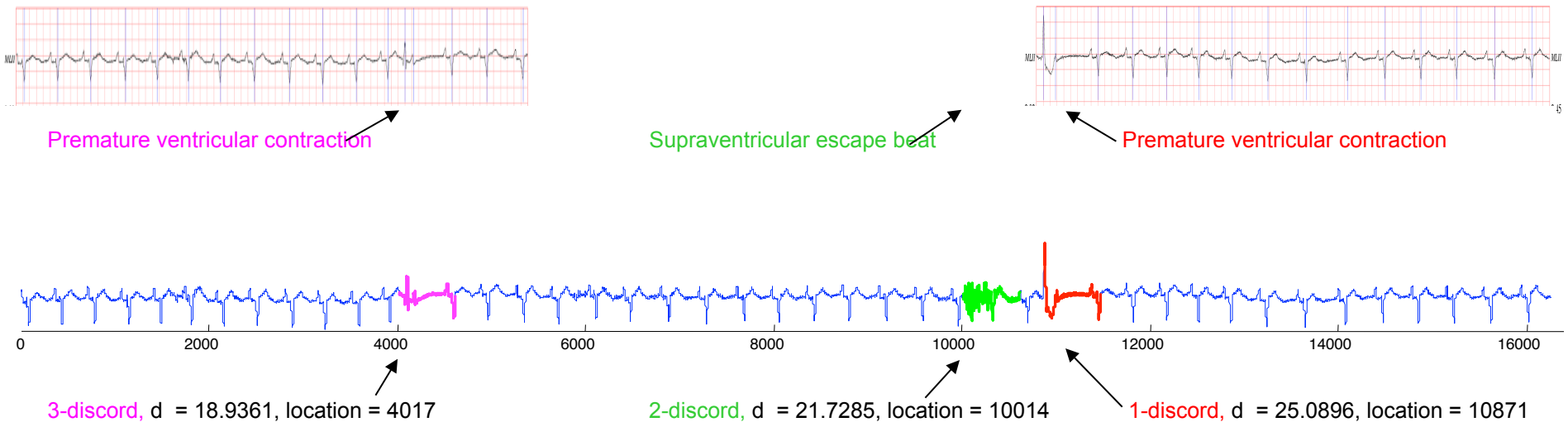


Example Two

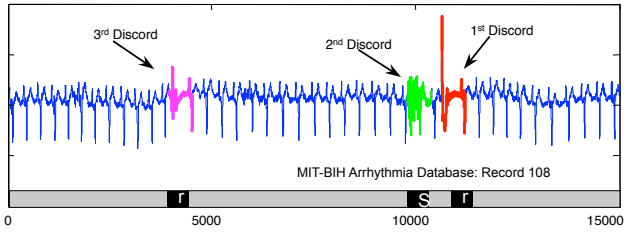


This discord is subtle, lets zoom in to see why it is a discord.





The time series is record mitdb/x_mitdb/x_108 from the PhysioNet Web Server (The local copy in the UCR archive is called mitdbx_mitdbx_108.txt). It is a two feature time series, here we are looking at just the MLII column.
Cardiologists from MIT have annotated the time series, here we have added colored makers to draw attention to those annotations.
Here we show the results of finding the top 3 discords on this dataset. We chose a length of 600, because this a little longer than the average length of a single heartbeat.



Anomaly (interestingness) detection

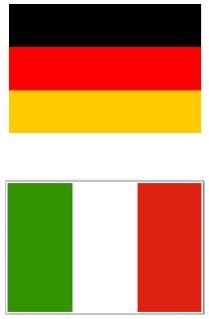
In spite of the nice example in the previous slide, the anomaly detection problem is wide open.

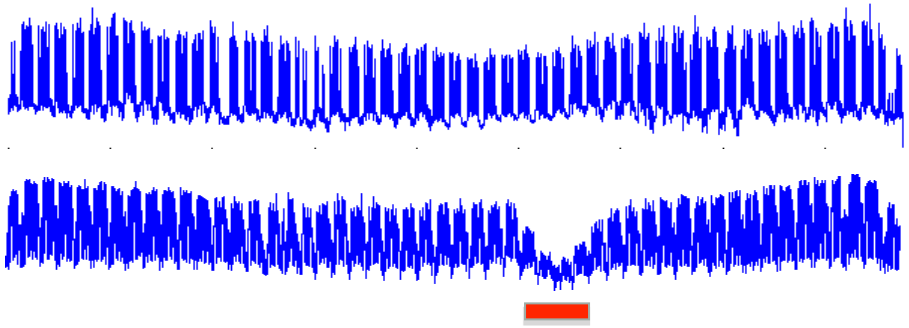
How can we find interesting patterns…

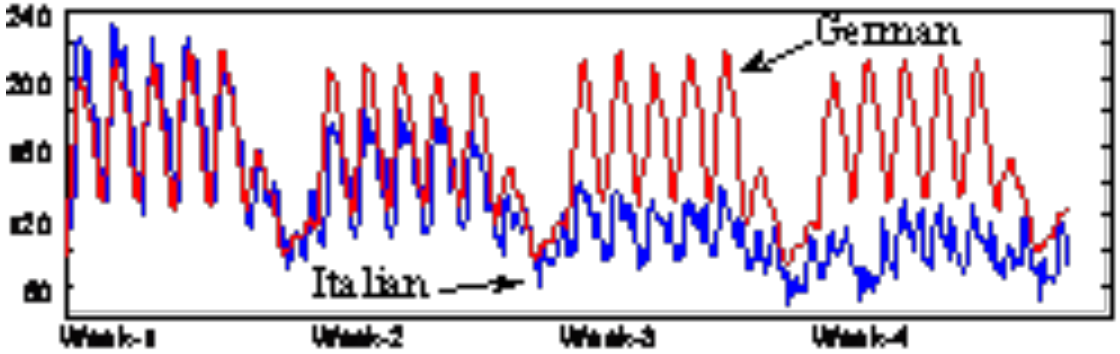
- Without (or with very few) false positives…
- In truly massive datasets...
- In the face of concept drift…
- With human input/feedback…
- With annotated data…

Contrast-Set Mining

A motivating example: What differentiates German and Italian consumer electrical power demands?







Jessica Lin and Eamonn Keogh. 2006. Group SAX: Extending the notion of contrast sets to time series and multimedia data. In Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases. Berlin, Germany. Sept 18-22. Pages 284-296. Lecture Notes in Computer Science, Springer.