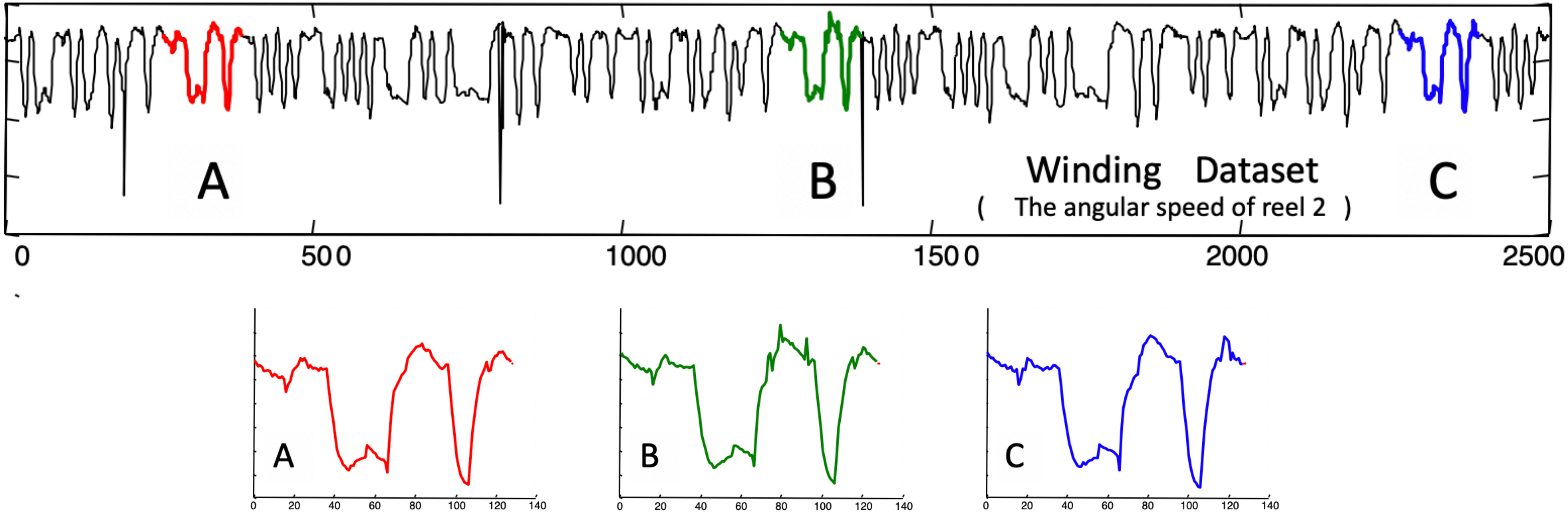
Time Series - Shapelet/Motif Discovery

UNIVERSITÀ DI PISA

Motif

Time Series Motif Discovery

- Finding repeated patterns, i.e., pattern mining.
- Are there any repeated patterns, of length *m* in the TS?



Why Find Motifs?

- Mining **association rules** in TS requires the discovery of motifs. These are referred to as primitive shapes and frequent patterns.
- Several **TS classifiers** work by constructing typical prototypes of each class. These prototypes may be considered motifs.
- Many **TS anomaly detection** algorithms 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.

How do we find Motifs?

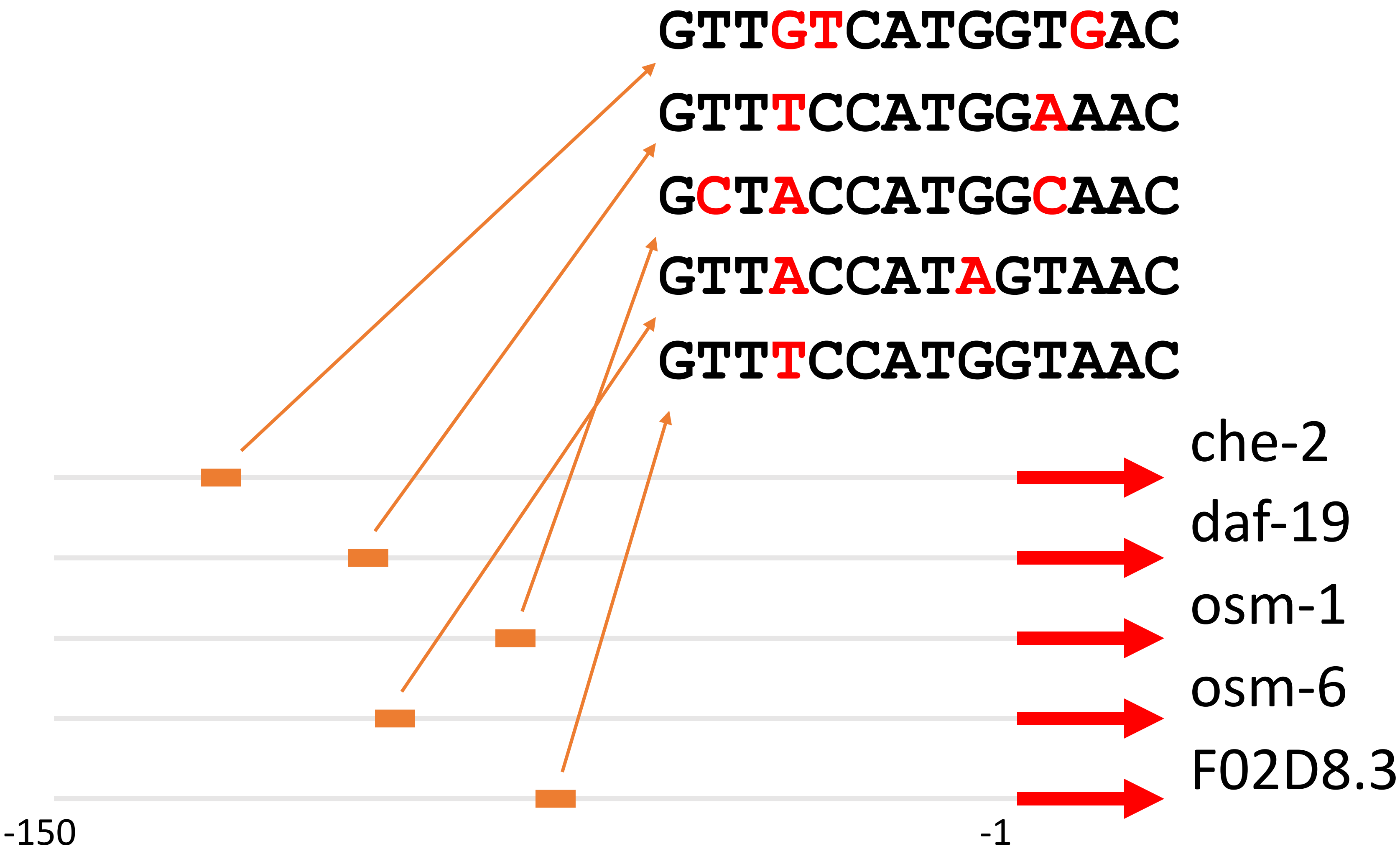
- Given a predefined motif length *m*, a brute-force method searches for motifs from all possible comparisons of subsequences.

- It is obviously very slow and computationally expensive.

- The most reference algorithm is based on a hot idea from bioinformatics, random projection* and the fact that SAX allows to use lower bound discrete representations of TSs.

*J Buhler and M Tompa. Finding motifs using random projections. In RECOMB'01. 2001.

Motif in Bioinformatics



GTTGTCATGGTGAC

GTTTCCATGGAAAC

GCTACCATGGCAAC

GTTACCATAGTAAC

GTTTCCATGGTAAC

che-2

daf-19

osm-1

osm-6

F02D8.3

-150

-1

The Motif Discovery Algorithm

- **EThe general problem:**
  - Find the motif M by using a set of sequences called *(w,d)-motif:* sequence of length w that differ from a *d* points
- **Guiding principle:**
  - Some instances of a motif agree on a subset of positions.
- **Use information from multiple motif instances to construct model**

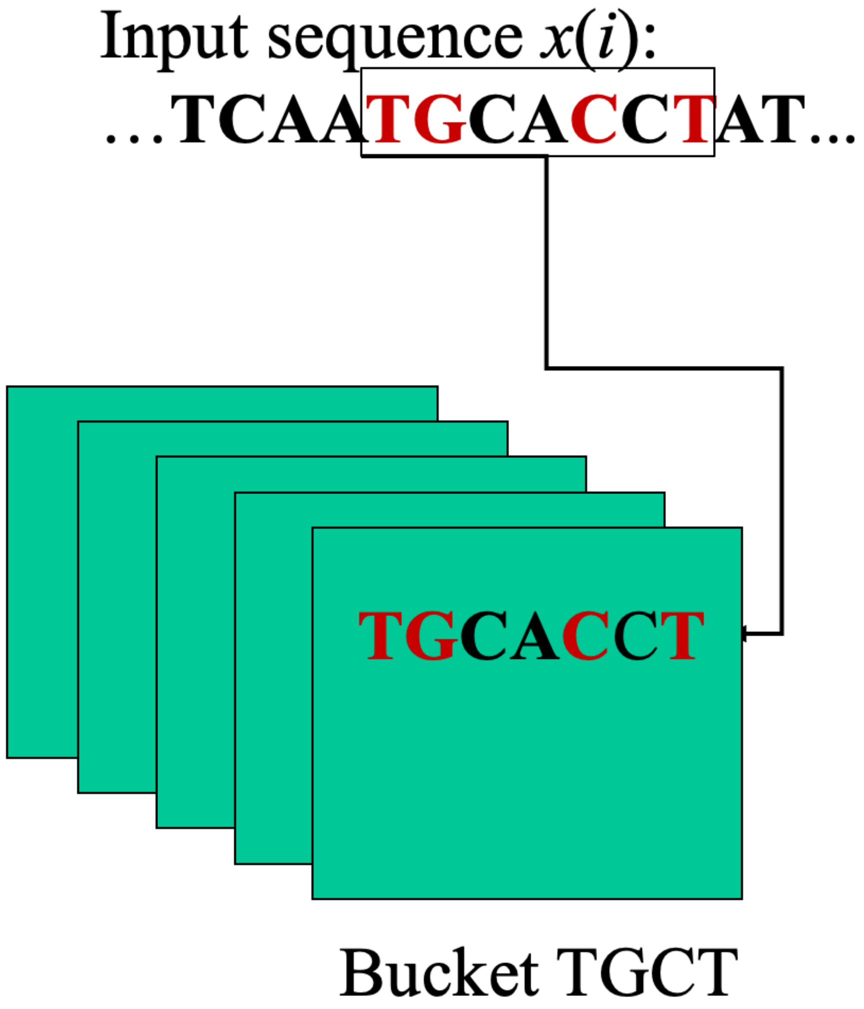
k-Projections

- Choose *k* positions in string of length *l*.
- Concatenate elements at chosen *k* positions to form *k*-tuple.
- In *l*-dimensional Hamming space, projection onto *k* dimensional subspace.

P = (2, 4, 5, 7, 11, 12, 13)

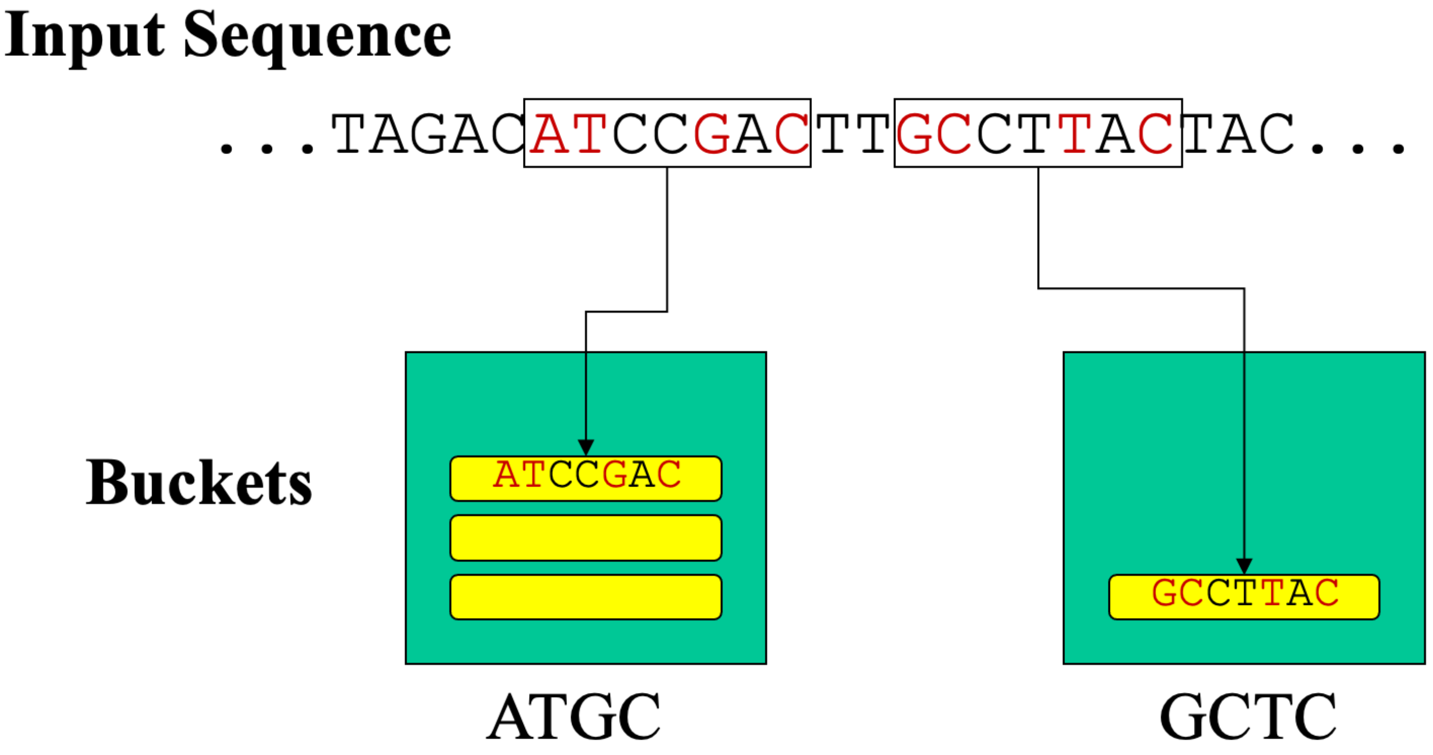
Random Projection Algorithm

- Choose a projection by **selecting *k* positions uniformly at random**.
- For each *l*-tuple in input sequences, **hash into bucket** based on letters at *k* selected positions.
- **Recover motif from bucket** containing multiple *l*-tuples.



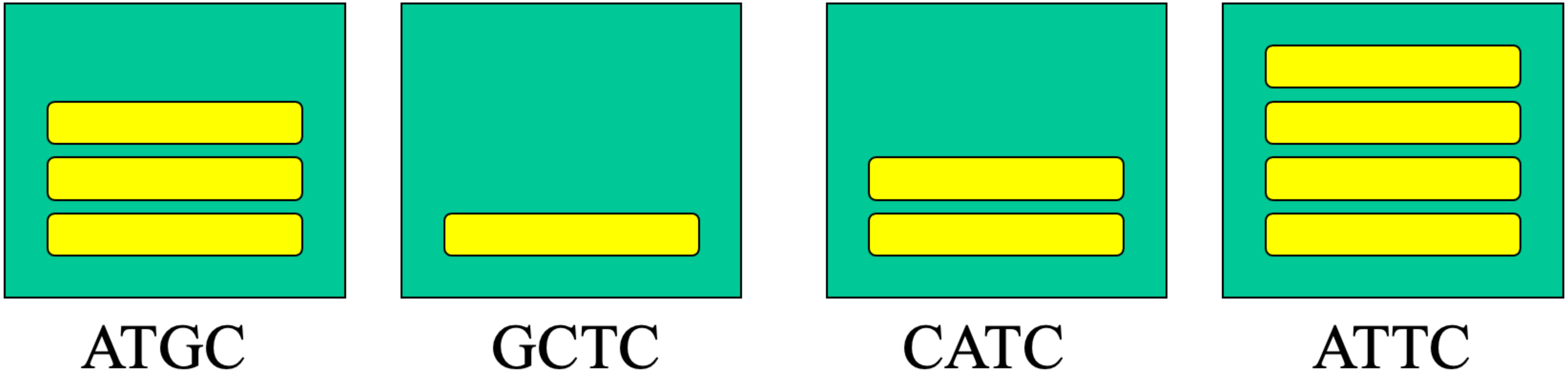
Example

- *l* = 7 (motif size) , *k* = 4 (projection size)
- Choose projection (1,2,5,7)



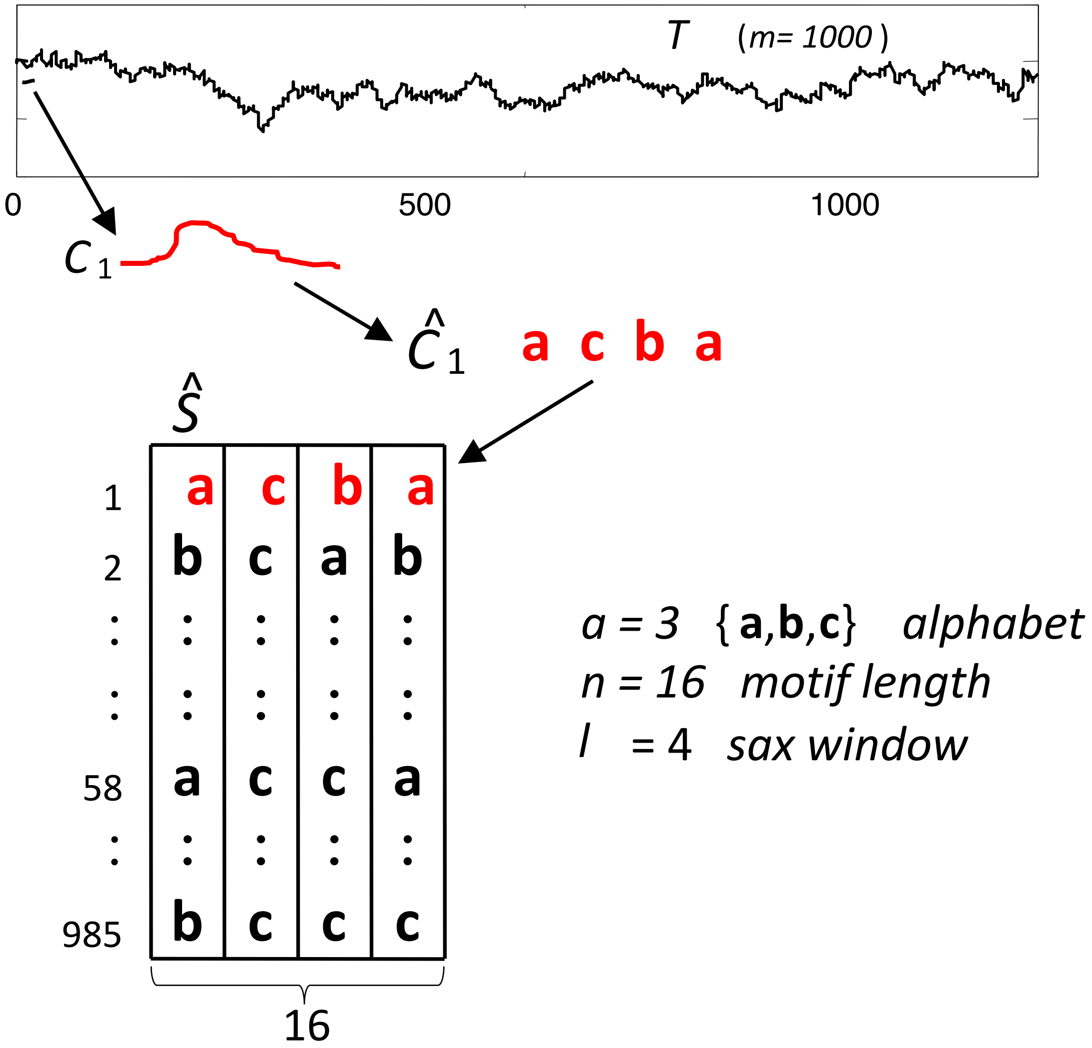
Hashing and Buckets

- Hash function h(*x*) obtained from *k* positions of projection.
- Buckets are labeled by values of h(*x*).
- *Enriched buckets*: **contain at least *s*** *l*-tuples, for some parameter *s.*



Example of the Motif Discovery Algorithm

- 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}.

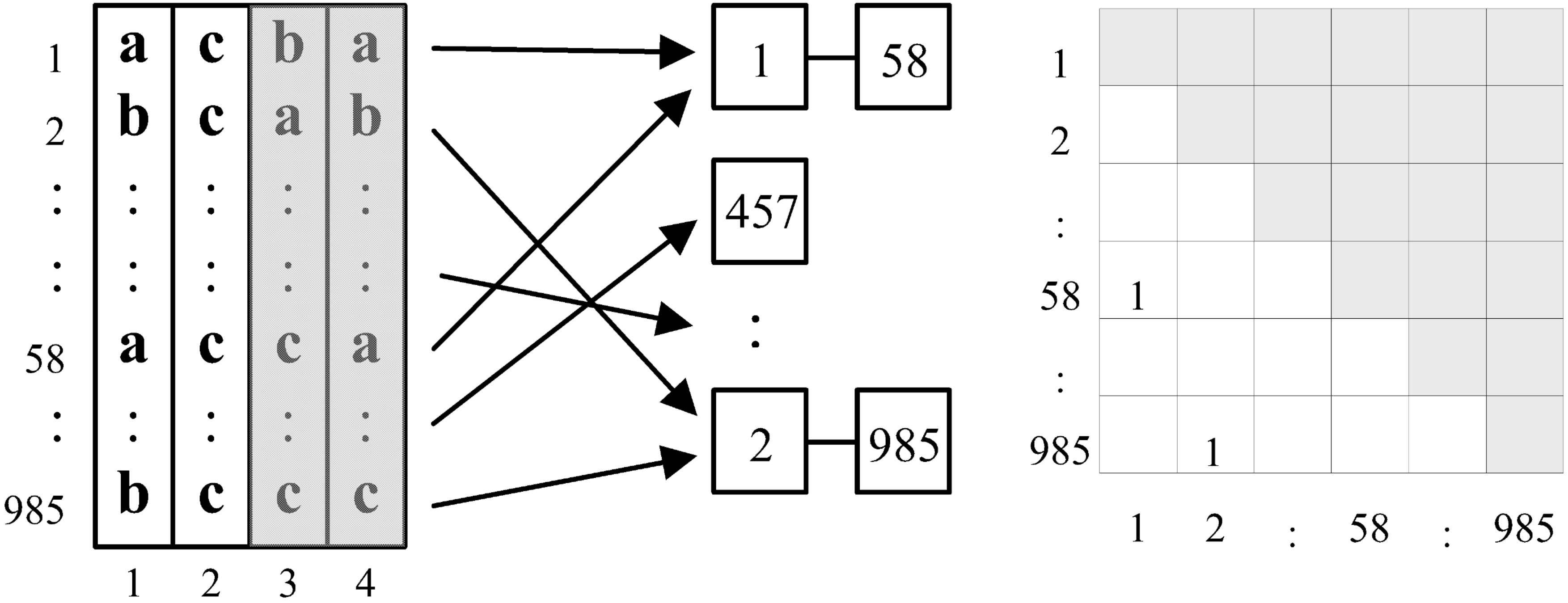


$\hat{S}$				
1	a	c	b	a
2	b	c	a	b
:	:	:	:	:
:	:	:	:	:
58	a	c	c	a
:	:	:	:	:
985	b	c	c	c

a = 3 {**a**,**b**,**c**} *alphabet*
n = 16 *motif length*
l = 4 *sax window*

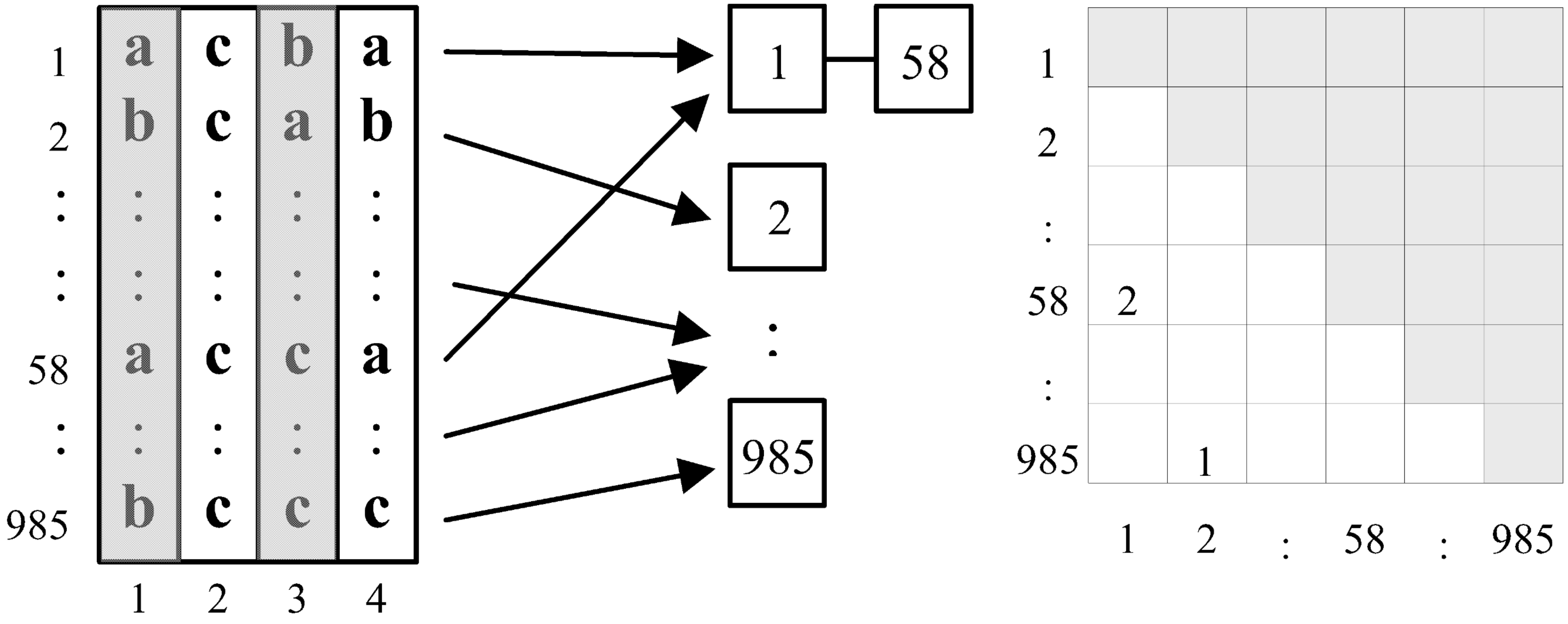
Example of the Motif Discovery Algorithm

- 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.



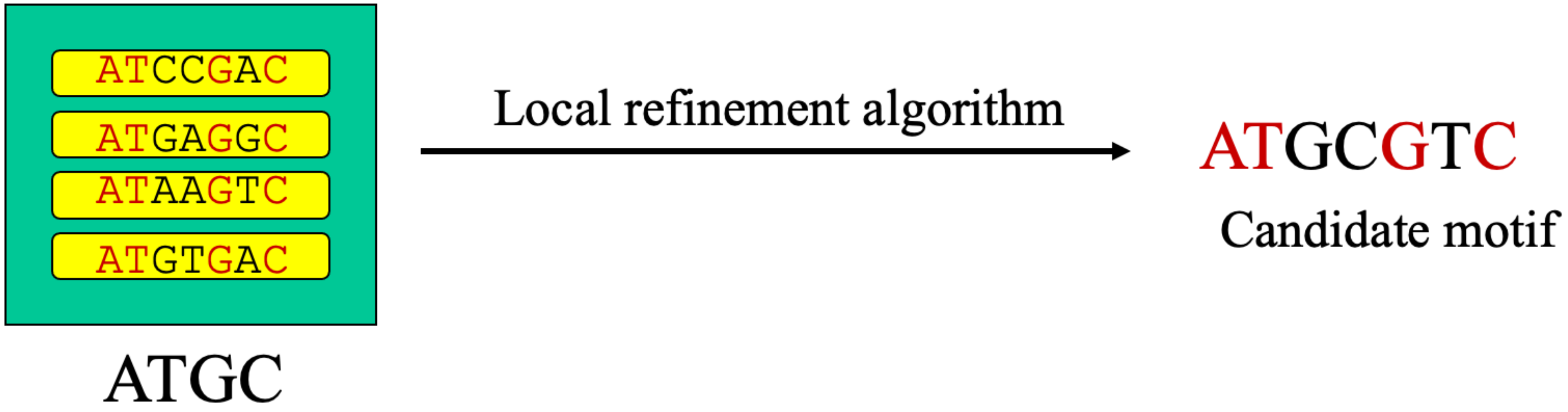
Example of the Motif Discovery Algorithm

- 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.



Motif Refinement

- How do we recover the motif from the sequences in the enriched buckets?
- *k* symbols are known from hash value of bucket.
- Use information in other *l*-*k* positions as starting point for local refinement scheme, e.g. EM



ATCCGAC

ATGAGGC

ATAAGTC

ATGTGAC

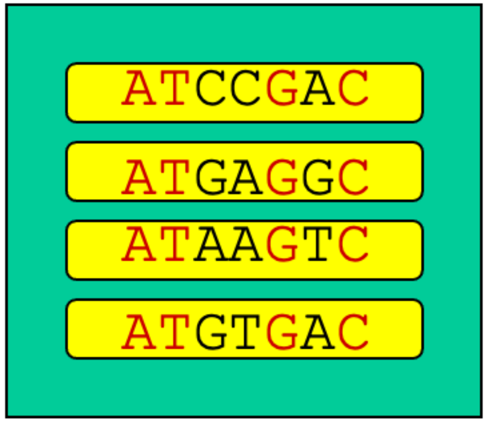
ATGC

Local refinement algorithm

ATGCGTC

Candidate motif

Frequency Matrix Model from Bucket



ATGC

$$\begin{array}{c} A \\ C \\ G \\ T \end{array} \begin{pmatrix} 1 & 0 & .25 & .5 & 0 & .5 & 0 \\ 0 & 0 & .25 & .25 & 0 & 0 & 1 \\ 0 & 0 & .5 & 0 & 1 & .25 & 0 \\ 0 & 1 & 0 & .25 & 0 & .25 & 0 \end{pmatrix}$$

Frequency matrix W

EM algorithm

Refined matrix W*

EM Motif Refinement

- For each bucket h containing more than s sequences, form weight matrix $\mathbf{W}_h$
- Use EM algorithm with starting point $\mathbf{W}_h$ to obtain refined weight matrix model $\mathbf{W}_h^*$
- For each input sequence $x(i)$, return l tuple $y(i)$ which maximizes likelihood ratio:

$$\Pr(y(i) \mid \mathbf{W}_h^*)/\Pr(y(i) \mid P_0).$$

- $\mathcal{T} = \{y(1), y(2), \ldots, y(N)\}$
- $C(\mathcal{T})$ = consensus string

Expectation Maximization (EM)

- $S = \{ x(1), \ldots, x(N)\}$: set of input sequences
- Given:
  - W = An initial probabilistic motif model
  - P_0 = background probability distribution.
- Find value W_{max} that maximizes likelihood ratio:

$$\frac{\Pr(S \mid W_{\max}, P_0)}{\Pr(S \mid P_0)}$$

- EM is local optimization scheme. Requires starting value W

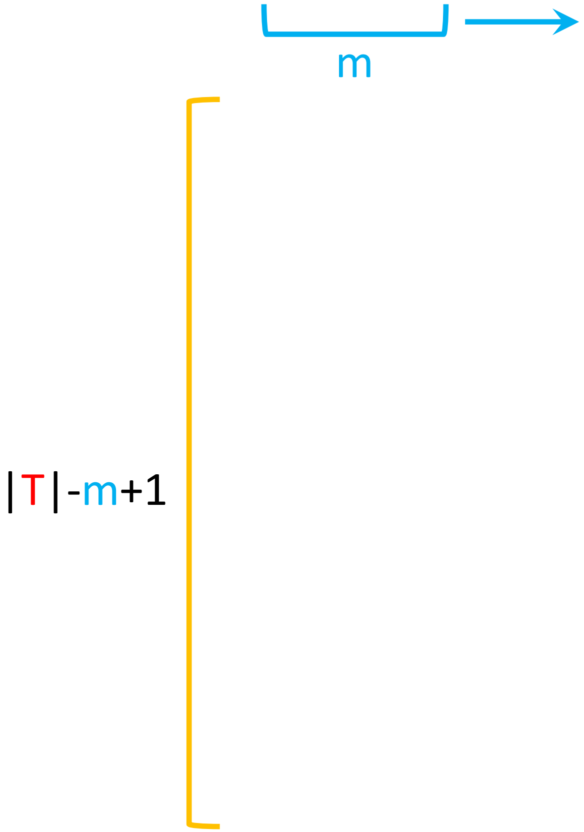
A Single Iteration

- Choose a random *k*-projection.
- Hash each *l*-subsequences *x* in input sequence into bucket labelled by *h*(*x*).
- From each bucket *B* with at least *s* sequences, form weight matrix model, and perform EM refinement.
- Candidate motif is the *best one* found from refinement of all enriched buckets.

Matrix Profile

- The Matrix Profile (MP) is a data structure that annotates a TS and can be exploited for many purposed: e.g. efficient Motif Discovery.
- Given a time series, T and a desired subsequence length, m.

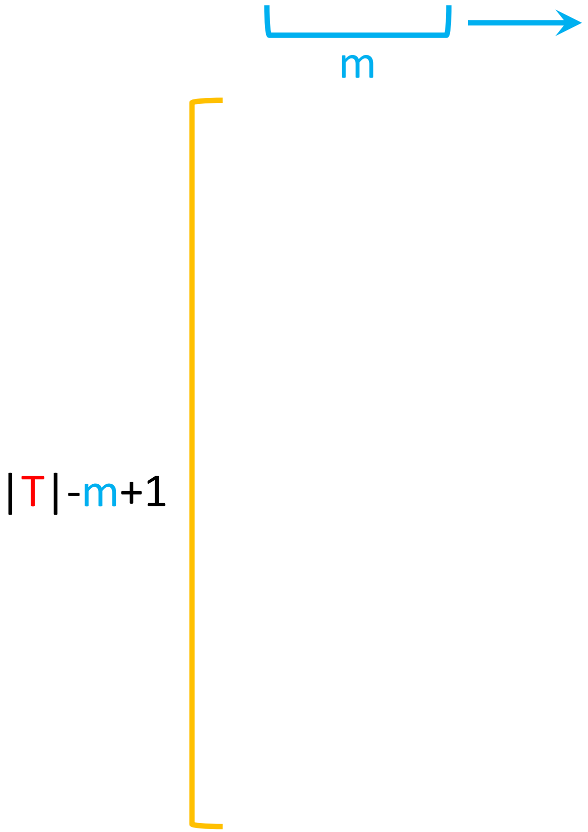
Matrix Profile



We can use sliding window of length *m* to extract all subsequences of length *m*.

…

Matrix Profile



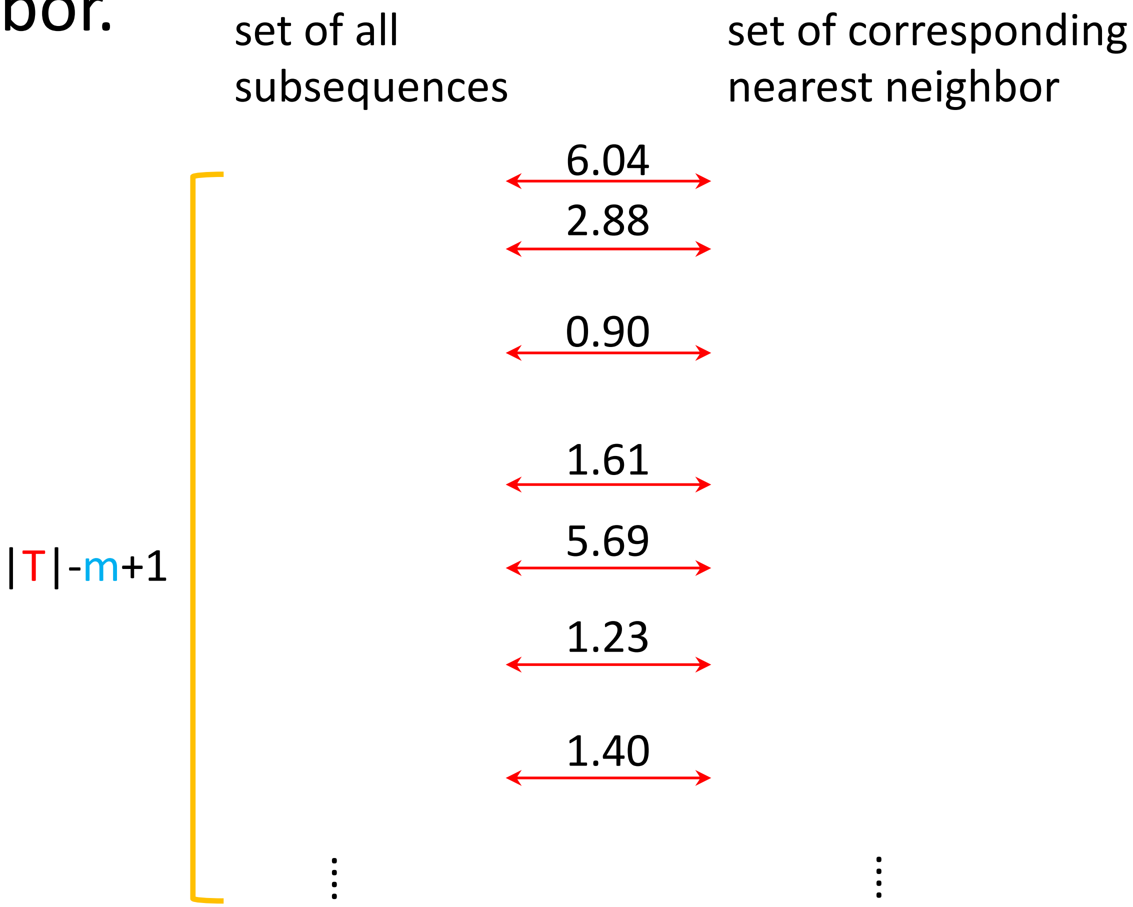
We can then compute the pairwise distance among these subsequences.

…

0	7.6952	7.7399	…
7.6952	0	7.7106	…
7.7399	7.7106	0	…
…	…	…	…

Matrix Profile

- For each subsequence we keep only the distance with the closest nearest neighbor.



set of all subsequences

set of corresponding nearest neighbor

|T|-m+1

6.04

2.88

0.90

1.61

5.69

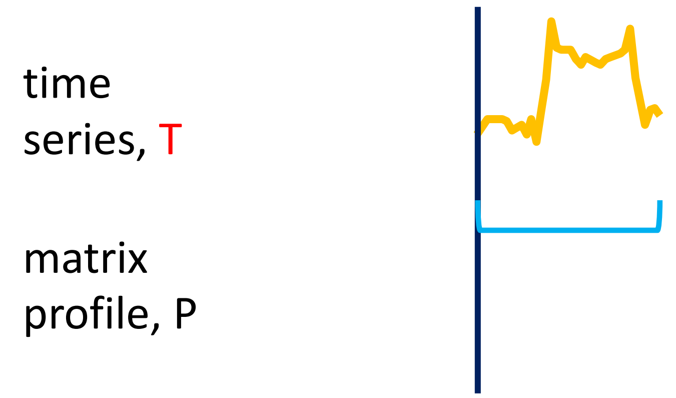
1.23

1.40

⋮ ⋮

Matrix Profile

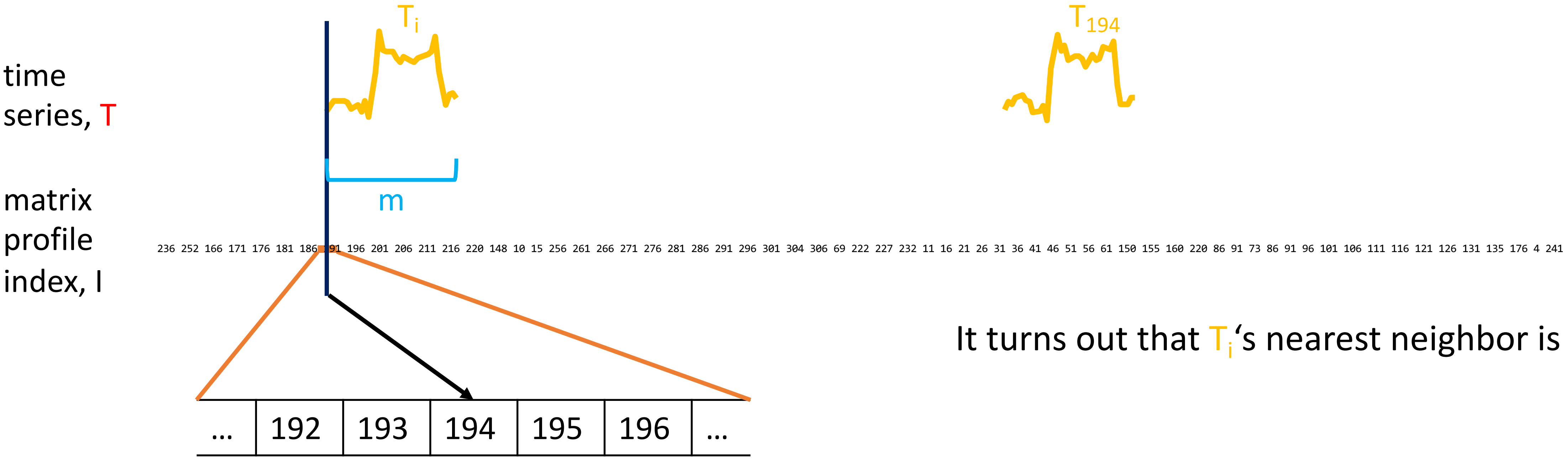
- The distance to the corresponding nearest neighbor of each subsequence can be stored in a vector called **matrix profile P**.



The matrix profile value at location *i* is the distance between T_i and its nearest neighbor

Matrix Profile

- The index of corresponding nearest neighbor of each subsequence is also stored in a vector called matrix profile index.



time series, T

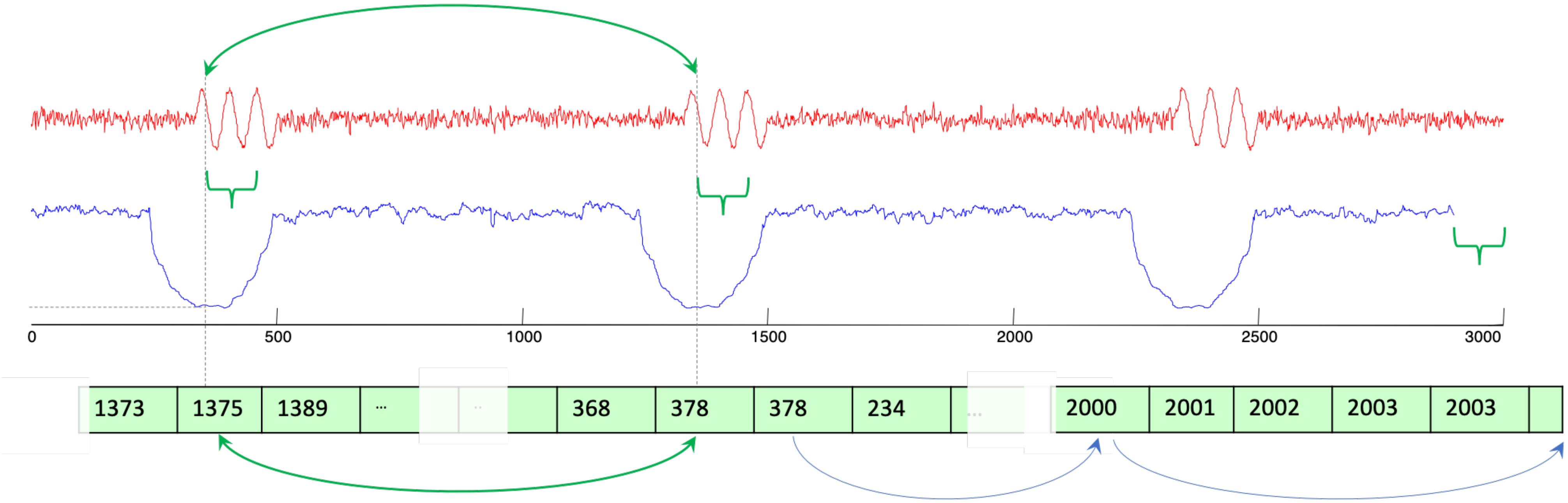
matrix profile index, I

It turns out that T_i's nearest neighbor is T_{194}

The matrix profile value at location *i* is the distance between T_i and its nearest neighbor

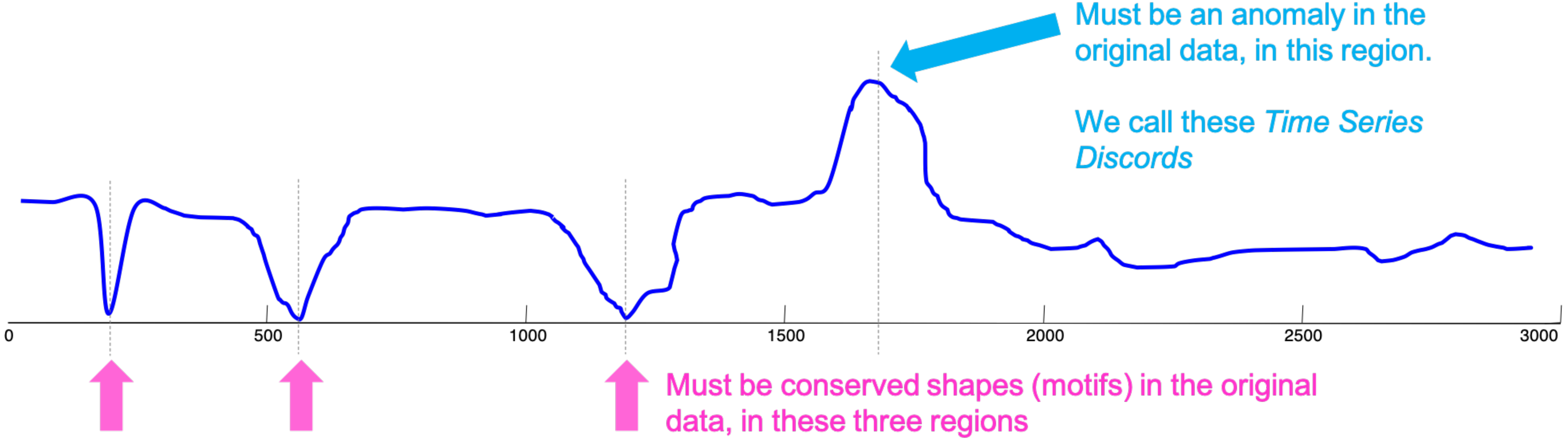
Matrix Profile

- The MP index allows to find the nearest neighbor to any subsequence in constant time.
- Note that the pointers in the matrix profile index are not necessarily symmetric.
- If A points to B, then B may or may not point to A
- The classic TS motif: the two smallest values in the MP must have the same value, and their pointers must be mutual.



How to “read” a Matrix Profile

- For relatively low values, you know that the subsequence in the original TS must have (at least one) relatively similar subsequence elsewhere in the data (such regions are “motifs”)
- For relatively high values, you know that the subsequence in the original TS must be unique in its shape (such areas are anomalies).



Must be an anomaly in the original data, in this region.

We call these *Time Series Discords*

Must be conserved shapes (motifs) in the original data, in these three regions

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

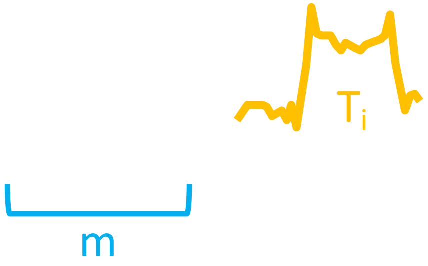
inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

Matrix profile is initialized as inf vector

This is just a toy example, so the values and the vector length does not fit the time series shown above

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

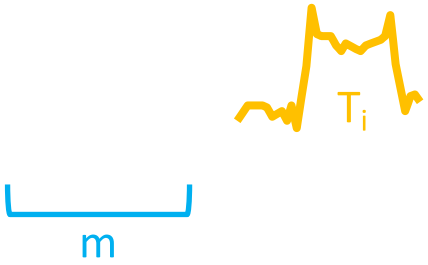


inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

At the first iteration, a subsequence T_i is randomly selected from T

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.



inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

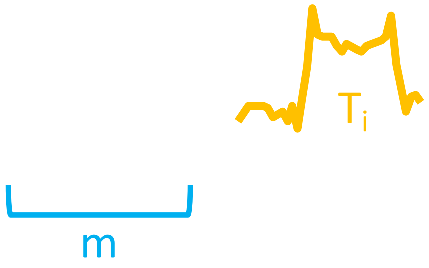
We compute the distances between T_i and every subsequences from T (time complexity = O(|T|log(|T|)))
We then put the distances in a vector based on the position of the subsequences

3	2	0	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

The distance between T_i and T_1 (first subsequence) is 3

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.



inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

We compute the distances between T_i and every subsequences from T (time complexity = $O(|T|\log(|T|))$)
We them put the distances in a vector based on the position of the subsequences

3	2	0	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

Let say T_i happen to be the third subsequences, therefore
the third value in the distance vector is 0

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

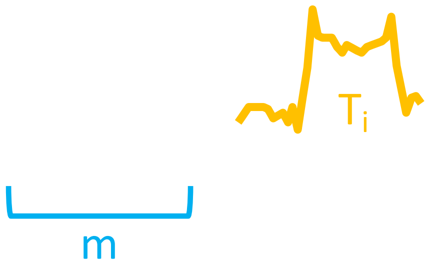
min ↕

Matrix profile is updated by apply elementwise minimum to these two vectors

3	2	0	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.



3	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

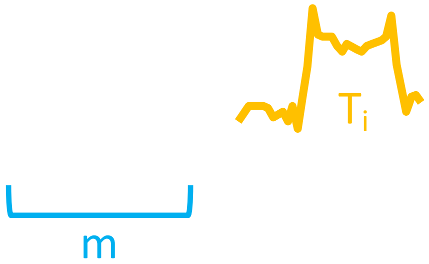
min

Matrix profile is updated by apply elementwise minimum to these two vectors

3	2	0	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.



3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

After we finish to update matrix profile for the first iteration

3	2	0	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

In the second iteration, we randomly select another subsequence T_j and it happens to be the 12th subsequences

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

Once again, we compute the distance between T_j and every subsequences of T

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

min ↕

The same elementwise minimum

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

m

2	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

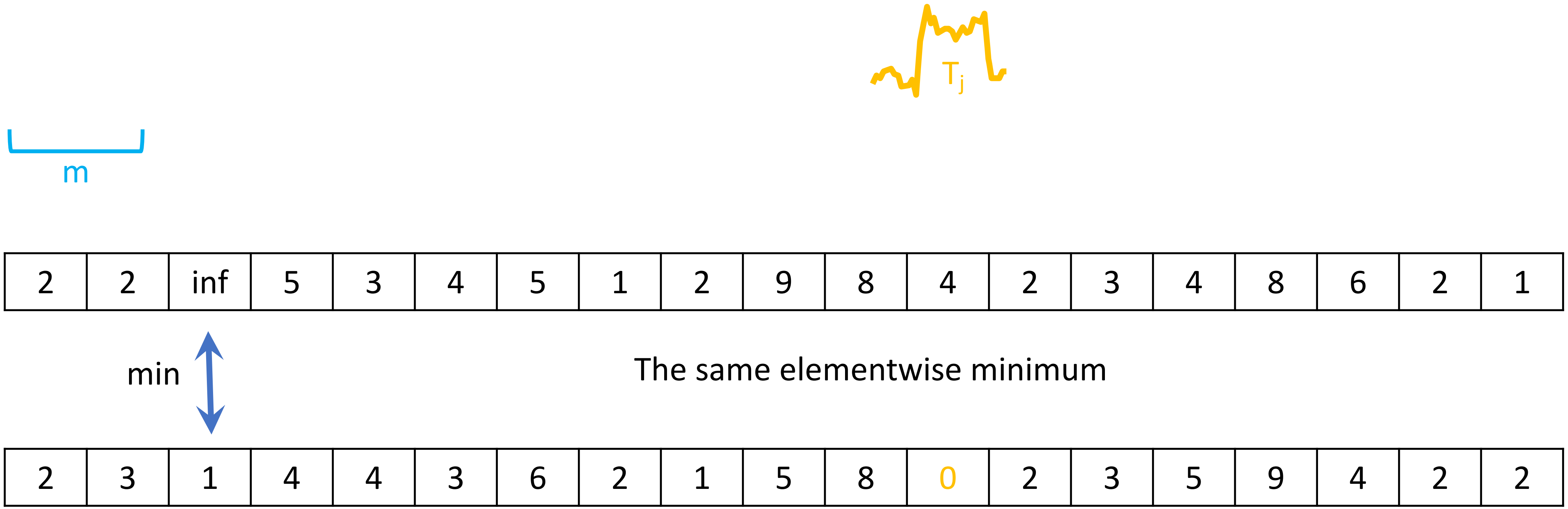
min

The same elementwise minimum

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.



2	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

min

The same elementwise minimum

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

2	2	1	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

min

The same elementwise minimum

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

m

2	2	1	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

min

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

We repeat the two steps (distance computation and update) until we have used every subsequences

How to Compute Matrix Profile?

- Given a time series, T and a desired subsequence length, m.

m

2	2	1	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1

min

2	3	1	4	4	3	6	2	1	5	8	0	2	3	5	9	4	2	2

There are |T| subsequences and the distance computation is O(|T|log(|T|))

The overall time complexity is $O(|T|^2\log(|T|))$

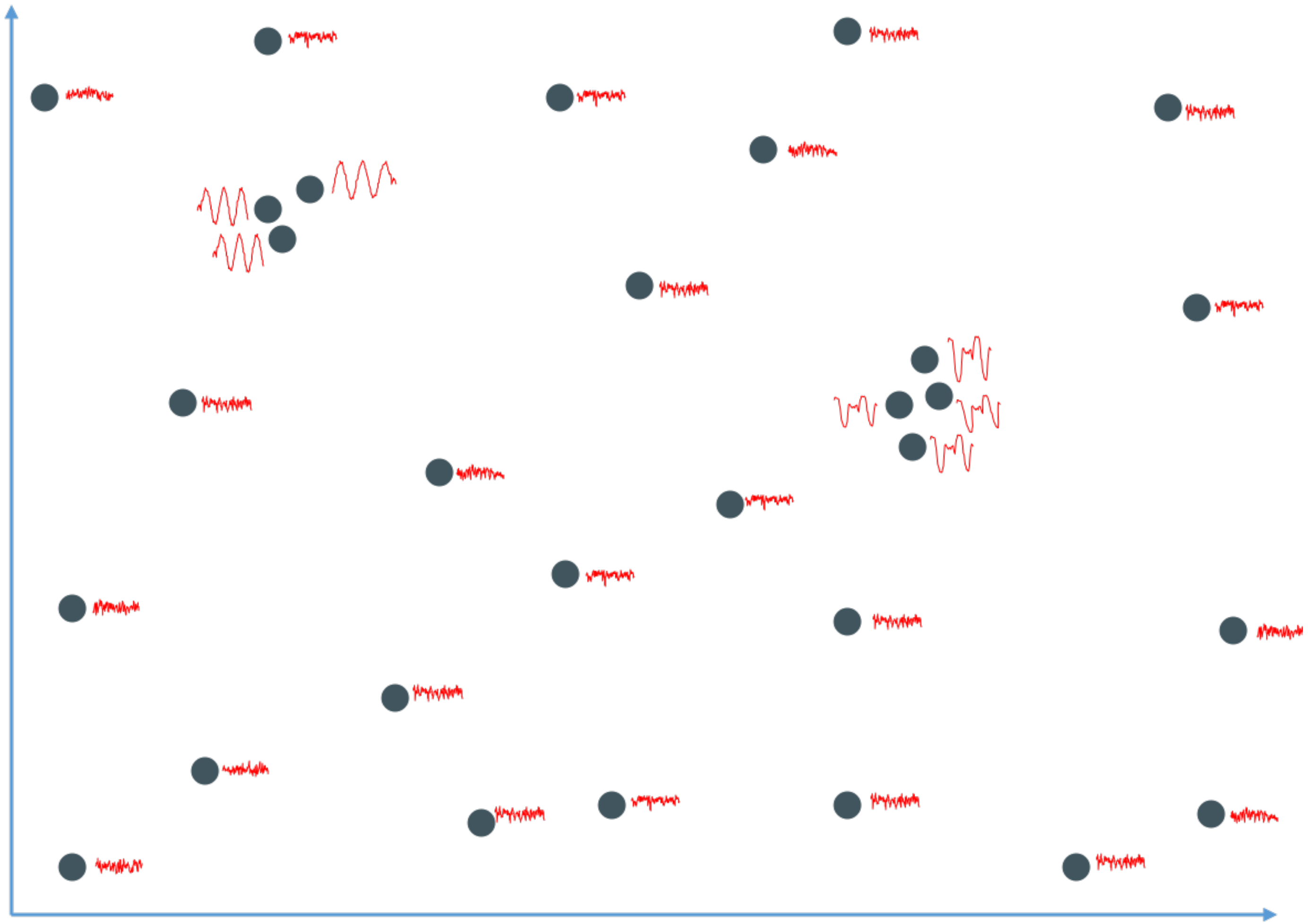
Motif Discovery From Matrix Profile

time
series, T

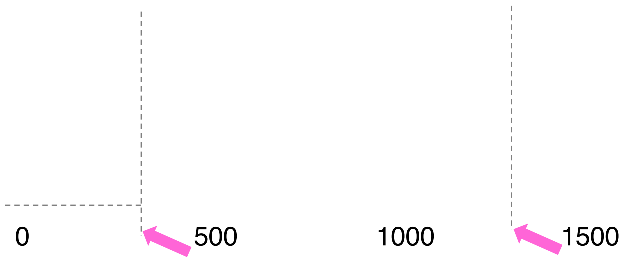
matrix
profile, P

Local minimums are corresponding to motifs

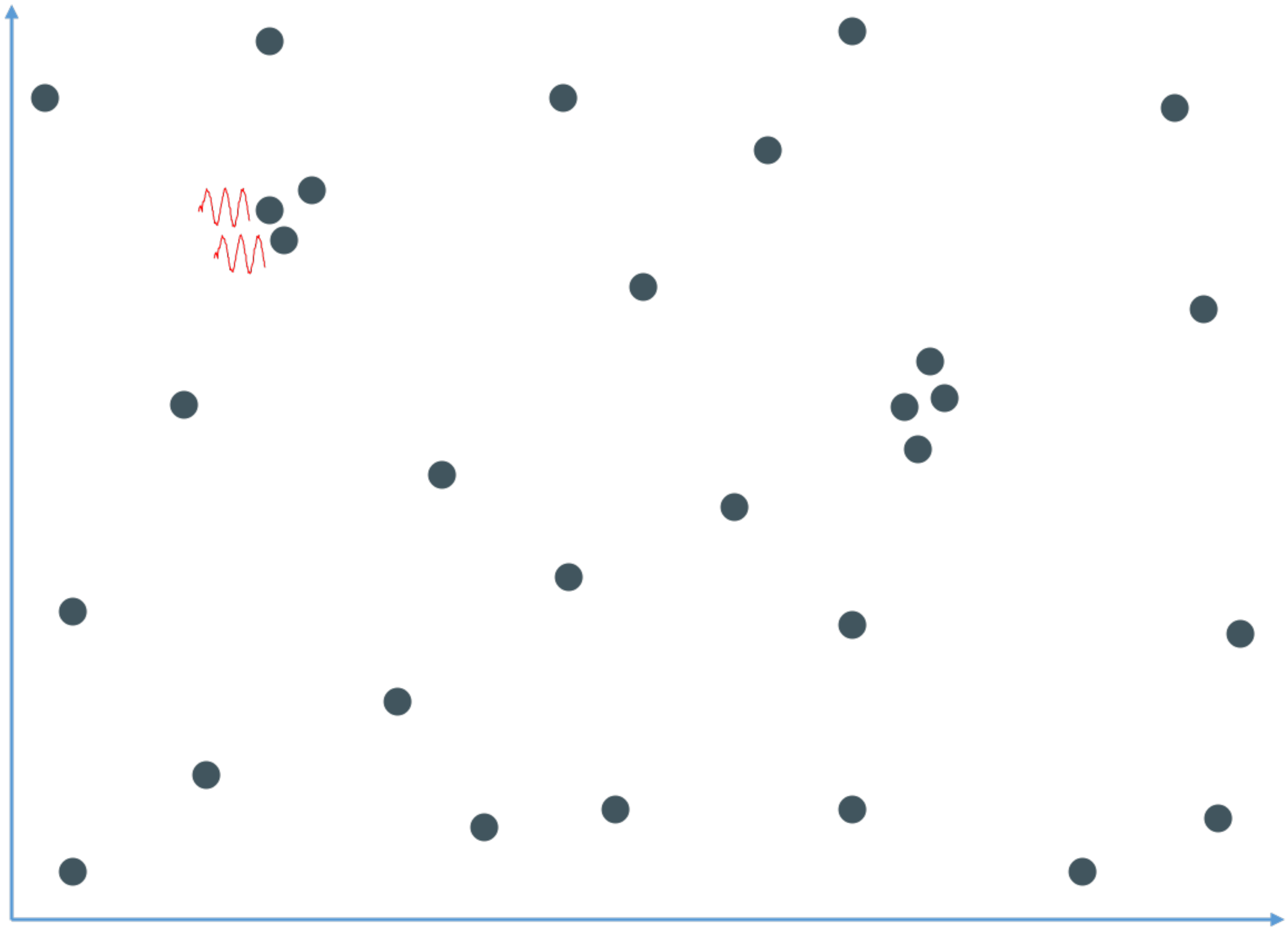
Motif Discovery From Matrix Profile



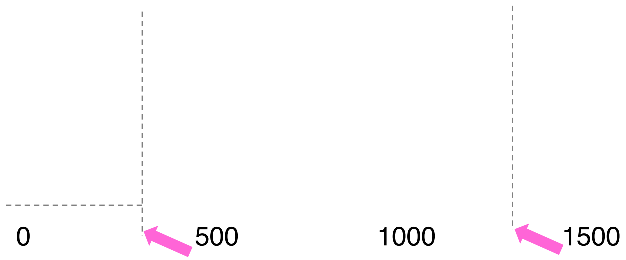
- It is sometime useful to think of time series subsequences as points in m-dimensional space.
- In this view, dense regions in the m-dimensional space correspond to regions of the time series that have a low corresponding MP.



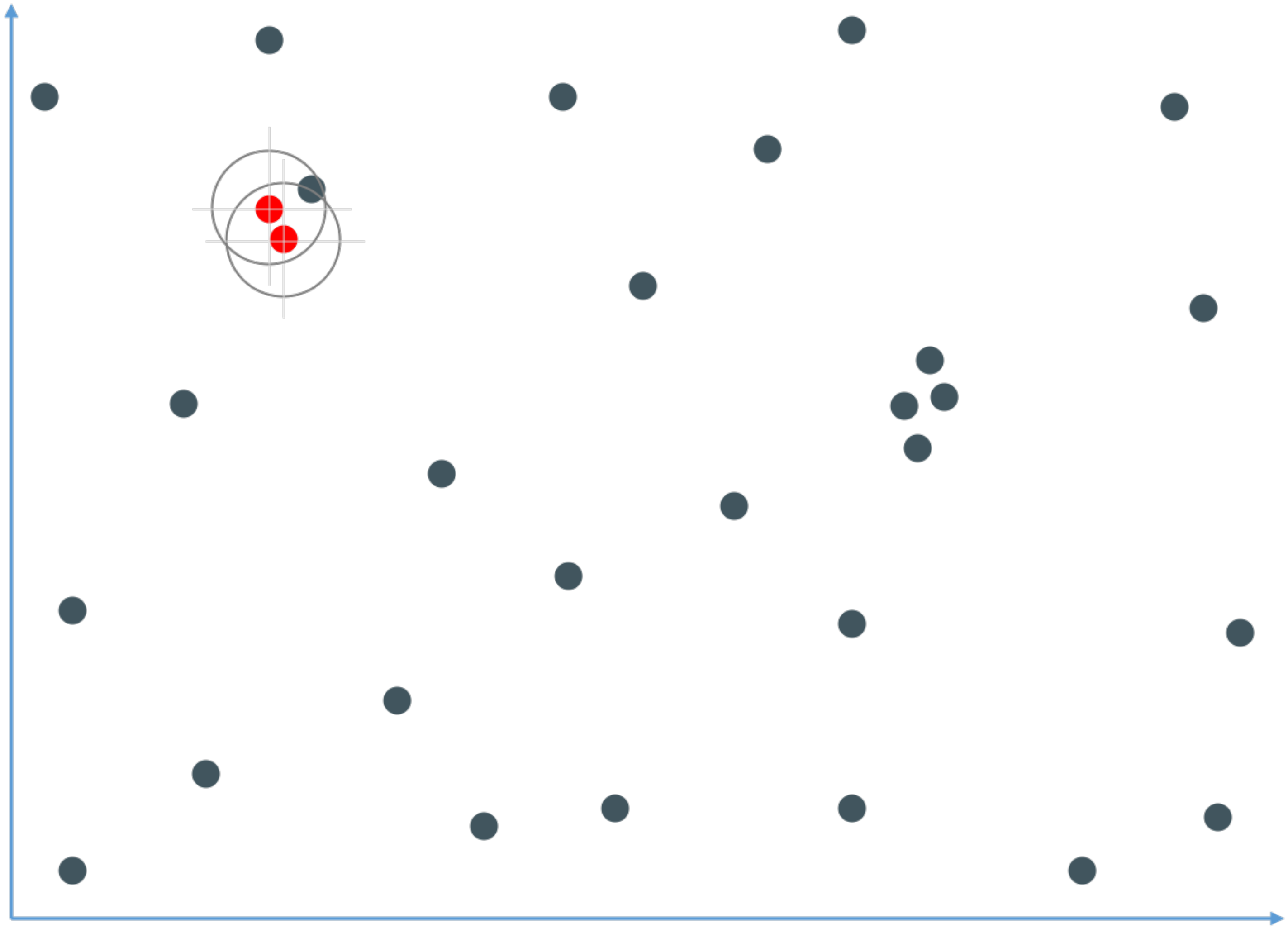
Top-K Motifs



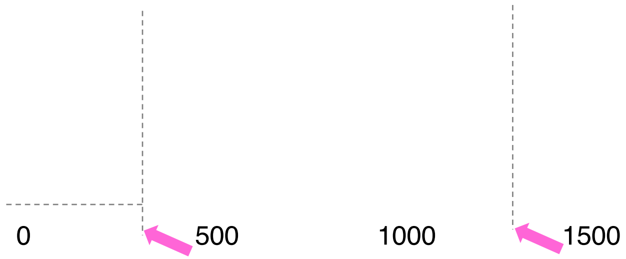
- We need a parameter R.
- 1 < R < (small number, say 3)
- Let’s make R = 2 for now.
- We begin by finding the nearest pair of points, the *motif pair*….
- This pair of subsequences correspond to lowest pair of values in the MP



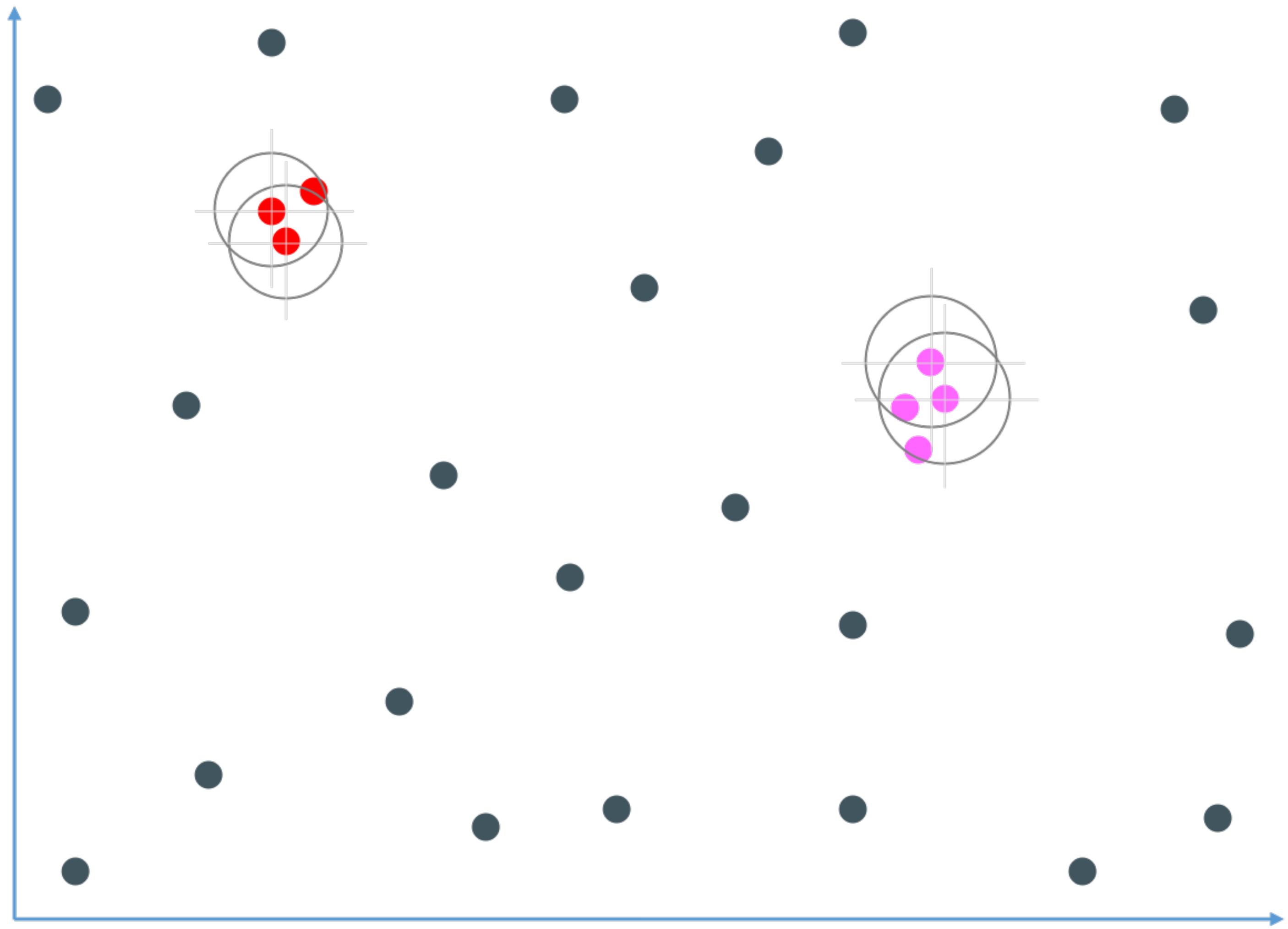
Top-K Motifs



- We find the nearest pair of points are D1 apart.
- Let’s draw a circle, D1 times R, around both points.
- Any points that are within either of these circles, are added to this motif, in this case just one.
- The Top-1 motif has three members, it is done.



Top-K Motifs

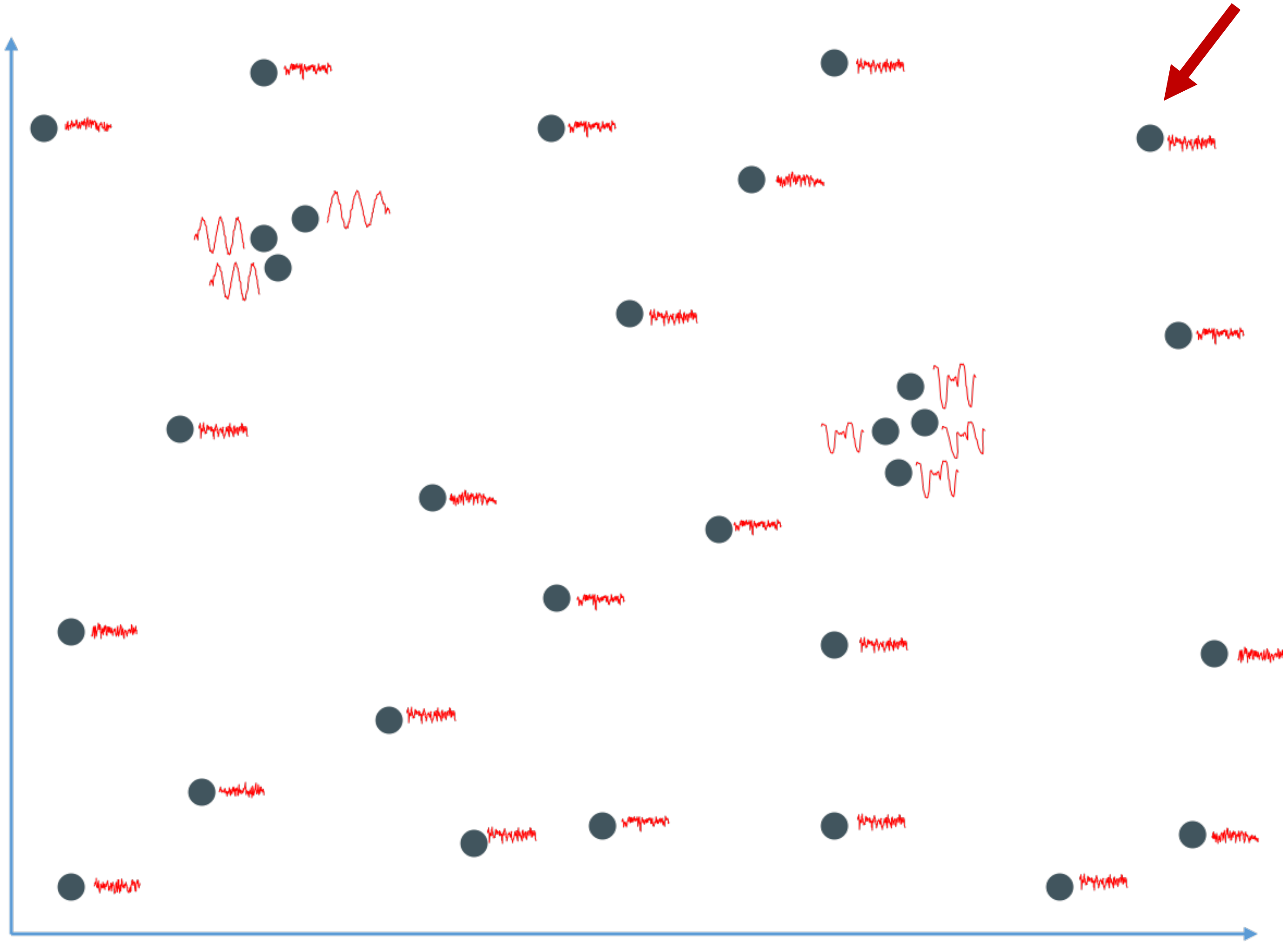


- Now let's find the Top-2 motif. We find the ***nearest pair of points***, excluding anything from the top motif.
- The nearest pair of points are D2 apart.
- Let's draw a circle D2 times R, around both points.
- Any points that are within either of these circles, is added to this motif, in this case there are two for a total of four items in the Top-2 Motif

Top-K Motifs

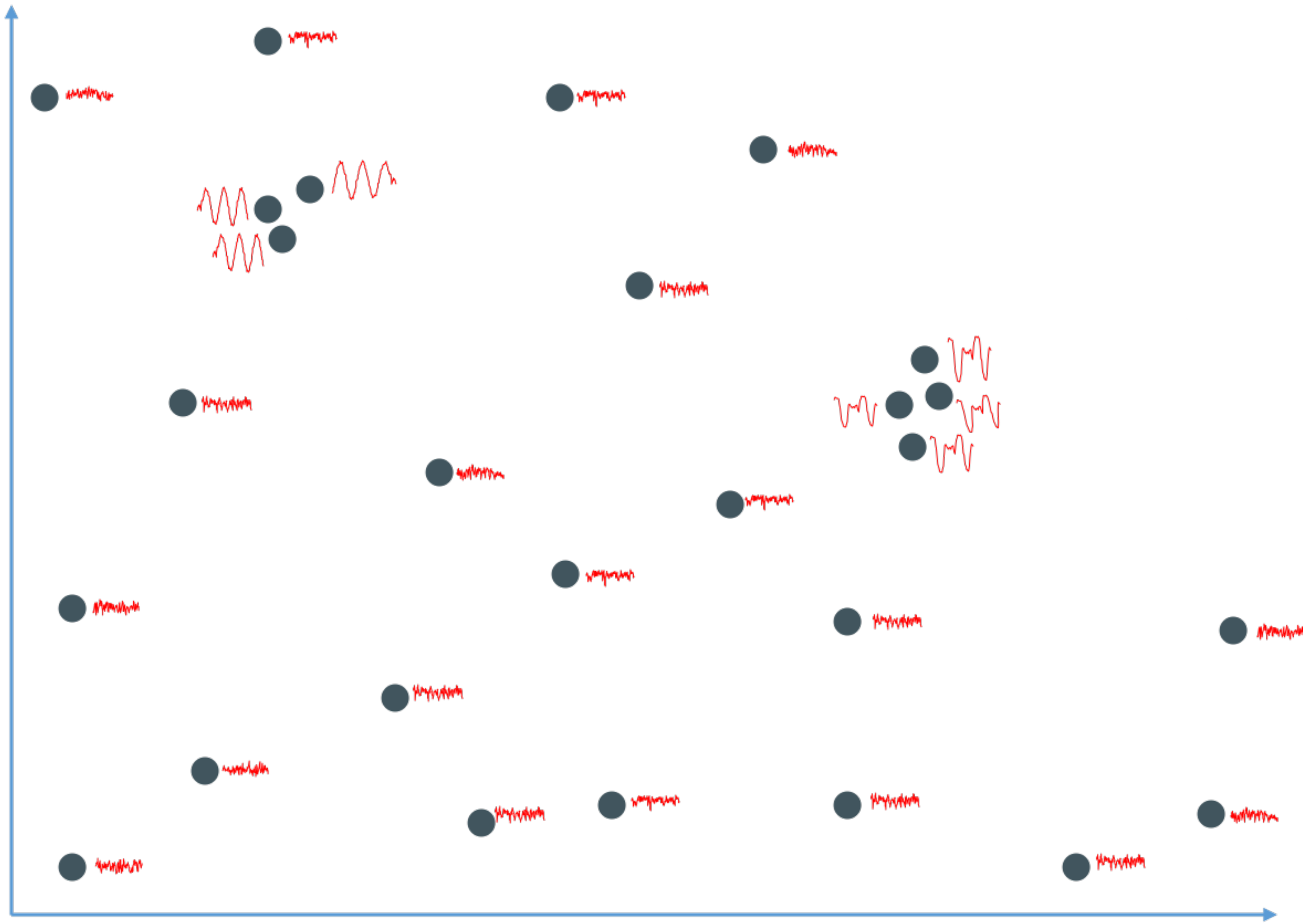
- We have done with the Top-2 Motif
- Note that we will always have:
    - $D_1 < D_2 < D_3 \ldots D_K$
- **When to stop?** (what is K?)
- We could use MDL or a predefined K.

Anomaly Discovery From Matrix Profile



- We need a parameter E of subsequences to exclude in the vicinity of the anomaly.
- Let’s make E = 2 for now.
- We begin by finding the subsequence with the highest distance in the MP
- This corresponds to the biggest anomaly

Top-K Anomaly



- Then we look for the E closest subsequences to the anomaly.
- We remove all of them.
- We can use a predefined K or the MDL to stop.

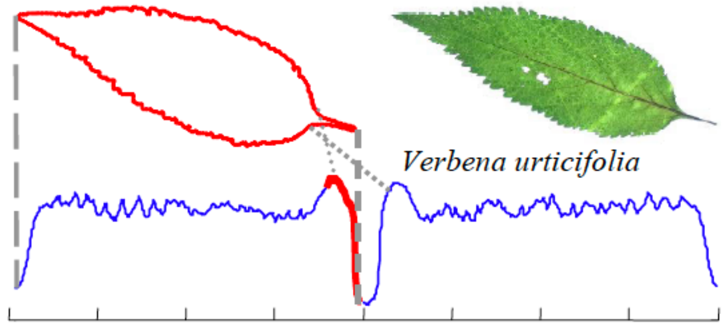
Shapelet

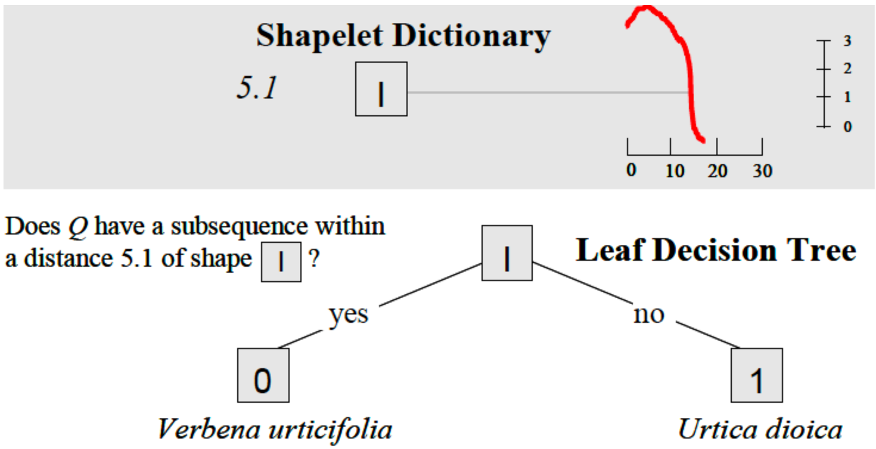
Time Series Classification

- Given a set X of n time series, $X = \{x_1, x_2, \ldots, x_n\}$, each time series has m ordered values $x_i = < x_{t1}, x_{t2}, \ldots, x_{tm} >$ and a class value c_i.
- The objective is to find a function f that maps from the space of possible time series to the space of possible class values.
- Generally, it is assumed that all the TS have the same length m.

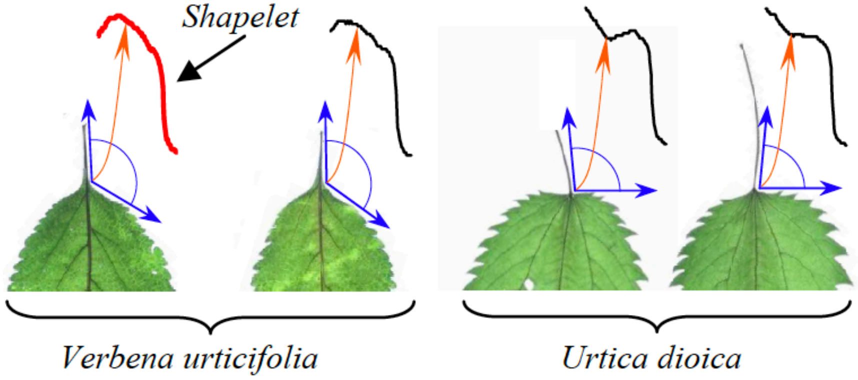
Shapelet-based Classification

1. Represent a TS as a vector of distances with representative subsequences, namely shapelets.
2. Use it as input for machine learning classifiers.





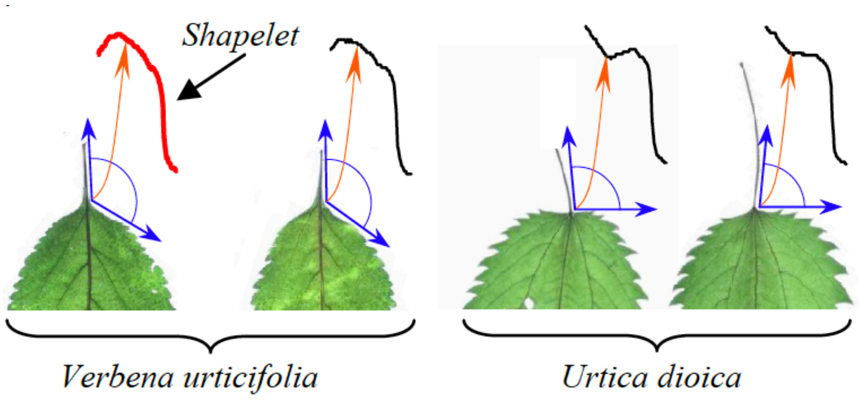
3.2	8.7
1.4	7.9
6.7	4.2
9.2	3.4



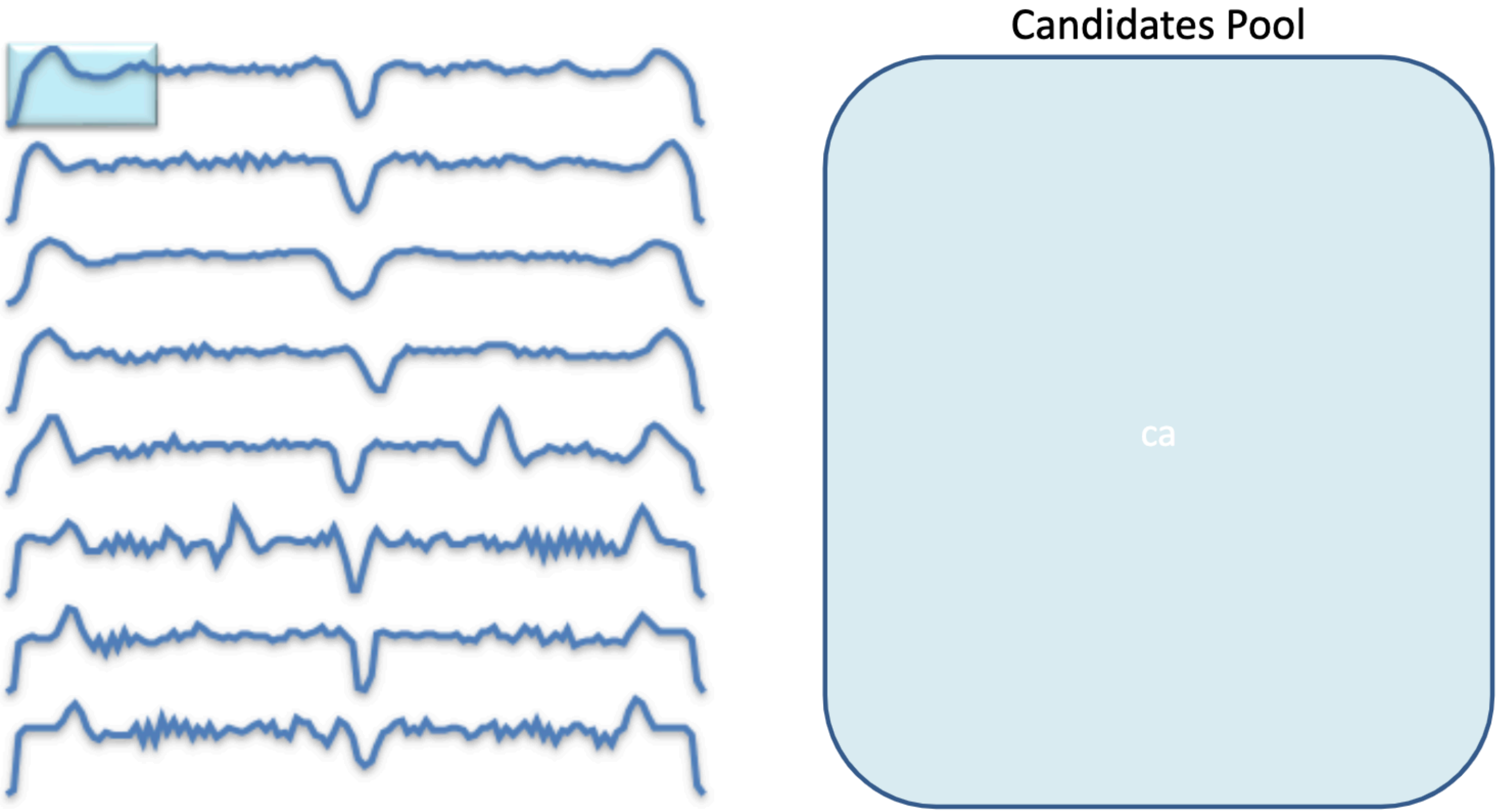
Time Series Shapelets

- Shapelets are TS subsequences which are maximally representative of a class.
- Shapelets can provide interpretable results, which may help domain practitioners better understand their data.
- Shapelets can be significantly more accurate/robust because they are *local features*, whereas most other state-of-the-art TS classifiers consider *global features*.

Verbena	0.87
Urtica	0.34



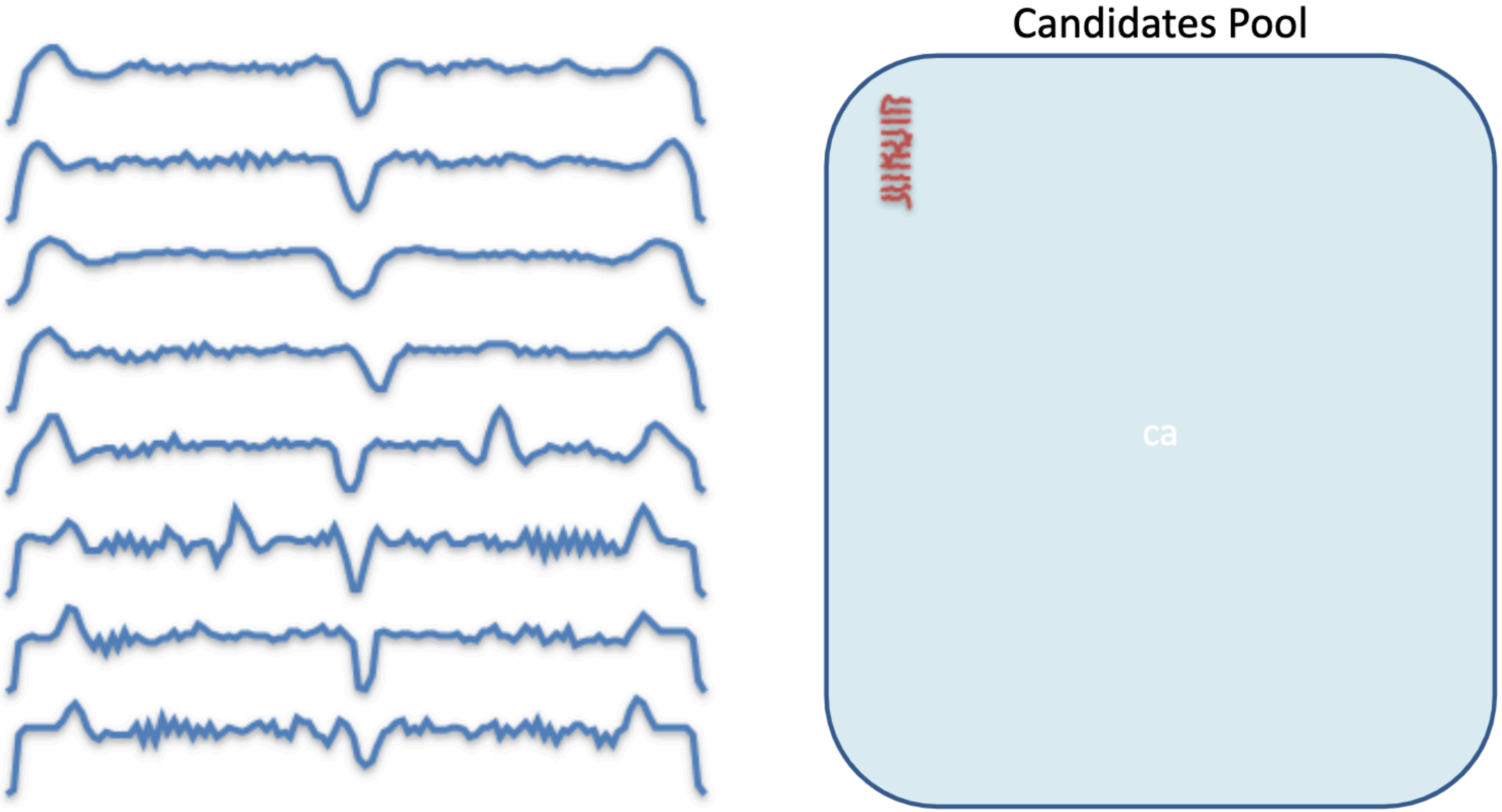
Extract Subsequences of all Possible Lengths



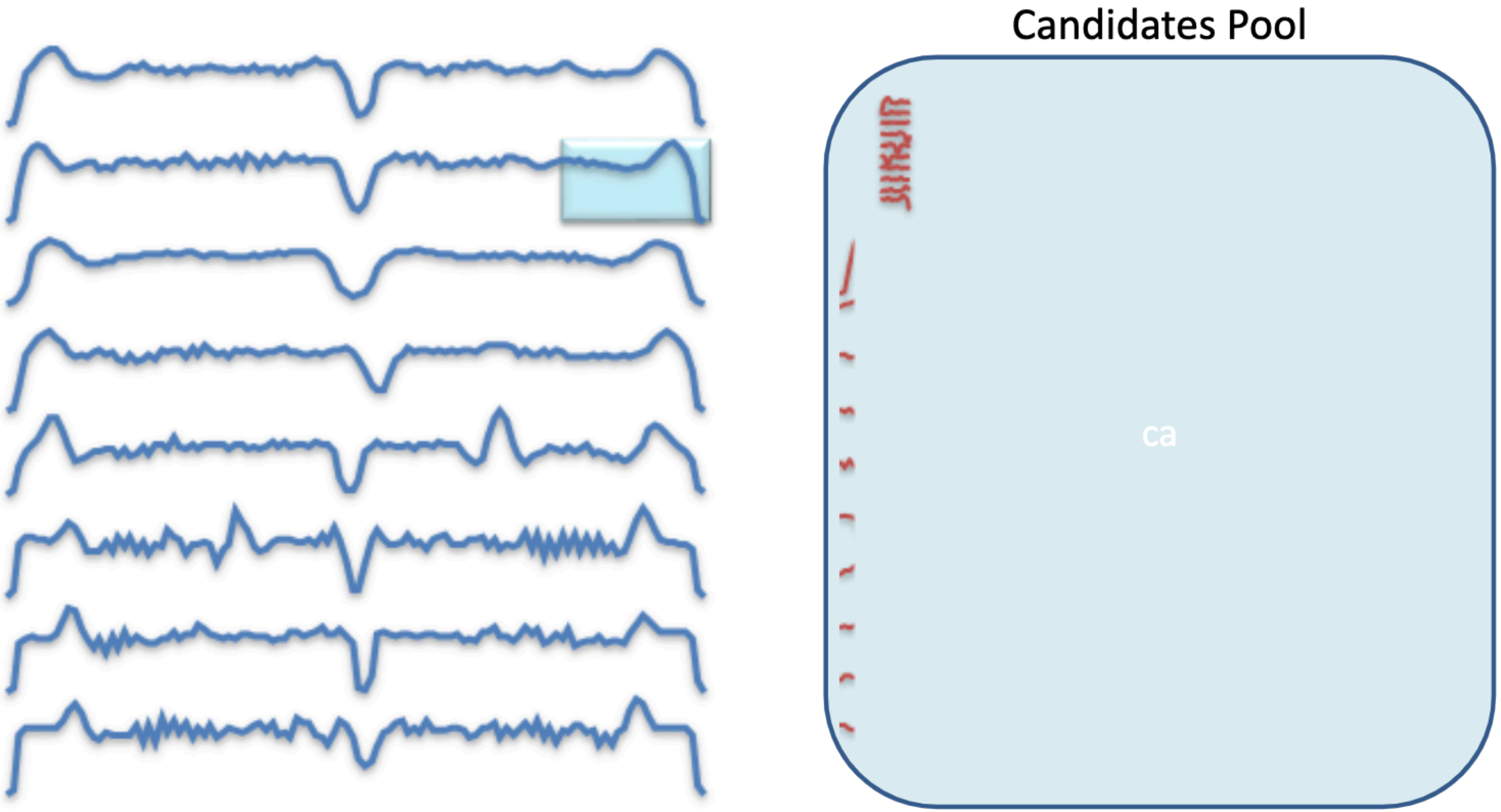
Candidates Pool

ca

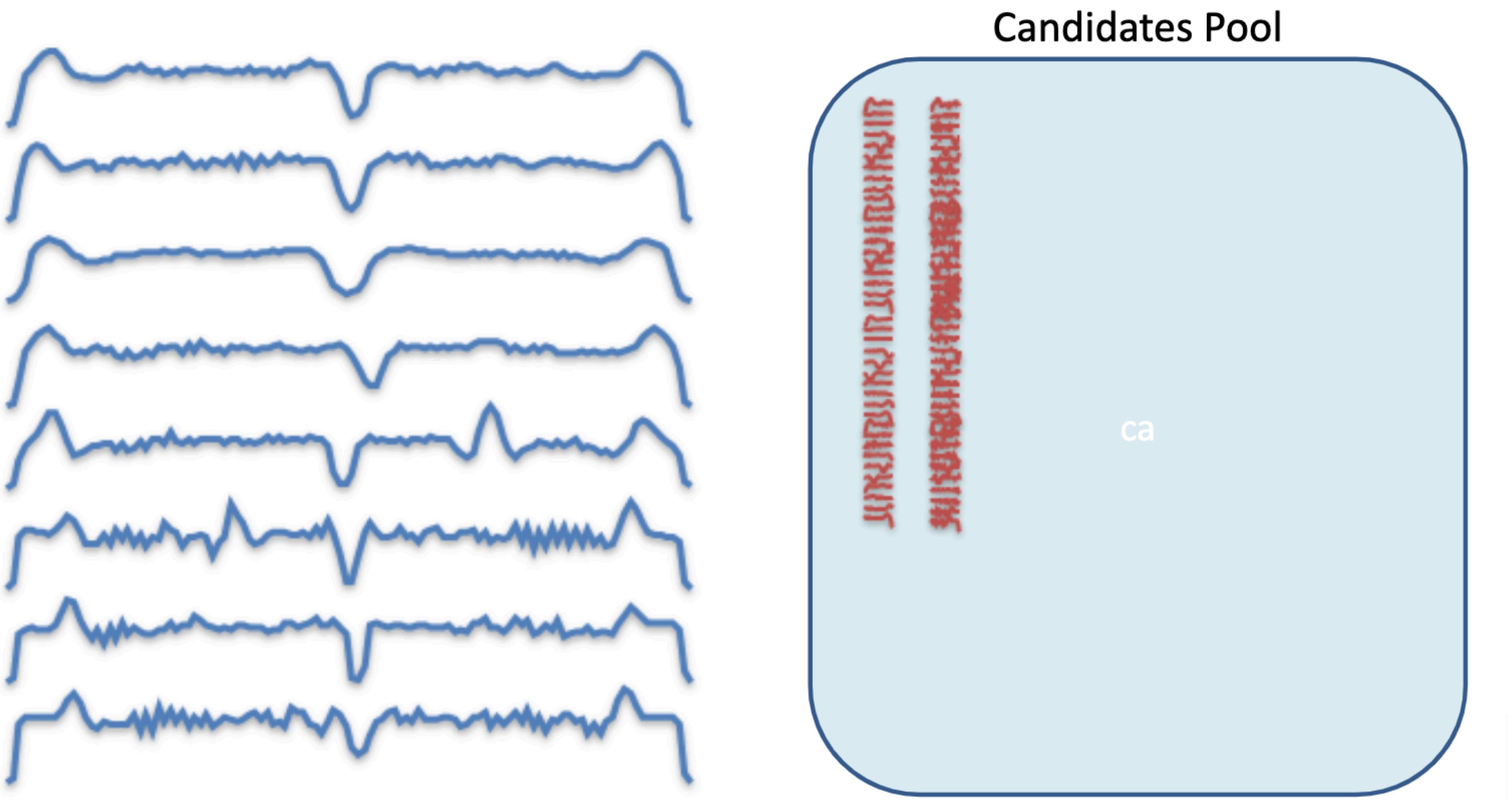
Extract Subsequences of all Possible Lengths



Extract Subsequences of all Possible Lengths

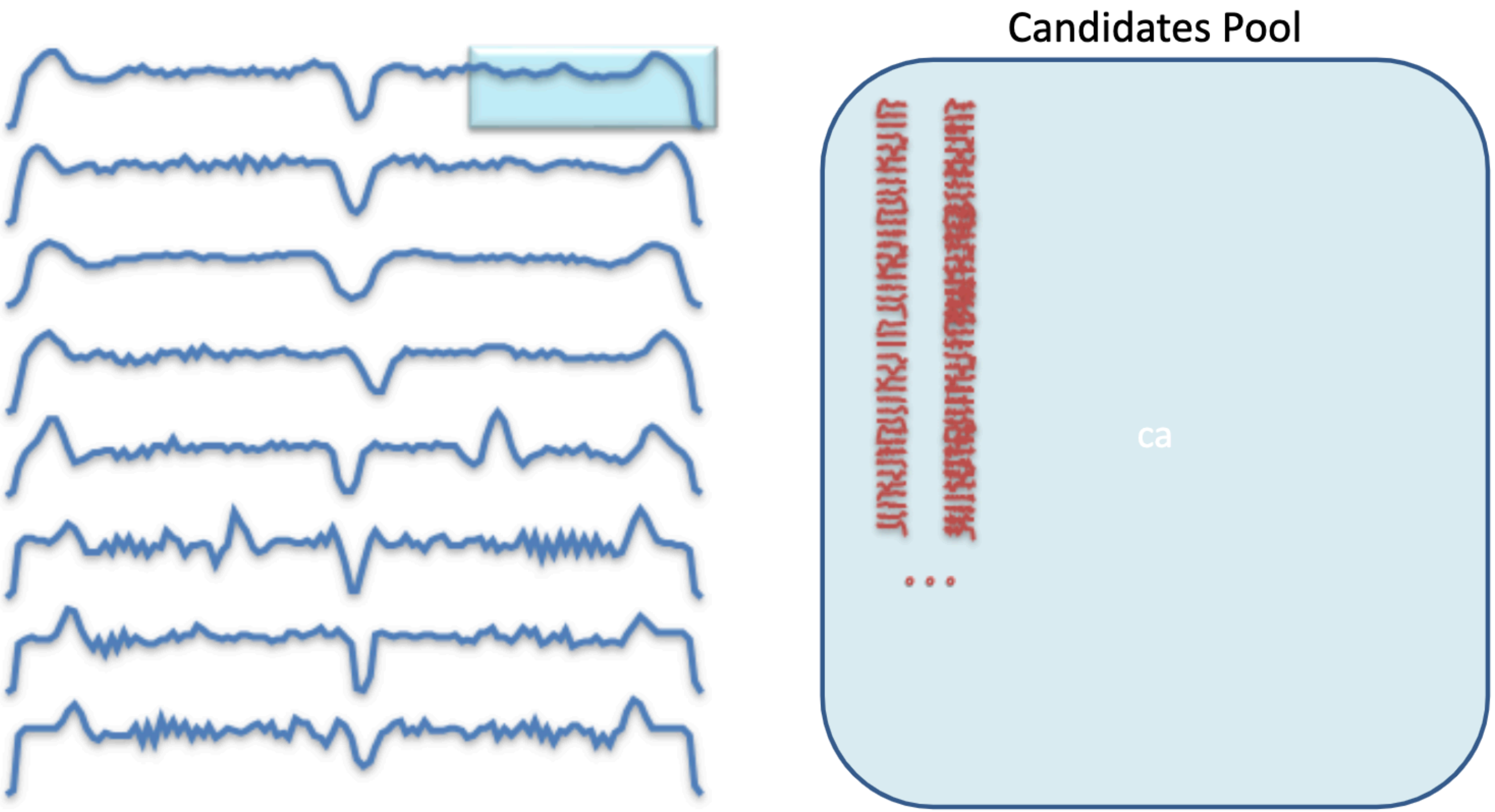


Extract Subsequences of all Possible Lengths



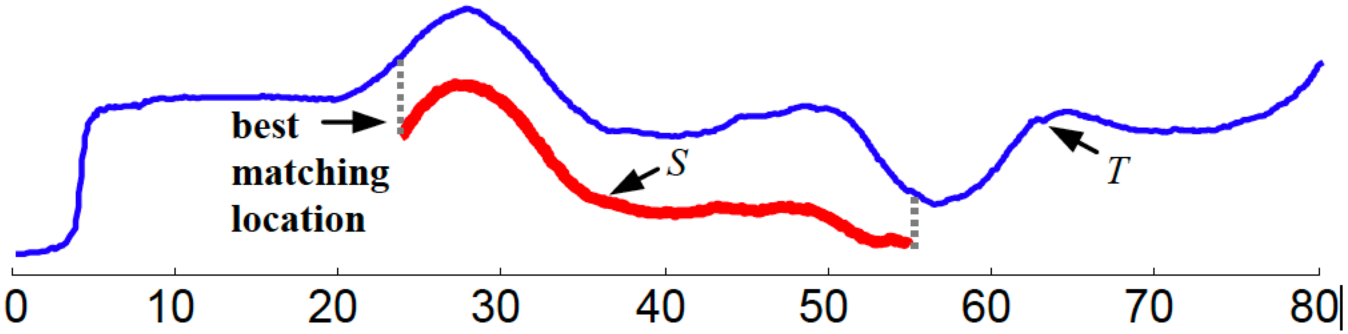
Candidates Pool

Extract Subsequences of all Possible Lengths



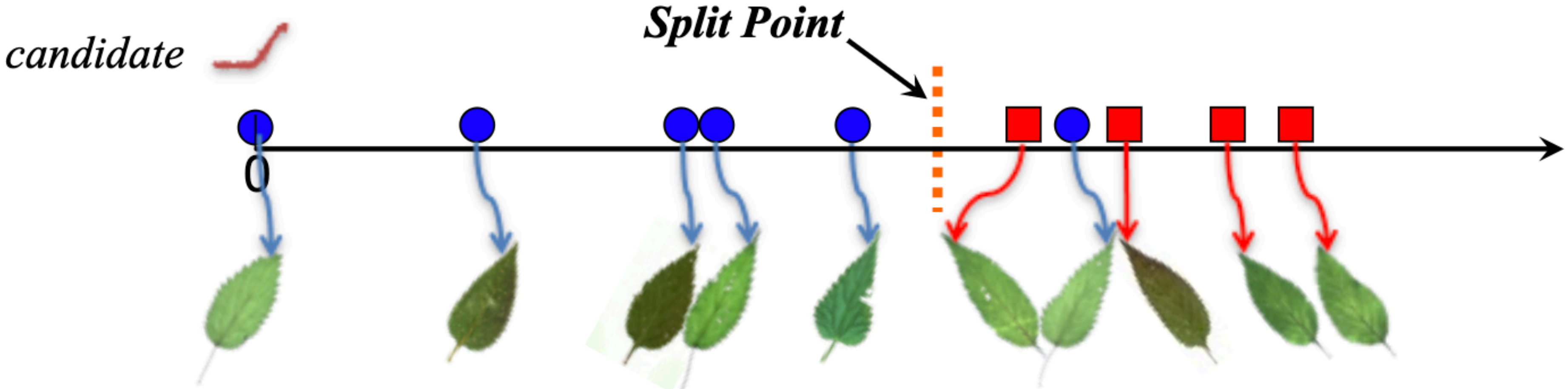
Distance with a Subsequence

- Distance from the TS to the subsequence *SubsequenceDist(T, S)* is a distance function that takes time series *T* and subsequence *S* as inputs and returns a nonnegative value *d*, which is the distance from *T* to *S*.
- $SubsequenceDist(T, S) = min(Dist(S, S')), \text{ for } S' \in S_T^{|S|}$
- where $S_T^{|S|}$ is the set of all possible subsequences of *T*
- Intuitively, it is the distance between *S* and its best matching location in *T*.

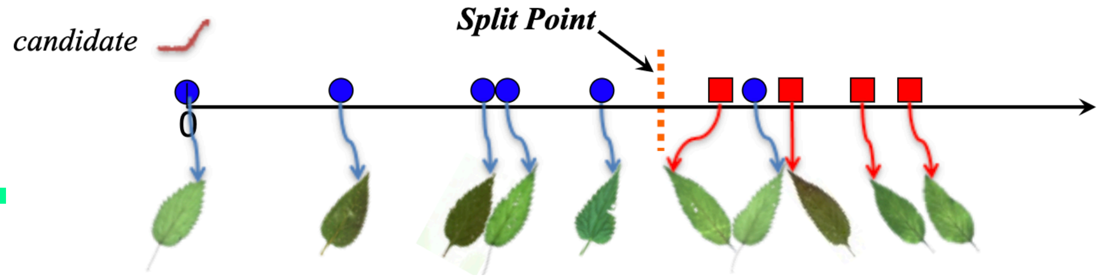


Testing The Utility of a Candidate Shapelet

- Arrange the TSs in the dataset *D* based on the distance from the candidate.
- Find the optimal split point that maximizes the information gain (same as for Decision Tree classifiers)
- Pick the candidate achieving best utility as the shapelet

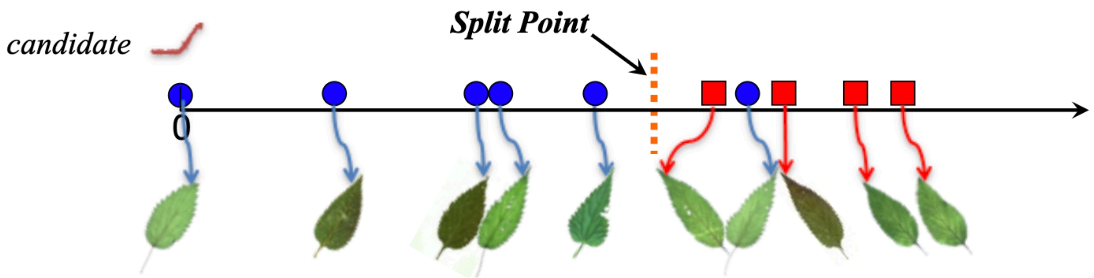


Entropy



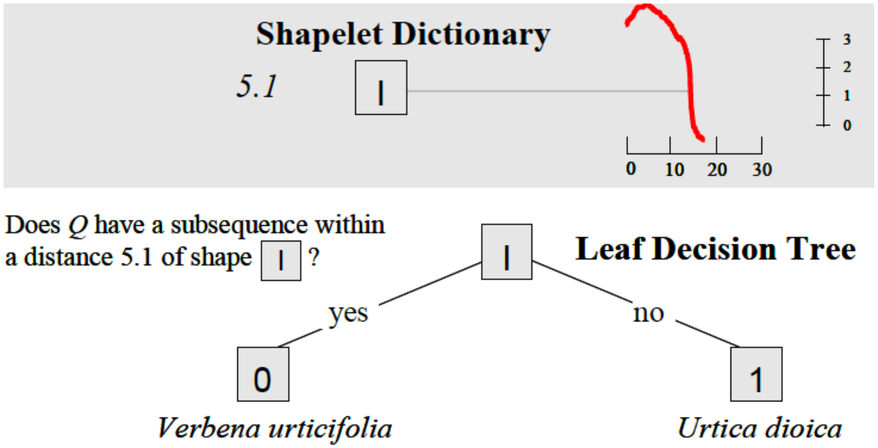
- A TS dataset *D* consists of two classes, A and B.
- Given that the proportion of objects in class A is *p(A)* and the proportion of objects in class B is *p(B)*,
- The **Entropy** of D is: $I(D) = -p(A)\log(p(A)) - p(B)\log(p(B))$.
- Given a strategy that divides the *D* into two subsets D_1 and D_2, the information remaining in the dataset after splitting is defined by the weighted average entropy of each subset.
- If the fraction of objects in D_1 is $f(D_1)$ and in D_2 is $f(D_2)$,
- The total entropy of *D* after splitting is $\hat{I}(D) = f(D_1)I(D_1) + f(D_2)I(D_2)$.

Information Gain



Split point distance from shapelet = 5.1

- Given a certain split strategy *sp* which divides *D* into two subsets D_1 and D_2, the entropy before and after splitting is $I(D)$ and $\hat{I}(D)$.
- The **information gain** for this splitting rule is:
- $Gain(sp) = I(D) - \hat{I}(D) =$
- $= I(D) - f(D_1)I(D_1) + f(D_2)I(D_2)$.
- We use the distance from *T* to a shapelet *S* as the splitting rule *sp*.



Does *Q* have a subsequence within a distance 5.1 of shape I ?

Problem

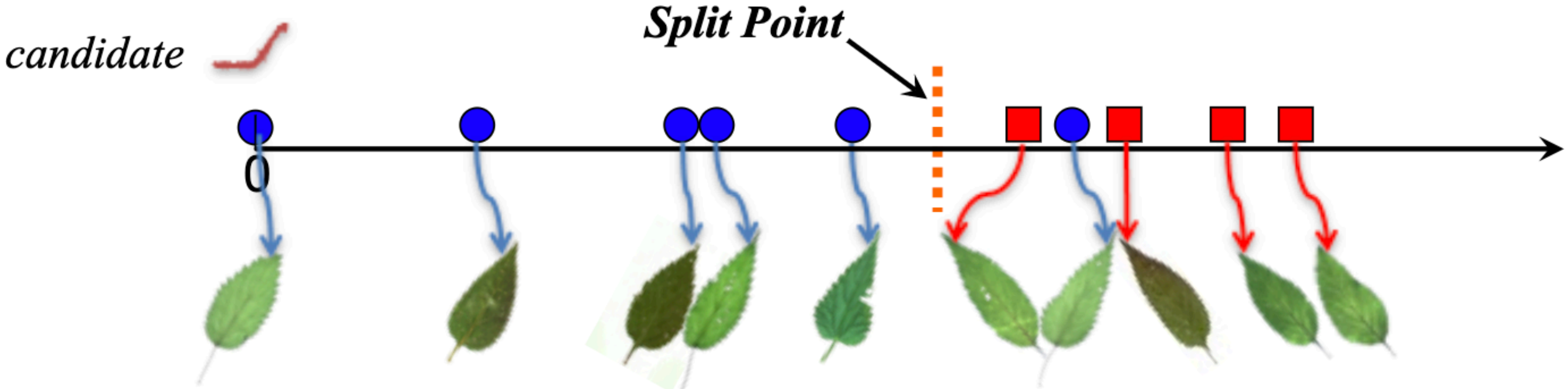
- The total number of candidate is

$$\sum_{l=MINLEN}^{MAXLEN} \sum_{T_i \in D} (|T_i| - l + 1)$$

- For each candidate you have to compute the distance between this candidate and each training sample (space inefficiency)
- For instance
  - 200 instances with length 275
  - 7,480,200 shapelet candidates

Speedup

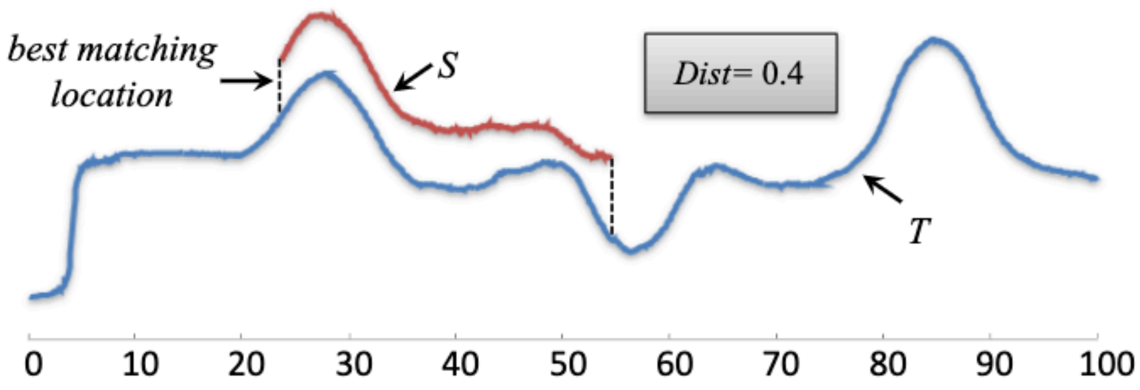
- Distance calculations form TSs to shapelet candidates is expensive.
- Reduce the time in two ways
- Distance Early Abandon
  - reduce the distance computation time between two TS
- Admissible Entropy Pruning
  - reduce the number of distance calculations

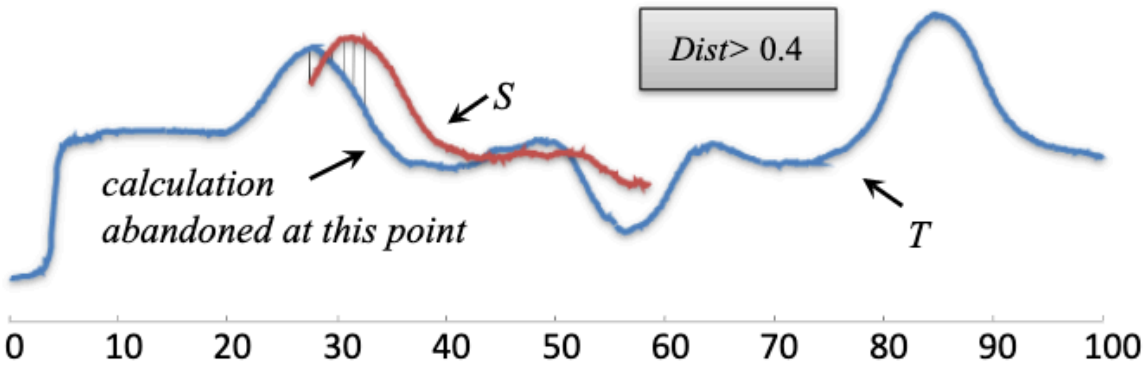


Distance Early Abandon

- We only need the minimum distance.

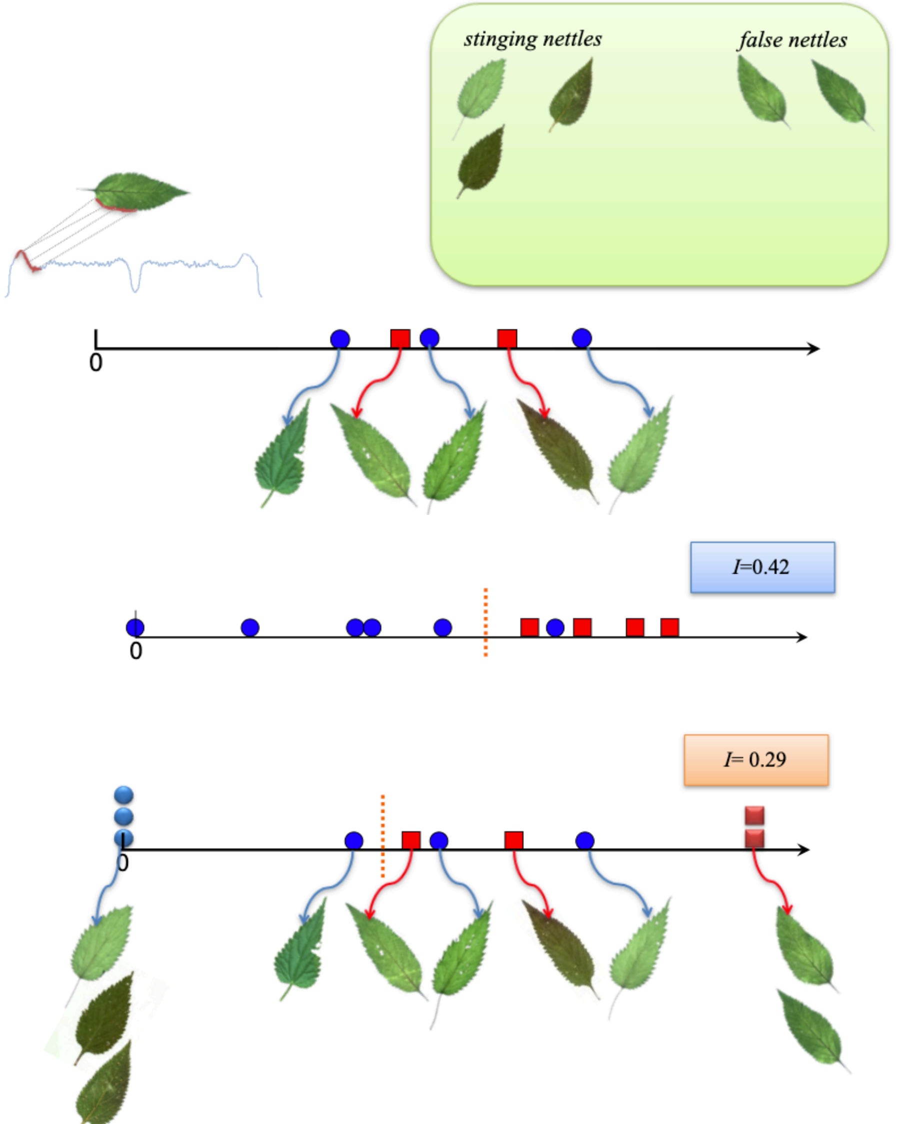
- Method
  - Keep the best-so-far distance
  - Abandon the calculation if the current distance is larger than best-so-far.





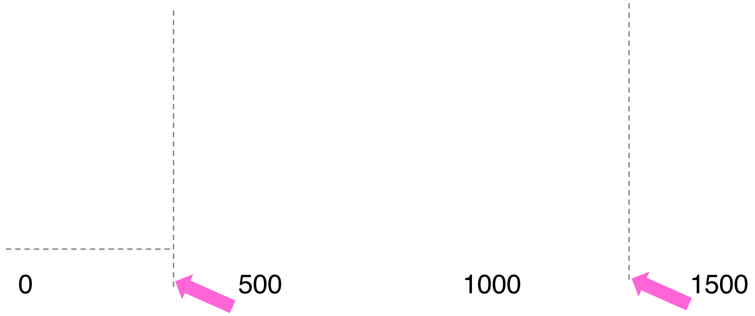
Admissible Entropy Pruning

- We only need the best shapelet for each class
- For a candidate shapelet
    - We do not need to calculate the distance for each training sample
    - After calculating some training samples, the **upper bound** of information gain (corresponding to the optimistic scenario) < best candidate shapelet
    - Stop calculation
    - Try next candidate

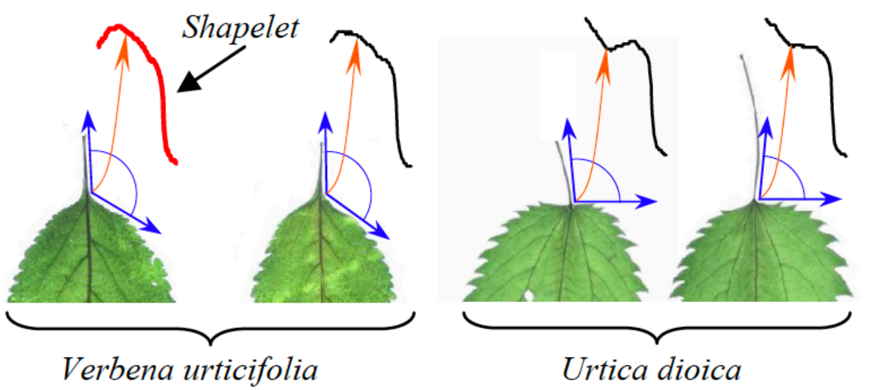


Motif/Shapelet Summary

- A **motif** is a repeated pattern/subsequence in a given TS.

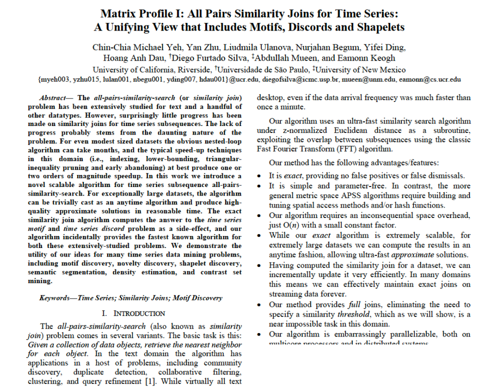


- A **shapelet** is a pattern/subsequence which is maximally representative of a class with respect to a given dataset of TSs.



References

- Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. Chin-Chia Michael Yeh et al. 1997
- Time Series Shapelets: A New Primitive for Data Mining. Lexiang Ye and Eamonn Keogh. 2016.
- Josif Grabocka, Nicolas Schilling, Martin Wistuba, Lars Schmidt-Thieme (2014): Learning Time-Series Shapelets, in Proceedings of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2014



Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets

Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding,
Hoang Anh Dau, [†]Diego Furtado Silva, [‡]Abdullah Mueen, and Eamonn Keogh
University of California, Riverside, [†]Universidade de São Paulo, [‡]University of New Mexico
{myeh003, yzhu015, lulan001, nbegu001, yding007, hdau001}@ucr.edu, diegofsilva@icmc.usp.br, mueen@unm.edu, eamonn@cs.ucr.edu

***Abstract*— The *all-pairs-similarity-search* (or *similarity join*) problem has been extensively studied for text and a handful of other datatypes. However, surprisingly little progress has been made on similarity joins for time series subsequences. The lack of progress probably stems from the daunting nature of the problem. For even modest sized datasets the obvious nested-loop algorithm can take months, and the typical speed-up techniques in this domain (i.e., indexing, lower-bounding, triangular-inequality pruning and early abandoning) at best produce one or two orders of magnitude speedup. In this work we introduce a novel scalable algorithm for time series subsequence all-pairs-similarity-search. For exceptionally large datasets, the algorithm can be trivially cast as an anytime algorithm and produce high-quality approximate solutions in reasonable time. The exact similarity join algorithm computes the answer to the *time series motif* and *time series discord* problem as a side-effect, and our algorithm incidentally provides the fastest known algorithm for both these extensively-studied problems. We demonstrate the utility of our ideas for many time series data mining problems, including motif discovery, novelty discovery, shapelet discovery, semantic segmentation, density estimation, and contrast set mining.**

Keywords—Time Series; Similarity Joins; Motif Discovery

I. INTRODUCTION

The *all-pairs-similarity-search* (also known as *similarity join*) problem comes in several variants. The basic task is this: *Given a collection of data objects, retrieve the nearest neighbor for each object*. In the text domain the algorithm has applications in a host of problems, including community discovery, duplicate detection, collaborative filtering, clustering, and query refinement [1]. While virtually all text processing algorithms have analogues in time series data mining, there has been surprisingly little progress on Time Series subsequences All-Pairs-Similarity-Search (TSAPSS).

We believe that this lack of progress stems not from a lack of interest in this useful primitive, but from the daunting nature of the problem. Consider the following example that reflects the needs of an industrial collaborator. A boiler at a chemical refinery reports pressure once a minute. After a year, we have a time series of length 525,600. A plant manager may wish to do a similarity self-join on this data with week-long subsequences (10,080) to discover operating regimes (summer vs. winter or light distillate vs. heavy distillate etc.) The obvious nested loop algorithm requires 132,880,692,960 Euclidean distance computations. If we assume each one takes 0.0001 seconds, then the join will take 153.8 days. The core contribution of this work is to show that we can reduce this time to 6.3 hours, using an off-the-shelf desktop computer. Moreover, we show that this join can be computed and/or updated incrementally. Thus we could maintain this join essentially forever on a standard desktop, even if the data arrival frequency was much faster than once a minute.

Our algorithm uses an ultra-fast similarity search algorithm under z-normalized Euclidean distance as a subroutine, exploiting the overlap between subsequences using the classic Fast Fourier Transform (FFT) algorithm.

Our method has the following advantages/features:

- It is *exact*, providing no false positives or false dismissals.
- It is simple and parameter-free. In contrast, the more general metric space APSS algorithms require building and tuning spatial access methods and/or hash functions.
- Our algorithm requires an inconsequential space overhead, just O(*n*) with a small constant factor.
- While our *exact* algorithm is extremely scalable, for extremely large datasets we can compute the results in an anytime fashion, allowing ultra-fast *approximate* solutions.
- Having computed the similarity join for a dataset, we can incrementally update it very efficiently. In many domains this means we can effectively maintain exact joins on streaming data forever.
- Our method provides *full* joins, eliminating the need to specify a similarity *threshold*, which as we will show, is a near impossible task in this domain.
- Our algorithm is embarrassingly parallelizable, both on multicore processors and in distributed systems

Time Series Shapelets: A New Primitive for Data Mining

Lexiang Ye
Dept. of Computer Science & Engineering
University of California, Riverside, CA 92521
lexiangy@cs.ucr.edu

Eamonn Keogh
Dept. of Computer Science & Engineering
University of California, Riverside, CA 92521
eamonn@cs.ucr.edu

ABSTRACT

Classification of time series has been attracting great interest over the past decade. Recent empirical evidence has strongly suggested that the simple nearest neighbor algorithm is very difficult to beat for most time series problems. While this may be considered good news, given the simplicity of implementing the nearest neighbor algorithm, there are some negative consequences of this. First, the nearest neighbor algorithm requires storing and searching the entire dataset, resulting in a time and space complexity that limits its applicability, especially on resource-limited sensors. Second, beyond mere classification accuracy, we often wish to gain some insight into the data.

In this work we introduce a new time series primitive, *time series shapelets*, which addresses these limitations. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. As we shall show with extensive empirical evaluations in diverse domains, algorithms based on the time series shapelet primitives can be interpretable, more accurate and significantly faster than state-of-the-art classifiers.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications – Data Mining

General Terms

Algorithms, Experimentation

1. INTRODUCTION

While the last decade has seen a huge interest in time series classification, to date the most accurate and robust method is the simple nearest neighbor algorithm [4][12][14]. While the nearest neighbor algorithm has the advantages of simplicity and not requiring extensive parameter tuning, it does have several important disadvantages. Chief among these are its space and time requirements, and the fact that it does not tell us anything about *why* a particular object was assigned to a particular class.

In this work we present a novel time series data mining primitive called *time series shapelets*. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. While we believe shapelets can have many uses in data mining, one obvious implication of them is to mitigate the two weaknesses of the nearest neighbor algorithm noted above.

Because we are defining and solving a new problem, we will take some time to consider a detailed motivating example. Figure 1 shows some examples of leaves from two classes, *Urtica dioica* (stinging nettles) and *Verbena urticifolia*. These two plants are commonly confused, hence the colloquial name “false nettle” for *Verbena urticifolia*.

Figure 1: Samples of leaves from two species. Note that several leaves have the insect-bite damage

Suppose we wish to build a classifier to distinguish these two plants; what features should we use? Since the intra-variability of color and size within each class completely dwarfs the inter-variability between classes, our best hope is based on the shapes of the leaves. However, as we can see in Figure 1, the differences in the global shape are very subtle. Furthermore, it is very common for leaves to have distortions or “occlusions” due to insect damage, and these are likely to confuse any global measures of shape. Instead we attempt the following. We first convert each leaf into a one-dimensional representation as shown in Figure 2.

Figure 2: A shape can be converted into a one dimensional “time series” representation. The reason for the highlighted section of the time series will be made apparent shortly

Such representations have been successfully used for the classification, clustering and outlier detection of shapes in recent years [8]. However, here we find that using a nearest neighbor classifier with either the (rotation invariant) Euclidean distance or Dynamic Time Warping (DTW) distance does not significantly outperform random guessing. The reason for the poor performance of these otherwise very competitive classifiers seems to be due to the fact that the data is somewhat noisy (i.e. insect bites, and different stem lengths), and this noise is enough to swamp the subtle differences in the shapes.

KDD'09, June 29–July 1, 2009, Paris, France