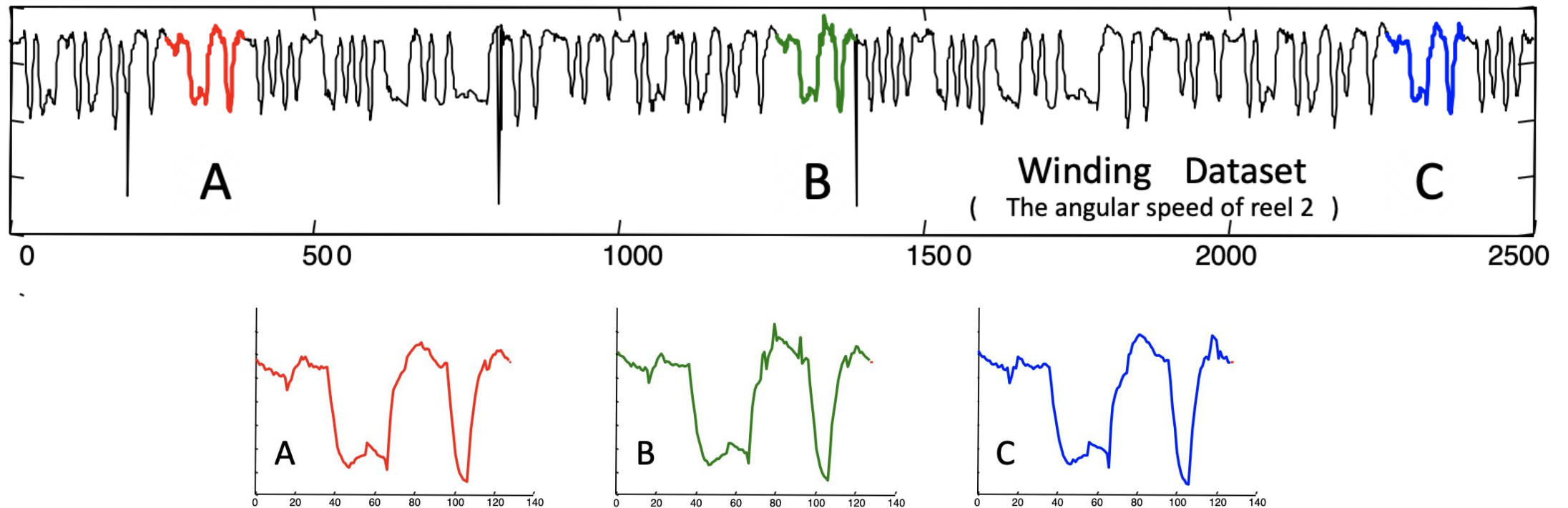


Time Series - Motif Discovery

Time Series Motif Discovery

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

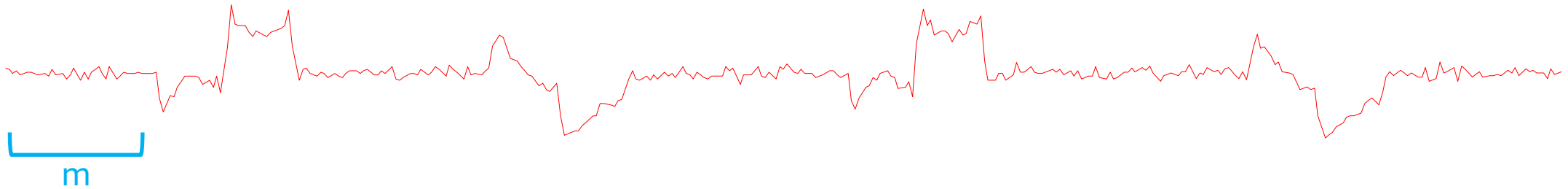


Why Find Motifs?

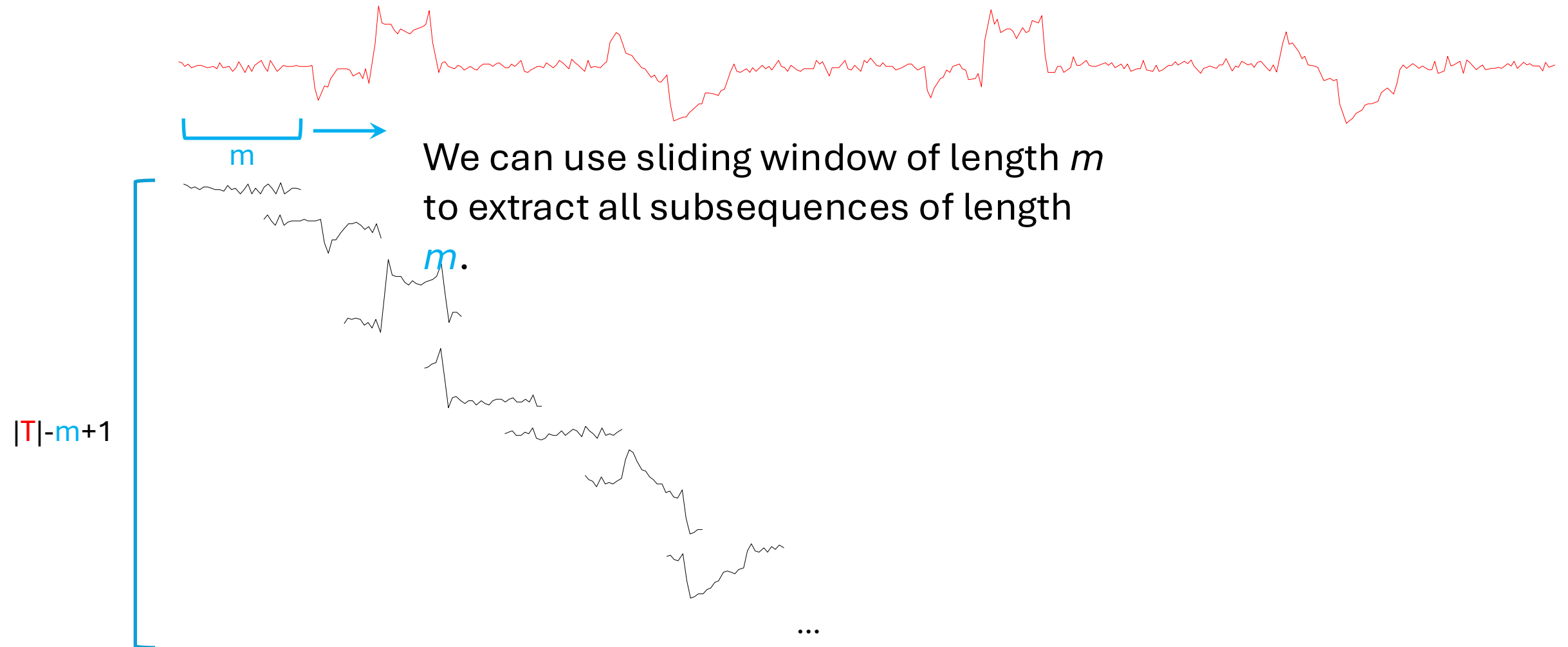
- Mining **association rules** in TS requires the discovery of motifs. These are referred to as **primitive shapes** and **frequent patterns**.
- Several **TS classifiers** work by constructing typical **prototypes** of each class. These prototypes may be considered motifs.
- Many **TS anomaly detection**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

Matrix Profile

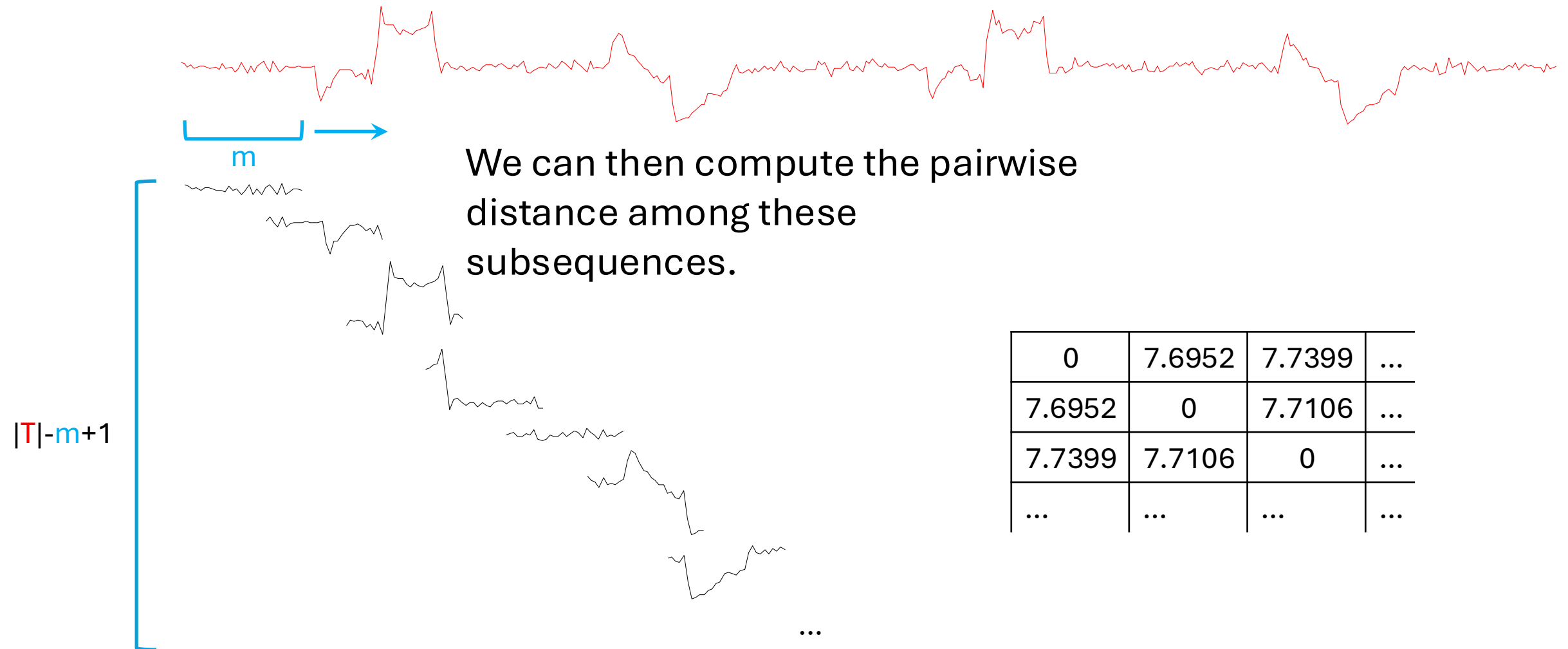
- The Matrix Profile (MP) is a data structure that annotates a TS and can be exploited for many purposes: e.g. efficient Motif Discovery.
- Given a time series, T and a desired subsequence length, m .



Matrix Profile

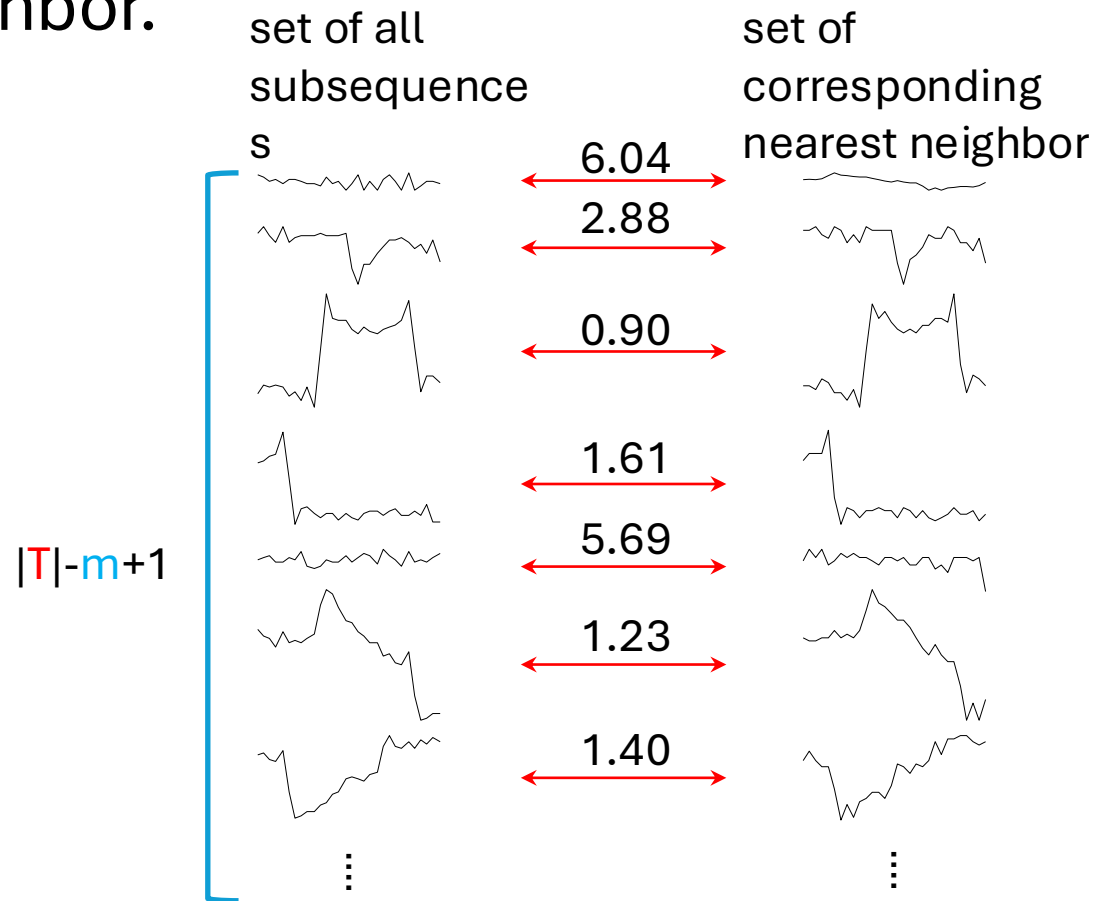


Matrix Profile



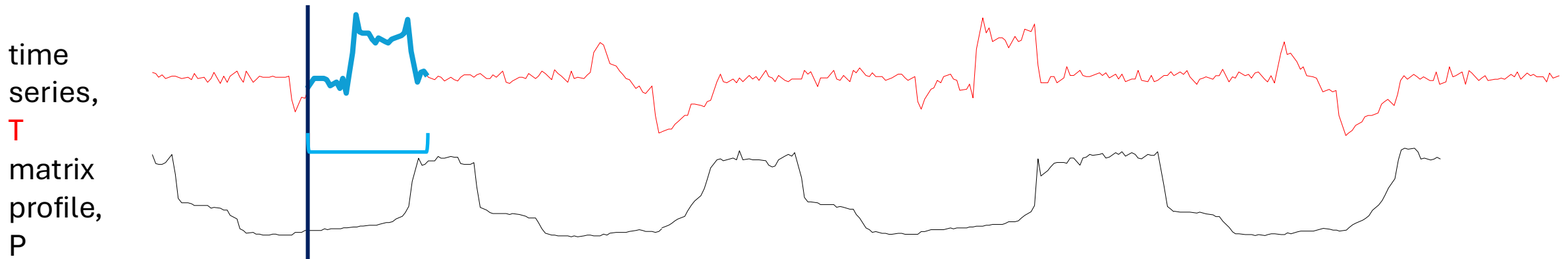
Matrix Profile

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



Matrix Profile

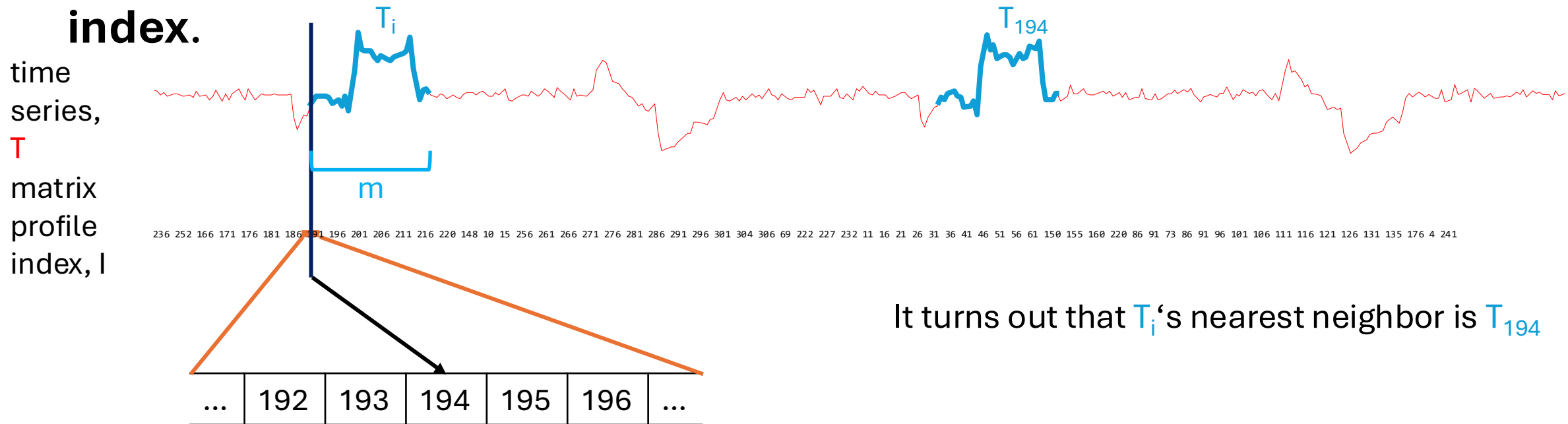
- The **distance** to the corresponding **nearest neighbor** of each subsequence can be stored in a vector called **matrix profile P**.



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

Matrix Profile

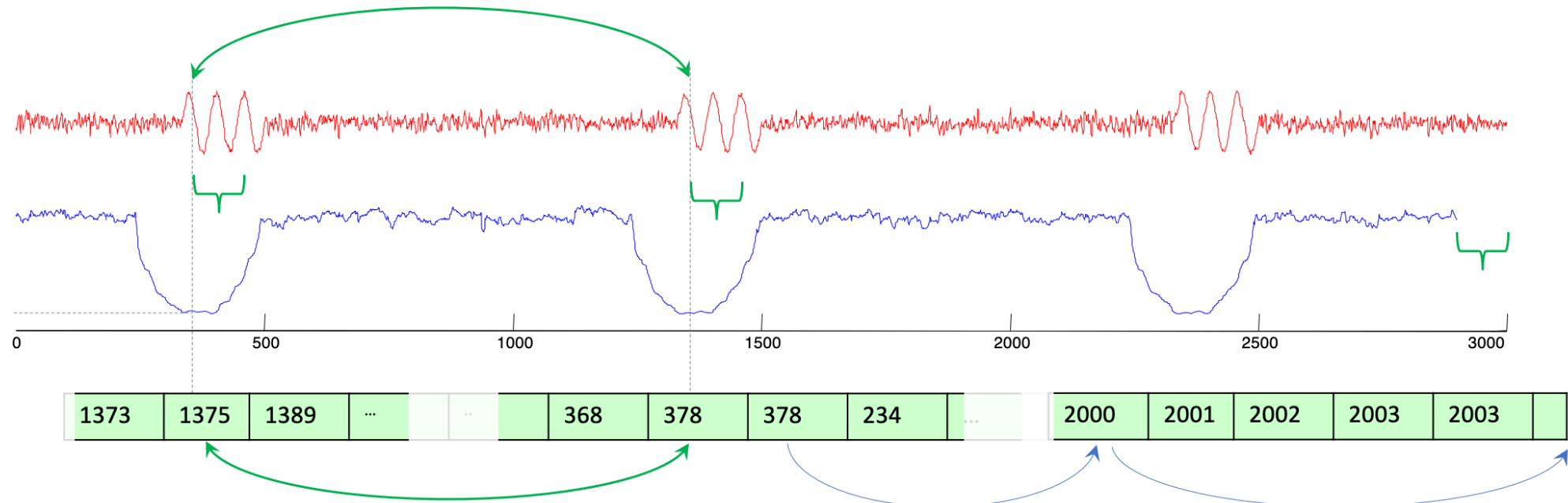
- The **index of corresponding nearest neighbor** of each subsequence is also stored in a vector called **matrix profile index**.



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

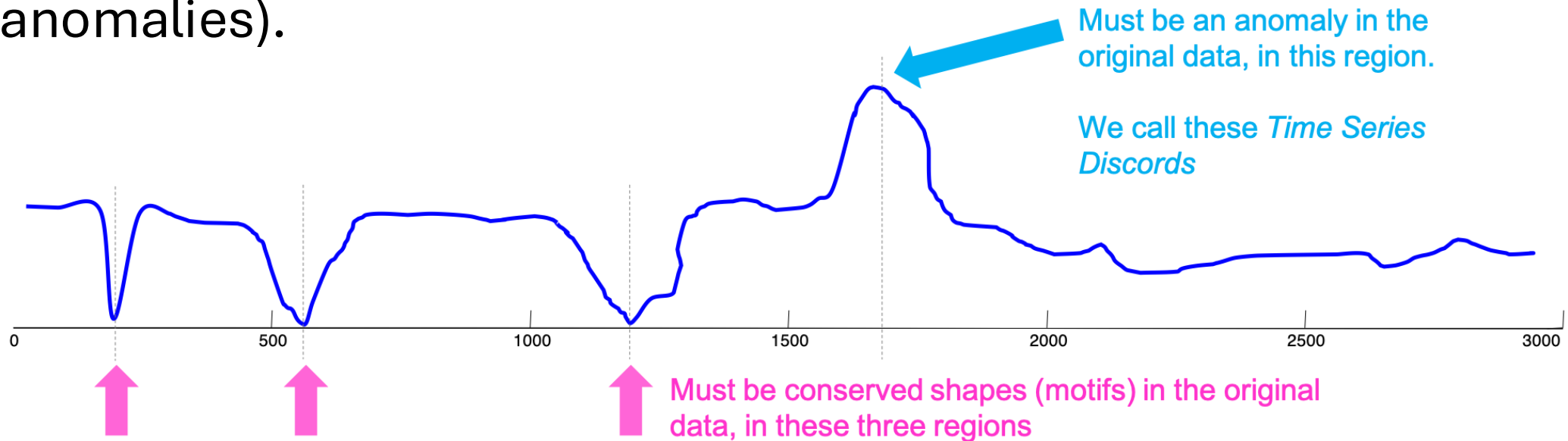
Matrix Profile

- 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.



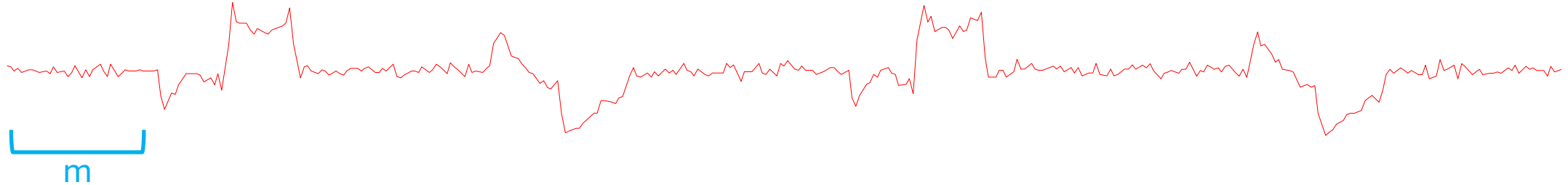
How to “read” a Matrix Profile

- 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).



How to Compute Matrix Profile?

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



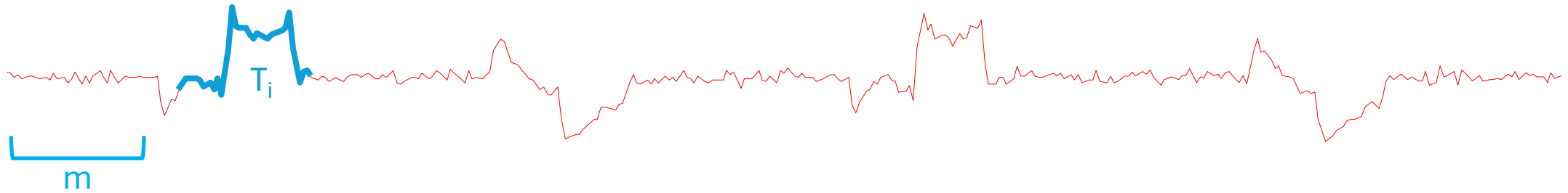
inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

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 .

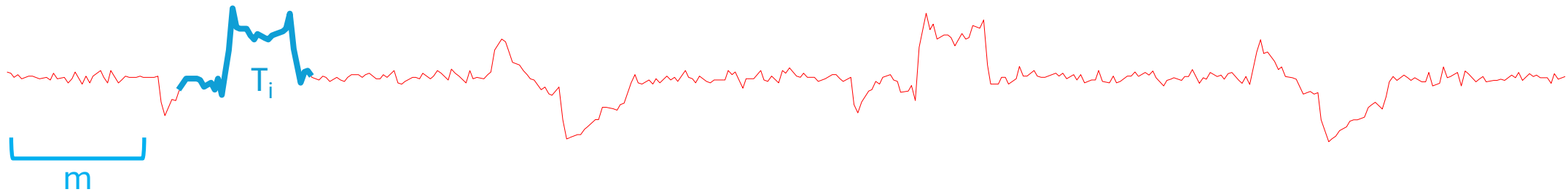


inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

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

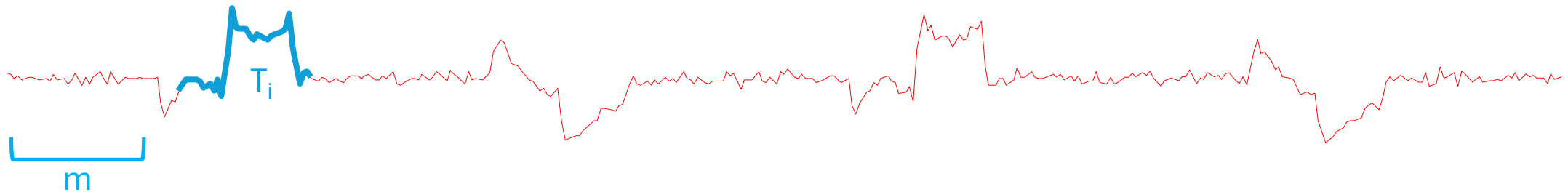
3	2	0	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



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
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

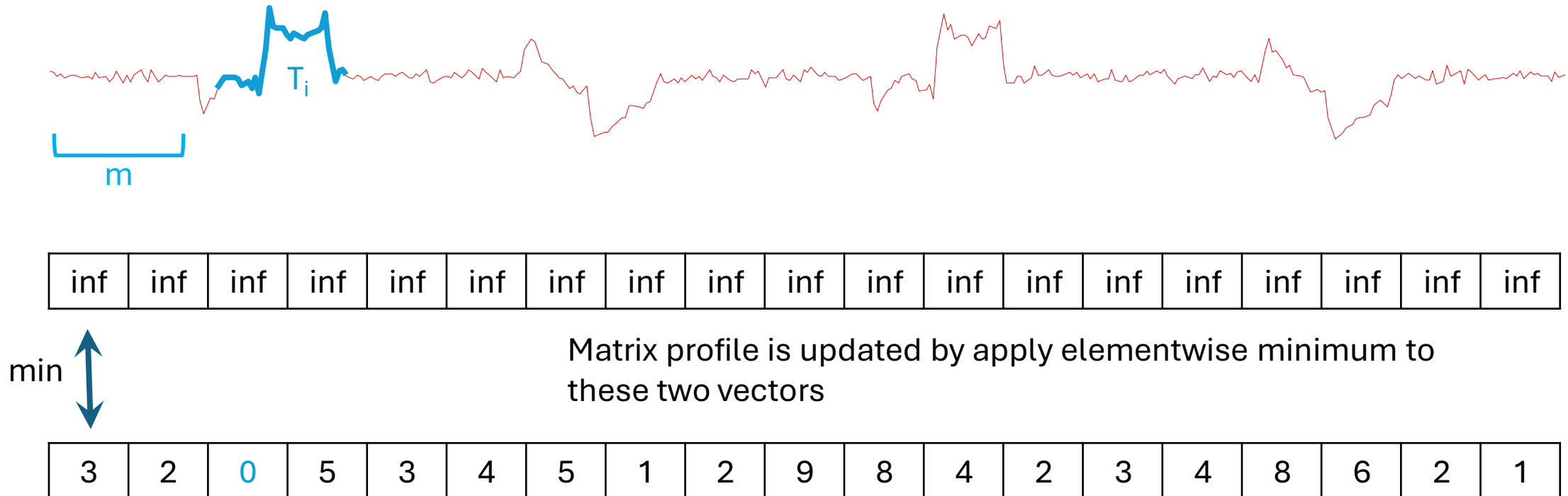
We compute the distances between T_i and every subsequences from T (time complexity = $O(|T|\log(|T|))$)
We then put the distances in a vector based on the position of the subsequences

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

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

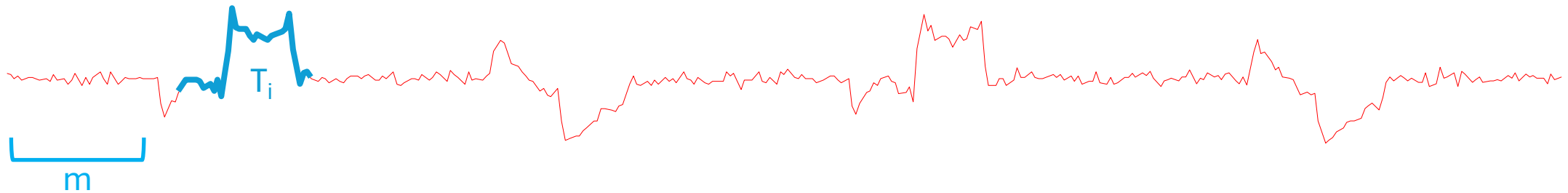
How to Compute Matrix Profile?

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



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
---	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

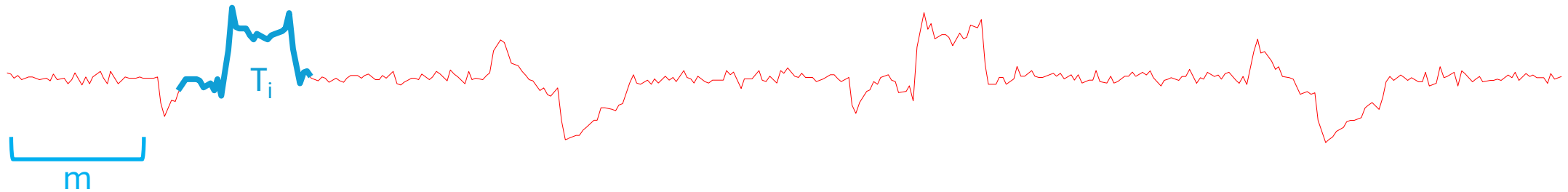
min
↕

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 .



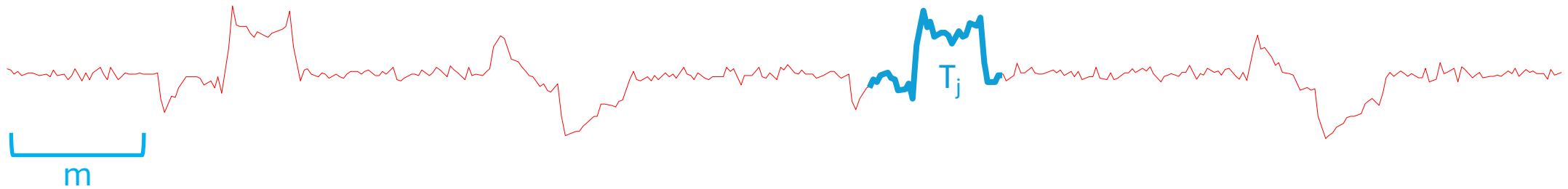
3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1
---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

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 .

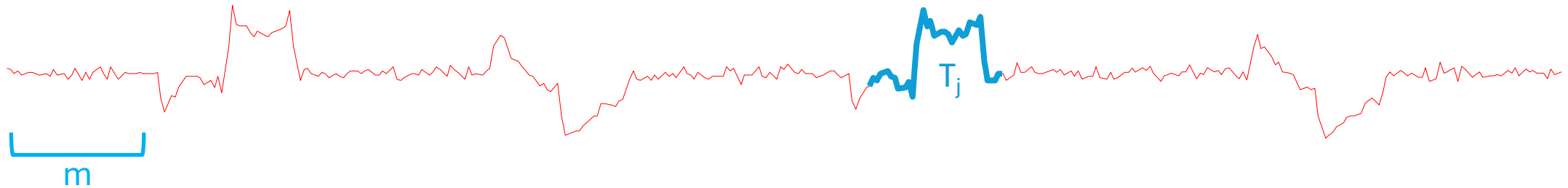


3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1
---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

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

How to Compute Matrix Profile?

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



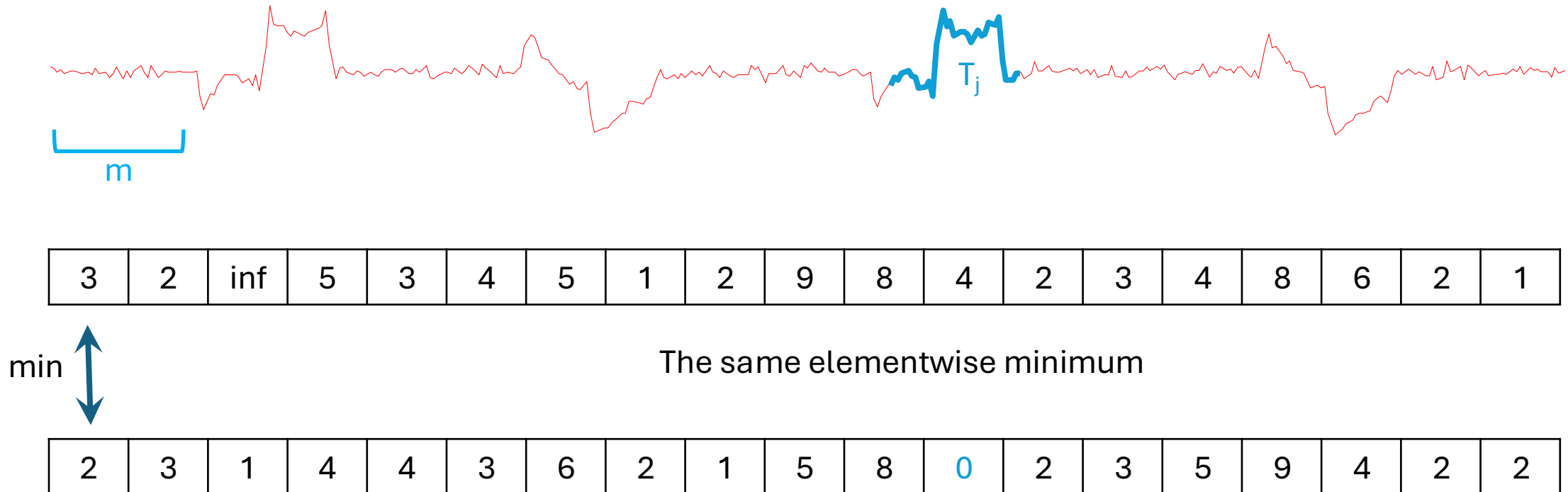
3	2	inf	5	3	4	5	1	2	9	8	4	2	3	4	8	6	2	1
---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

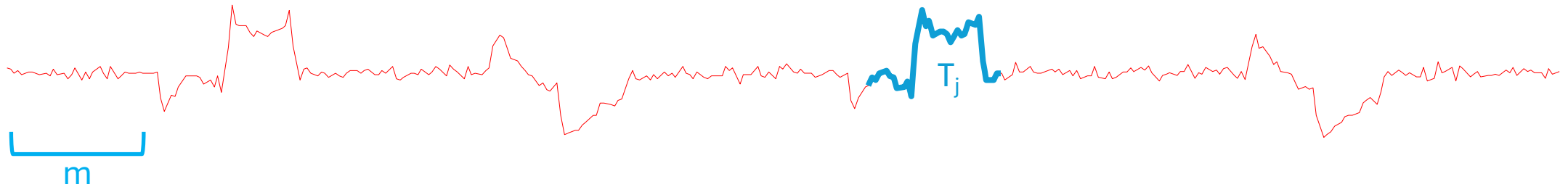
How to Compute Matrix Profile?

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



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
---	---	-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

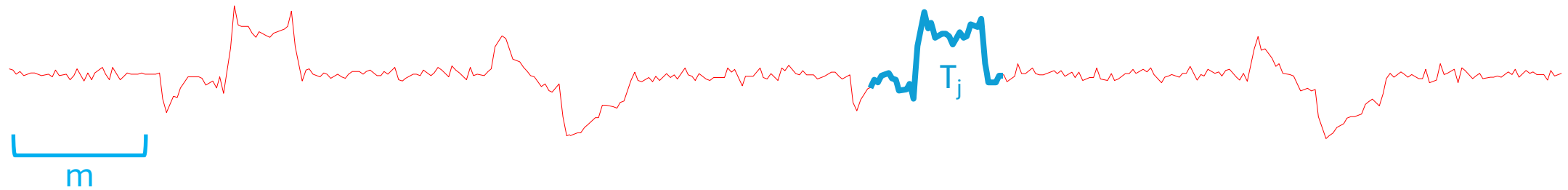
min

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

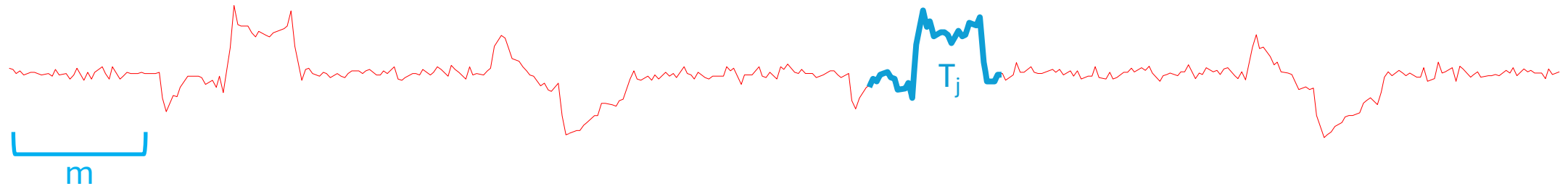
min
↕

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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

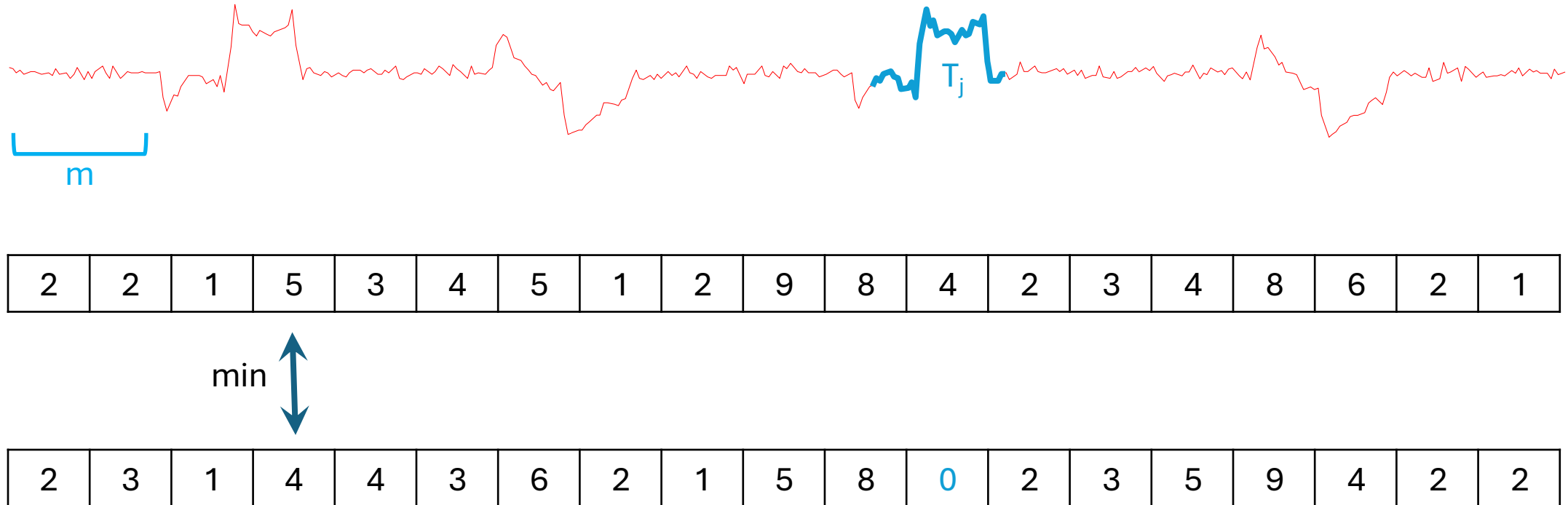
min

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?

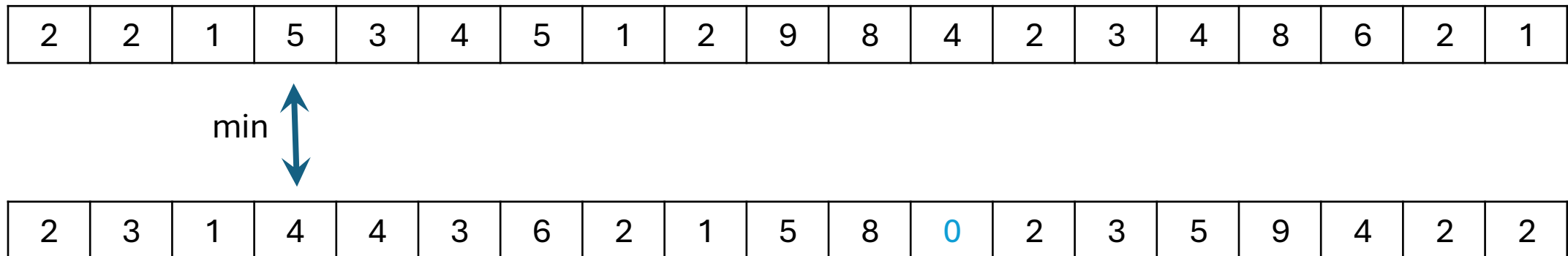
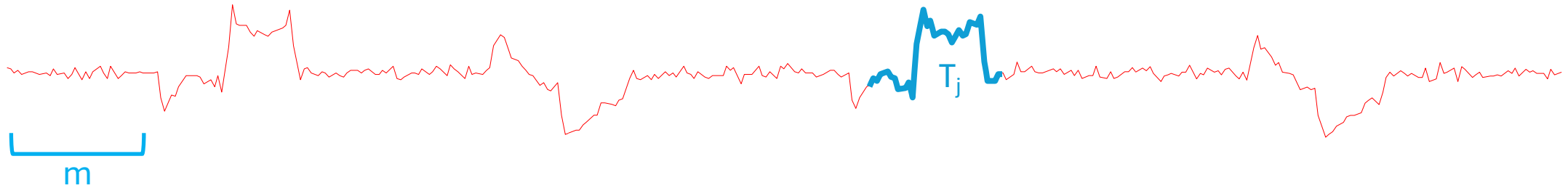
- Given a time series, T and a desired subsequence length, m .



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

How to Compute Matrix Profile?

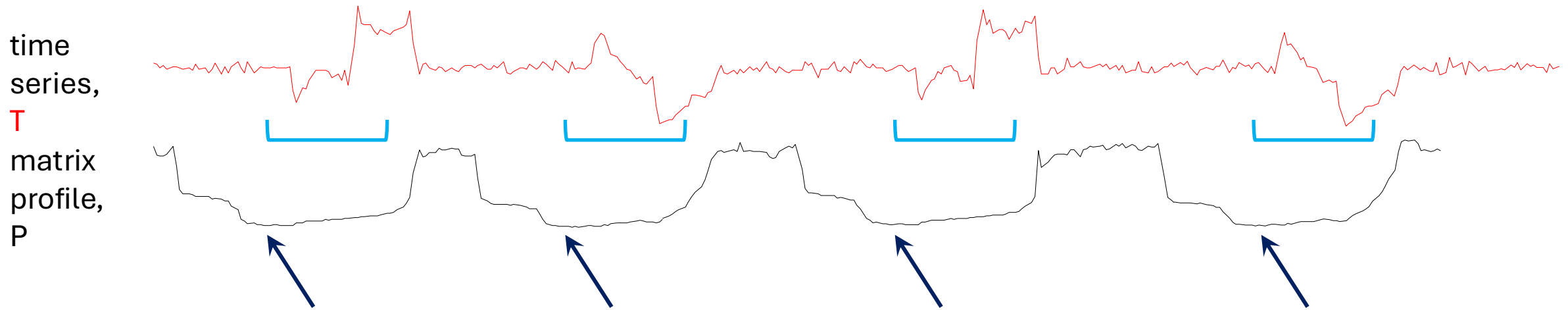
- Given a time series, T and a desired subsequence length, m .



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

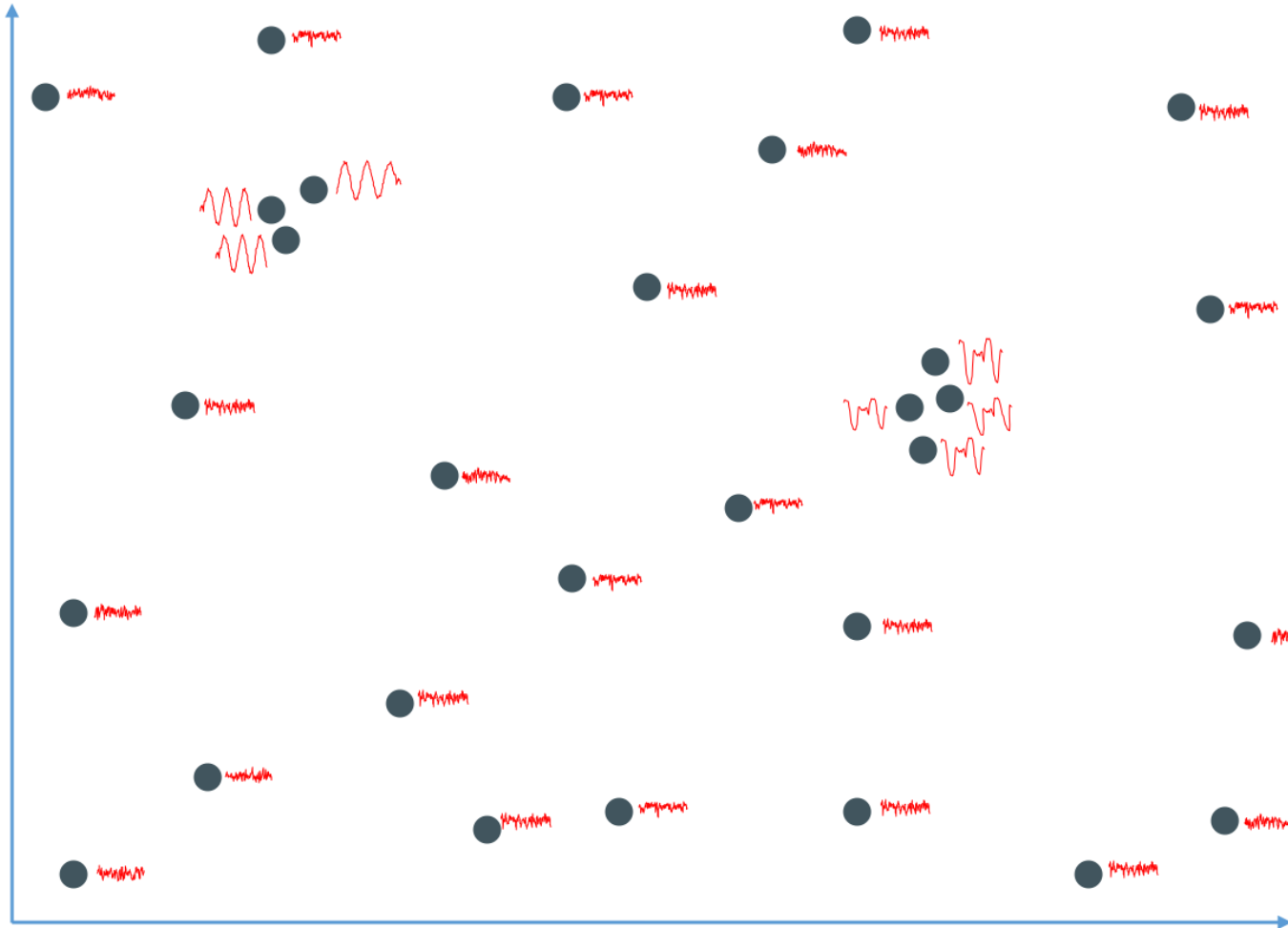
The overall time complexity is $O(|T|^2\log(|T|))$

Motif Discovery From Matrix Profile

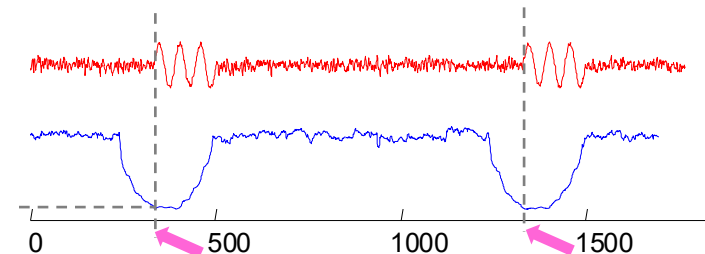


Local minimums are corresponding to motifs

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.



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

Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets
Chin-Chia Michael Yeh, Yan Zhi, Linduolu Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Duu, Diego Furtado Silva, Abdullahi Museen, and Emomou Kevgh
University of California, Riverside, Universidade de São Paulo, University of New Mexico
{mym003, yzhi015, luluo001, shoguo001, yding007, lduu0011}@ucr.edu, diego.furtado@unesp.br, museen@unm.edu, emomou@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 *naïve-join* 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* problems 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, chapter 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 132,880,692,060 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 6.2 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 *ε*-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 metrics space APSS algorithms require building and tuning spatial access methods and/or hash functions.
- Our algorithm requires an inconsequential space overhead, just $O(\epsilon)$ 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.

Given all these features, our algorithm has implications for many time series data mining tasks [5][11][28].

The rest of the paper is organized as follows. Section II reviews related work and introduces the necessary background materials and definitions. In Section III we introduce our algorithm and its anytime and incremental variants. Section IV sees a detailed empirical evaluation of our algorithm and shows its implications for many data mining tasks. Finally, in Section V we offer conclusions and directions for future work.

II. RELATED WORK AND BACKGROUND

The basic variant of *similarity join* problem we are interested in is as follows: Given a collection of data objects, retrieve the nearest neighbor for every object.

Other common variants include retrieving the top- K nearest neighbors or the nearest neighbor for each object if that neighbor is within a user-specified threshold. ϵ (back variations are trivial generalizations of our proposed algorithm, so we omit them from further discussion). The latter variant results in a much easier problem, provided that the threshold is small. For example, [1] notes that virtually all research efforts “exploit a similarity threshold more aggressively in order to limit the set