tsmp: An R Package for Time Series with Matrix Profile

by Francisco Bischoff and Pedro Pereira Rodrigues

Abstract This article describes **tsmp**, an R package that implements the MP concept for TS. The **tsmp** package is a toolkit that allows all-pairs similarity joins, motif, discords and chains discovery, semantic segmentation, etc. Here we describe how the **tsmp** package may be used by showing some of the use-cases from the original articles and evaluate the algorithm speed in the R environment. This package can be downloaded at `https://CRAN.R-project.org/package=tsmp`.

Introduction: time series data mining

A TS is a sequence of real-valued numbers indexed in time order. Usually, this sequence is taken in a regular period of time, which will be assumed to be true in this context. The interests in TS data mining have been growing along with the increase in available computational power. This kind of data is easily obtained from sensors (e.g., ECG), (ir)regular registered data (e.g., weekly sales, stock prices, brachial blood pressure). Even other kinds of data can be converted to TS format, such as shapes (Wei et al., 2006) and DNA sequences (Shieh and Keogh, 2008). TS are generally large, high dimensional and continuously updated which requires algorithms fast enough in order to be meaningful. Besides, unlike other kinds of data, which usually have exact answers, TS are usually analysed in an approximated fashion.

These characteristics have been challenging researchers to find faster and more accurate methods to retrieve meaningful information from TS. This required one or more of these methods: dimensionality reduction, constraints, domain knowledge, parameter tweaks. Only afterwards could the data mining tasks be applied in feasable time. Typical tasks include motif and discord discovery, subsequence matching, semantic segmentation, rule discovery, similarity search, anomaly detection, clustering, classification, indexing, etc. (Fu, 2011).

This paper describes the **tsmp** package (Bischoff, 2018) which uses a novel approach to TS data mining: the MP Yeh et al. (2017b), which is based on the APSS (also known as similarity join). The APSS' task is to, given a collection of data objects, retrieve the nearest neighbour for each object. The remaining part of this paper is organised as follows: In Section 2.2 we describe the reasoning behind the MP, in Section 2.3 we present the **tsmp** package with examples, in Section 2.4 we compare the performance of the R implementation, and in Section 2.5 we conclude with a brief discussion.

The matrix profile

The reader may be aware of what a DM is. It is widely used in TS for clustering, classification, motif search, etc. But, even for modestly sized datasets, the algorithms can take months to compute even with speed-up techniques such as indexing (Shieh and Keogh, 2008; Fu et al., 2008), lower-bounding (Keogh and Ratanamahatana, 2005), data discretization (Lin et al., 2003) and early abandoning (Faloutsos et al., 1994). At best, they can be one or two orders of magnitude faster.

The MP is an ordered vector that stores the Euclidean distance between each pair within a similarity join set. One (inefficient) way would be to use the full DM of every iteration of a sliding window join and retrieve just the smallest (non-diagonal) value of each row. The MP also has a companion vector called PI, that gives us the position of the nearest neighbour of each subsequence.

This method has a host of interesting and exploitable properties. For example, the highest point on the MP corresponds to the TS discord, the (tied) lowest points correspond to the locations of the best TS motif pair, and the variance can be seen as a measure of the TS complexity. Moreover, the histogram of the values in the MP is the exact answer to the TS density estimation. Particularly, it has implications for TS motif discovery, TS joins, shapelet discovery (classification), density estimation, semantic segmentation, visualisation, rule discovery, clustering, etc. (Yeh et al., 2017b).

Some of the advantages/features of this method:

- 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.
- It requires an inconsequential space overhead, just $O(n)$ with a small constant factor.

- It is extremely *scalable*, and for *massive* 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.
- It provides *full joins*, eliminating the need to specify a similarity threshold, which is a near-impossible task in this domain.
- It is *parallelizable*, both on multicore processors and in distributed systems (Zhu et al., 2016).

The tsmp package

The **tsmp** package provides several functions that allow for an easy workflow using the MP concept for TS mining. The package is available from the CRAN at `https://CRAN.R-project.org/package=tsmp`. In Section 2.3.1 we explain how to install this package. In Section 2.3.2 we describe the syntax for the main functions in **tsmp**, giving an example of a particular model. In Section 2.3.3 we will further explain the available algorithms for MP computation and its current use. In Section 2.3.4 we show some examples of MP application for data mining.

Installation

The **tsmp** package can be installed in two ways:

The release version from CRAN:

```
install.packages("tsmp")
```

or the development version from GitHub:

```
# install.packages("devtools")
devtools::install_github("franzbischoff/tsmp")
```

Input arguments and example

The **tsmp** has a simple and intuitive workflow. First, you must compute the MP of the desired TS. Depending on the task, the user might want to follow one of three paths: univariate self-join, AB-join or multivariate self-join. One exception is the SiMPle algorithm that is a multivariate AB-join and will be explained in Section 2.3.3.

The main function is `tsmp()`, which has the following usage:

```
tsmp(..., window_size, exclusion_zone = 1/2,
  mode = c("stomp", "stamp", "simple", "mstomp", "scrimp"),
  verbose = 2, s_size = Inf, must_dim = NULL, exc_dim = NULL,
  n_workers = 1, .keep_data = TRUE)
```

The first argument `ellipsis` (the three dots) receives one or two TS. For self-joins, the user must input just one TS; two for AB-joins. Multivariate TS may be input as a matrix where each column represents one dimension. Alternatively, the user may input the Multivariate TS as a list of vectors. The second argument `window_size` is the size of the sliding window. These are the most basic parameters you need to set.

Further parameters are:

- `exclusion_zone`, is an important parameter for self-joins. This is used to avoid trivial matches and is a modifier of the `window_size`, i.e., for an `exclusion_zone` of 1/2, and `window_size` of 50, internally the result will be 25.
- `mode`, here the user may choose the algorithm used for the MP calculation. `stomp`, `stamp` and `scrimp` return equal results, although differing in some practical attributes, and they will be further explained in Section 2.3.3. `mstomp` is designed for Multivariate TS self-join only. `simple` is designed for Multivariate TS for self-join and AB-join, which will also be further explained in Section 2.3.3.
- `verbose`, controls the verbosity of the function. 0 means no feedback, 1 means text messages only, 2 (the default) means text messages and progress bar, and 3 also plays a sound when finished.

- `s_size`, controls the *anytime* algorithms. This is just a way to end the algorithm in a controlled manner because the *anytime* algorithms can be stopped *anytime* and the result will be returned.
- `must_dim`, is an optional argument for the `mstomp` algorithm. See next item.
- `exc_dim`, as `must_dim`, is an optional argument for the `mstomp` algorithm. These arguments control which dimensions must be included and which must be excluded from the multidimensional MP.
- `n_workers`, controls how many threads will be used for the `stamp`, `stomp`, and `mstomp`. Note that for small datasets, multiple threads add an overhead that makes it slower than just one thread.
- `.keep_data`, `TRUE` by default, keeps the input data inside the output object. This is useful for chained commands.

Example data

We think that the best and simple example to demonstrate the **tsmp** package is the motif search.

The **tsmp** package imports the `%>%` (pipe) operator from the **magrittr** package that makes the **tsmp** workflow easier.

The following code snippet shows an example of the workflow for motif search:

```
R> data <- mp_fluss_data$walkjogrun$data
R> motifs <- tsmp(data, window_size = 80, exclusion_zone = 1/2) %>%
+    find_motif(n_motifs = 3, radius = 10, exclusion_zone = 20) %T>% plot()
```

The `find_motif()` function is an S3 class that can receive as the first argument the output of `tsmp()` function as a univariate or multivariate MP. This allows us to use the pipe operator easily. The `plot()` function is also an S3 class extension for plotting objects from the **tsmp** package and works seamlessly.

Computational methods

There are several methods to compute the MP. The reason for that is the unquenchable need for speed of the UCR's researchers. Before starting, let's clarify that the time complexity of a brute force algorithm has a time complexity of $O(n^2m)$, for n being the length of the reference TS and m the length of the sliding window (query) that is domain dependent.

STAMP

This was the first algorithm used to compute the MP. It uses the MASS (Mueen et al., 2015) as the core algorithm for calculating the similarity between the query and the reference TS, called the DP. The ultimate MP comes from merging the element-wise minimum from all possible DP. This algorithm has the time complexity of $O(n^2 \log n)$ and space complexity of $O(n)$ (Yeh et al., 2017b). The *anytime* property is achieved using a random approach where the best-so-far MP is computed using the DP that have been already calculated.

STOMP

This was the second algorithm used to compute the MP. It also uses the MASS to calculate the DP but only for the first iteration of each batch. The researchers noticed that they could reuse the values calculated of the first DP to make a faster calculation in the next iterations. This results on a time complexity of $O(n^2)$, keeping the same space complexity of $O(n)$. This algorithm is also suitable for a GPU framework (although this was not yet implemented in **tsmp** package) (Zhu et al., 2016). The main drawback of STOMP compared with STAMP is the lack of the *anytime* property. In scenarios where a fast convergence is needed (e.g., finding the top-k motifs) it may be required only 5% of the MP computation to provide a very accurate approximation of the final result.

SCRIMP

The SCRIMP algorithm is still experimental at the time of this article. It combines the best features of STOMP and STAMP, having a time complexity of $O(n^2)$ and the *anytime* property (UCR, 2016).

SiMPle

The SiMPle algorithm is a variation designed for music analysis and exploration (Silva et al., 2018). Internally it uses STOMP for MP computation and allows multidimensional self-joins and AB-joins. The resulting MP is computed using all dimensions. One major difference is that it doesn't apply any z-normalization on the data, since for music domain this would result in spurious similarities.

mSTOMP

The mSTOMP algorithm was designed to motif search in multidimensional data (Yeh et al., 2017a). Performing motif search on *all* dimensions is almost guaranteed to produce meaningless results, so this algorithm, differently from SiMPle, doesn't compute the MP using all dimensions naïvely, but the d-dimensional MP for every possible setting of d, simultaneously, in $O(dn^2 \log d)$ time and $O(dn)$ space. The resulting MP allow motif search in multiple dimensions and also to identify which dimensions are relevant for the motifs founded.

Data mining tasks

Motif search

In Section 2.3.2 we have shown a basic example of the workflow for motif search. Let's take a look at the result of that code:

```
R> motifs

Matrix Profile
--------------
Profile size = 9922
Window size = 80
Exclusion zone = 40
Contains 1 set of data with 10001 observations and 1 dimension

Motif
-----
Motif pairs found = 2
Motif pairs indexes = [584, 741] [4799, 5329]
Motif pairs neighbors = [2948, 9900, 8265] [7023, 8861, 2085, 248]
```

As we can see, this is a summary that **tsmp** package automatically generates from the resulting object. One nice property is that the object always holds the original MP and by default also holds the input data so that you can keep mining information from it. If the dataset is too big or you are concerned about privacy, you may set the argument `.keep_data = FALSE`.

In addition to this summary, you can see the results using `plot()` in Figure 1:

```
R> plot(motifs, type = "matrix")
```

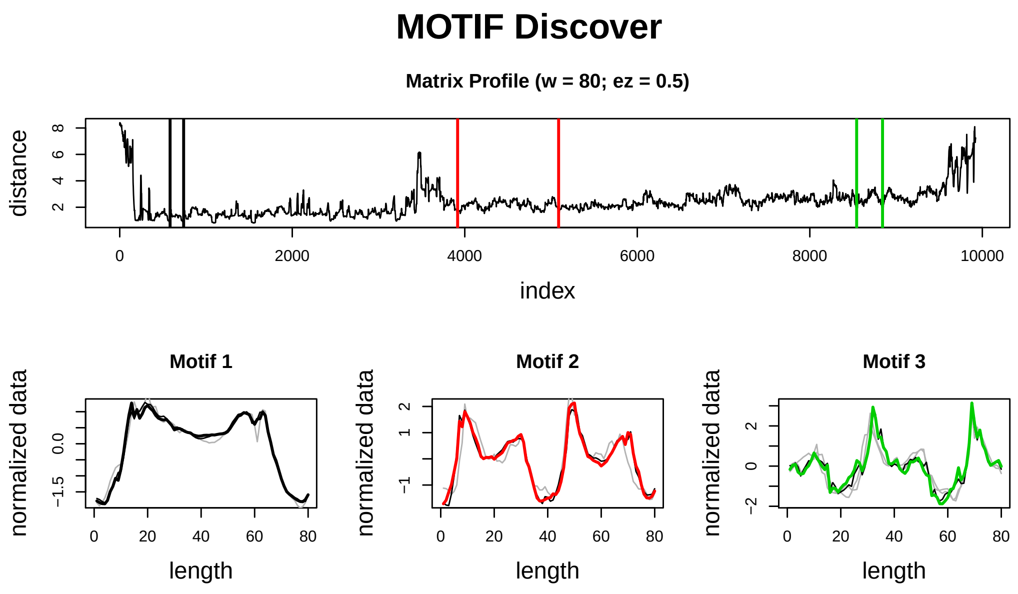

Figure 1: The upper graphic shows the computed MP with each motif pair as a coloured vertical bar. The lower graphics show each motif in colour and the founded neighbours in grey.

This dataset is the *WalkJogRun* PAMAP's dataset (Reiss and Stricker, 2012). It contains the recording of human movements in three states, walking, jogging and running. As we can see, the plot shows the motifs of each state. Experienced readers might say that this is not the purpose of motif search, and we agree. The result shown here was achieved using a large `radius` and `exclusion_zone` to force the algorithm to look for distant motifs. Semantic segmentation is the proper algorithm for this task, and we will show this in the next section.

Semantic segmentation

As previously explained, the resulting object holds the original data and MP. So let's save some time and use the resulting object from the last section to try to find where the human subject started to jog and to run:

```
R> segments <- motifs %>% fluss(num_segments = 2)
R> segments

Matrix Profile
--------------
Profile size = 9922
Window size = 80
Exclusion zone = 40
Contains 1 set of data with 10001 observations and 1 dimension

Arc Count
---------
Profile size = 9922
Minimum normalized count = 0.063 at index 3448

Fluss
-----
Segments = 2
Segmentation indexes = 3448 6687
```

We can see that this object now holds information of the FLUSS algorithm (Gharghabi et al., 2017), but the motif information is still there and can be retrieved using `as.motif()`. In Figure 2 we can see the graphic result of the segmentation.

```
R> plot(segments, type = "data")
```

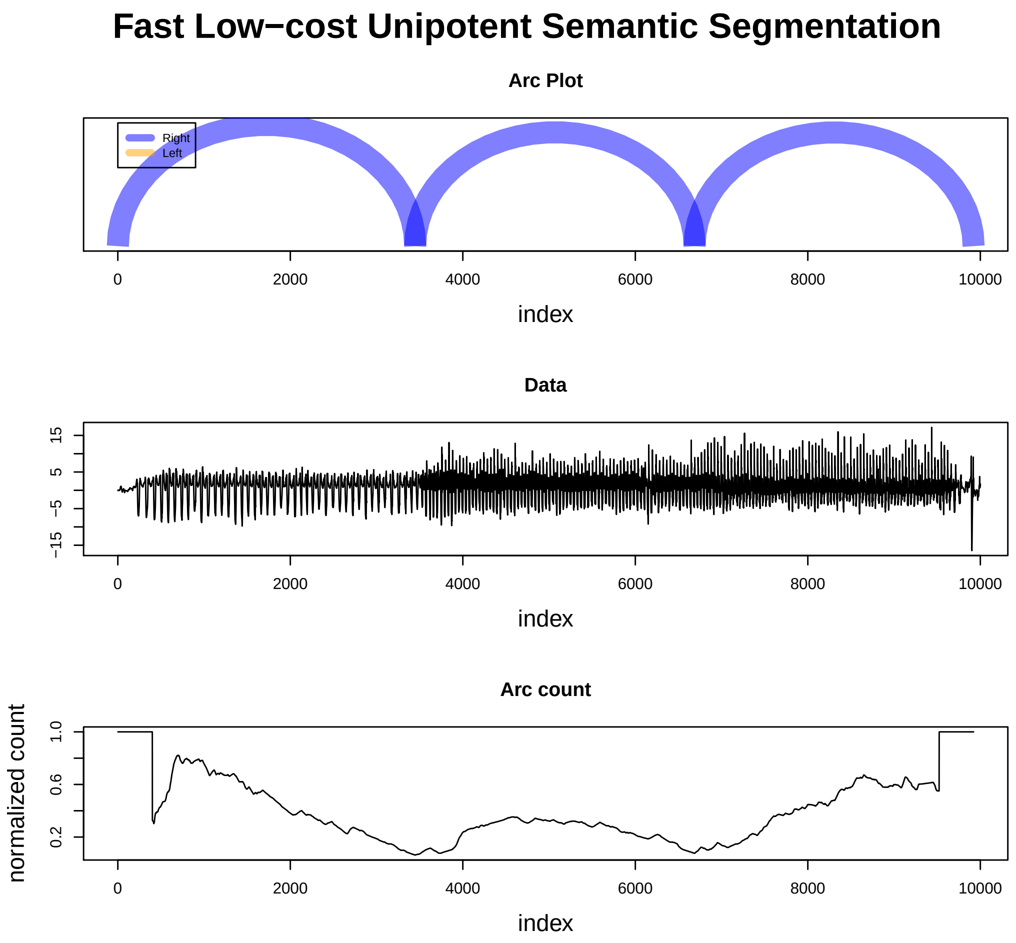

Figure 2: Semantic segmentation using MP. The upper graphic shows the arc plot of predicted semantic changes (ground truth is 3800 and 6800). The middle graphic shows the data. The lower graphic shows the normalised arc counts with correction for the "edge-effect" (Gharghabi et al., 2017).

Time series chains

As a final example of practical application, let's search for a new kind of primitive: time series chains (Zhu et al., 2018a). This algorithm looks for patterns that are not just similar but evolve through time. The dataset used in this example is a record of the Y-axis of a mobile phone accelerometer while placing it on a walking subject's pocket (Hoang et al., 2015). The authors of this dataset wanted to analyse the stability of the mobile phone as it slowly settles in the pocket. This is a good example of a pattern that changes through time. Let's start with the workflow for this example:

```
R> chains <- mp_gait_data %>% tsmp(window_size = 50, exclusion_zone = 1/4,
+    verbose = 0) %>% find_chains()
R> chains

Matrix Profile
--------------
Profile size = 855
Window size = 50
Exclusion zone = 13
Contains 1 set of data with 904 observations and 1 dimension

Chain
-----
Chains found = 58
```

```
Best Chain size = 6
Best Chain indexes = 148 380 614 746 778 811
```

Here we see that the algorithm found 58 chains. *Id est*, it found 58 evolving patterns with at least three elements, and the best one is presented in the last line, a chain with six elements. Figure 3 shows the patterns discovered.

```
R> plot(chains, ylab = "")
```

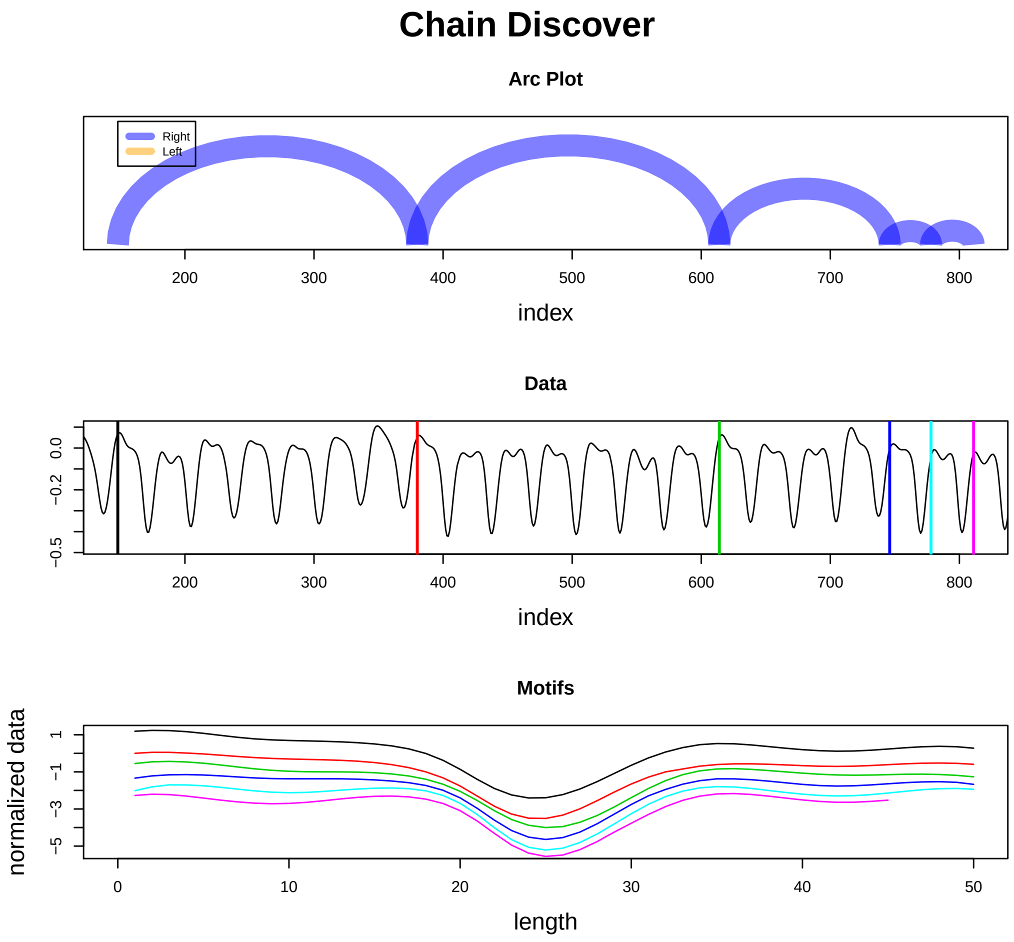

Figure 3: Finding evolving patterns using MP. The upper graphic shows the arc plot of the discovered patterns. The middle graphic shows the data and the position of every pattern as a vertical coloured line. The lower graphic shows the patterns for comparison. They are y-shifted for visualisation only.

Speed

While this new method for TS data mining is extremelly fast, we have to take into consideration that the R environment is not as fast as a low-level implementation such as C/C++. In Table 1 we present the comparison to the MATLAB version that is available at the UCR. Yeh et al. (2017b) shows that the slowest algorithm (STAMP) can be hundreds of times faster than the MK algorithm (the fastest known *exact* algorithm for computing TS motifs) (Yoon et al., 2015), while the R implementation is just 1.65 to 8.04 times slower than MATLAB's, which is not a problem for an R researcher.

```
R> set.seed(2018)
R> data <- cumsum(sample(c(-1, 1), 40000, TRUE))
```

Algorithm	R Time*	MATLAB Time*	Threads
scrimp	45.30	27.49	1
stomp	52.72	10.27	8
stomp	136.01	16.91	1
stamp	140.25	55.57	8
stamp	262.03	113.18	1

Table 1: Performances of R and MATLAB implementations on an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz using a random walk dataset. *Median of 5 trials, in seconds.

Conclusion

The examples in Section 2.3.4 show how straightforward the usage of **tsmp** package is. Regardless, these examples are just a glimpse of the potential of the MP. Several new algorithms based on MP are being developed and will be gradually implemented in the **tsmp** package (Linardi et al., 2018; Zhu et al., 2018b; Gharghabi et al., 2018; Imani et al., 2018). Yeh et al. (2017a) for example, have developed an algorithm to allow MDS visualisation of motifs. Gharghabi et al. (2018) have developed a new distance measure that better suits repetitive patterns (Imani et al., 2018).

The MP has the potential to revolutionise the TS data mining due to its generality, versatility, simplicity and scalability (UCR, 2016). All existing algorithms for MP have been proven to be flexible to be used in several domains using very few parameters and they are also robust, showing good performance with dimensionality reduced data and noisy data. In addition, a yet to be published article shows a fantastic score of $> 10^{18}$ pairwise comparisons a day using GPU for motif discovery (Zimmerman et al., 2018).

The **tsmp** package is the first known MP toolkit available on any statistical language, and we hope it can help researchers to better mining TS and also to develop new methods based on MP.

Acknowledgements

We would like to thank the researchers from UCR for their contribution and permission to use their base code to be implemented in this package. Particularly to Prof. Eamonn Keogh whose work and assistance led to this project. We also acknowledge the participation in project NanoSTIMA (NORTE-01-0145-FEDER-000016) which was financed by the North Portugal Regional Operational Program (NORTE 2020) under the PORTUGAL 2020 Partnership Agreement and through the European Regional Development Fund (ERDF).

Acronyms

- APSS: all-pairs similarity search
- CRAN: Comprehensive R Archive Network
- DM: distance matrix
- DP: distance profile
- ECG: electrocardiogram
- FLUSS: fast low-cost unipotent semantic segmentation
- GPU: graphics processor unit
- MASS: Mueen's algorithm for similarity search
- MDS: multidimensional space
- MP: matrix profile
- mSTOMP: Multivariate scalable time series ordered-search matrix profile
- PI: profile index
- SCRIMP: Scalable column independent matrix profile
- SiMPle: Similarity matrix profile
- STAMP: Scalable time series anytime matrix profile
- STOMP: Scalable time series ordered-search matrix profile
- TS: time series
- UCR: University of California Riverside

Bibliography

F. Bischoff. *tsmp: Time Series with Matrix Profile*, 2018. URL https://CRAN.R-project.org/package=tsmp. R package version 0.3.2. [p76]

C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *ACM SIGMOD Record*, 23(2):419–429, jun 1994. ISSN 01635808. doi: https://doi.org/10.1145/191843.191925. [p76]

A. W. C. Fu, E. Keogh, L. Y. H. Lau, C. A. Ratanamahatana, and R. C. W. Wong. Scaling and time warping in time series querying. *VLDB Journal*, 17(4):899–921, 2008. ISSN 10668888. doi: https://doi.org/10.1007/s00778-006-0040-z. [p76]

T. C. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011. ISSN 09521976. doi: https://doi.org/10.1016/j.engappai.2010.09.007. [p76]

S. Gharghabi, Y. Ding, C.-C. M. Yeh, K. Kamgar, L. Ulanova, and E. Keogh. Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels. In *2017 IEEE International Conference on Data Mining (ICDM)*, volume 2017-Novem, pages 117–126. IEEE, nov 2017. ISBN 978-1-5386-3835-4. doi: https://doi.org/10.1109/ICDM.2017.21. [p80, 81]

S. Gharghabi, S. Imani, A. Bagnall, A. Darvishzadeh, and E. Keogh. Matrix Profile XII: MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018. [p83]

T. Hoang, D. Choi, and T. Nguyen. On the Instability of Sensor Orientation in Gait Verification on Mobile Phone. In *Proceedings of the 12th International Conference on Security and Cryptography*, pages 148–159. SCITEPRESS - Science and and Technology Publications, 2015. ISBN 978-989-758-117-5. doi: https://doi.org/10.5220/0005572001480159. [p81]

S. Imani, F. Madrid, W. Ding, S. Crouter, and E. Keogh. Matrix Profile XIII : Time Series Snippets : A New Primitive for Time Series Data Mining. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018. [p83]

E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, mar 2005. ISSN 0219-1377. doi: https://doi.org/10.1007/s10115-004-0154-9. [p76]

J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery - DMKD '03*, page 2, New York, New York, USA, 2003. ACM Press. ISBN 978-3-642-41397-1. doi: https://doi.org/10.1145/882085.882086. [p76]

M. Linardi, Y. Zhu, T. Palpanas, and E. Keogh. Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series. In *Proceedings of the 2018 International Conference on Management of Data - SIGMOD '18*, pages 1053–1066, New York, New York, USA, 2018. ACM Press. ISBN 9781450347037. doi: https://doi.org/10.1145/3183713.3183744. [p83]

A. Mueen, Y. Zhu, M. Yeh, K. Kamgar, K. Viswanathan, C. K. Gupta, and E. Keogh. The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance and Correlation Coefficient, 2015. URL https://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html. [p78]

A. Reiss and D. Stricker. Introducing a New Benchmarked Dataset for Activity Monitoring. In *2012 16th International Symposium on Wearable Computers*, pages 108–109. IEEE, jun 2012. ISBN 978-0-7695-4697-1. doi: https://doi.org/10.1109/ISWC.2012.13. [p80]

J. Shieh and E. Keogh. iSAX: indexing and mining terabyte sized time series. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, page 623, New York, New York, USA, 2008. ACM Press. ISBN 9781605581934. doi: https://doi.org/10.1145/1401890.1401966. [p76]

D. F. Silva, C.-C. M. Yeh, Y. Zhu, G. Batista, and E. Keogh. Fast Similarity Matrix Profile for Music Analysis and Exploration. *IEEE Transactions on Multimedia*, 14(8):1–1, 2018. ISSN 1520-9210. doi: https://doi.org/10.1109/TMM.2018.2849563. [p79]

UCR. UCR Matrix Profile Page, 2016. URL http://www.cs.ucr.edu/~eamonn/MatrixProfile.html. [p78, 83]

L. Wei, E. Keogh, and X. Xi. SAXually Explicit Images: Finding Unusual Shapes. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 711–720. IEEE, dec 2006. ISBN 0-7695-2701-7. doi: https://doi.org/10.1109/ICDM.2006.138. [p76]

C.-c. M. Yeh, N. Kavantzas, and E. Keogh. Matrix Profile VI : Meaningful Multidimensional Motif Discovery. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2017a. [p79, 83]

C. C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. Matrix profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 1317–1322, 2017b. ISSN 15504786. doi: https://doi.org/10.1109/ICDM.2016.89. [p76, 78, 82]

C. E. Yoon, O. OReilly, K. J. Bergen, and G. C. Beroza. Earthquake detection through computationally efficient similarity search. *Science Advances*, 1(11):e1501057–e1501057, dec 2015. ISSN 2375-2548. doi: https://doi.org/10.1126/sciadv.1501057. [p82]

Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-c. M. Yeh, and G. Funning. Matrix Profile II : Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. *Icdm*, 54(1):739–748, jan 2016. ISSN 0219-1377. doi: https://doi.org/10.1109/ICDM.2016.126. [p77, 78]

Y. Zhu, M. Imamura, D. Nikovski, and E. Keogh. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. *Knowledge and Information Systems*, pages 1–27, jun 2018a. ISSN 0219-1377. doi: https://doi.org/10.1007/s10115-018-1224-8. [p81]

Y. Zhu, C.-c. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh. Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds. In *2018 IEEE International Conference on Data Mining (ICDM)*, 2018b. [p83]

Z. Zimmerman, K. Kamgar, Y. Zhu, N. S. Senobari, B. Crites, and G. Funning. Scaling Time Series Motif Discovery with GPUs: Breaking the Quintillion Pairwise Comparisons a Day Barrier. *ACM*, 2018. doi: https://doi.org/10.1145/3357223.3362721. [p83]

Francisco Bischoff
CINTESIS - Center for Health Technology and Services Research
MEDCIDS - Community Medicine, Information and Health Decision Sciences Department
Faculty of Medicine of the University of Porto
Rua Dr. Placido Costa, s/n
4200-450 Porto, Portugal
ORCiD: 0000-0002-5301-8672
fbischoff@med.up.pt

Pedro Pereira Rodrigues
CINTESIS - Center for Health Technology and Services Research
MEDCIDS - Community Medicine, Information and Health Decision Sciences Department
Faculty of Medicine of the University of Porto
Rua Dr. Placido Costa, s/n
4200-450 Porto, Portugal
ORCiD: 0000-0001-7867-6682
pprodrigues@med.up.pt