Matrix Profile XVII: Indexing the Matrix Profile to Allow Arbitrary Range Queries

Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman and Eamonn Keogh
Department of Computer Science and Engineering
University of California, Riverside
{yzhu015, myeh003, zzimm001}@ucr.edu, eamonn@cs.ucr.edu

***Abstract*— Since its introduction several years ago, the Matrix Profile has received significant attention for two reasons. First, it is a very general representation, allowing for the discovery of time series motifs, discords, chains, joins, shapelets, segmentations etc. Secondly, it can be computed very efficiently, allowing for fast exact computation and ultra-fast approximate computation. For analysts that use the Matrix Profile frequently, its incremental computability means that they can perform ad-hoc analytics at any time, with almost no delay time. However, they can only issue *global* queries. That is, queries that consider all the data from time zero to the current time. This is a significant limitation, as they may be interested in localized questions about a contiguous subset of the data. For example, "*do we have any unusual motifs that correspond with that unusually cool summer two years ago*". Such ad-hoc queries would require recomputing the Matrix Profile for the time period in question. This is not an untenable computation, but it could not be done in interactive time. In this work we introduce a novel indexing framework that allows queries about arbitrary ranges to be answered in quasilinear time, allowing such queries to be interactive for the first time.**

Keywords*— *Time Series, Matrix Profile, Indexing

I. Introduction

The Matrix Profile is a data structure that annotates a time series [10]. For every subsequence in a time series, the Matrix Profile records the distance *to*, and the location *of*, that subsequence's nearest neighbor [10]. While it is a simple data structure, in a series of recent papers it has been shown that given *just* the Matrix Profile, one can trivially and quickly perform a host of high level time series data analytics, including motif discovery [10][11], discord discovery (anomalies) [10], chain discovery [13], various types of joins [10][11], semantic segmentation [3], etc.

The generality and utility of the Matrix Profile has led to a flurry of papers on computing it as efficiently as possible [10][11]. The current state-of-the-art algorithm, SCRIMP++ [12] is sufficiently fast enough to allow many "casual" users to interact with their data in real time. For example, in fields as diverse as geoscience [1] and entomology, users may "only" need to examine time series with a few hundred thousand data points. However, for some users, with time series with millions of data points, real-time interaction is untenable, and they must resort to precomputation. Consider the following example of a leading oil and gas company:

- A plant manager in Texas is in charge of a single massive distillation column. The column has been running the same "recipe" for the last eight years. Sampled once per minute, the 474 sensors have produced *tags* (time series) with about 4,200,000 data points each. While fast *approximate* algorithms exist for computing the Matrix Profile, because of worker safety issues [2], and the extraordinary costs of even one minute of down time, our plant manager insists on *exact* computation during any analytic operations.

Using a modern desktop, computing the Matrix Profile for a single tag of this length takes about nine hours; hours; on a GPU it takes just twenty-two minutes. Moreover, the Matrix Profile can be computed *incrementally* [10]. Updating each Matrix Profile takes less than $1/100^{th}$ of a second, and only needs to be done once a minute. This allows our diligent plant manager to have the up-to-date Matrix Profile precomputed whenever she needs to run some analytics.

This appears to be the end of the issue. But let us return to the analytical needs of our plant manager. She occasionally sets aside a morning to investigate whether there are any motifs that correlate with *foaming*, an extremely undesirable and expensive event that can reduce chemical separation efficiency and may even produce contaminated products that have to be discarded or reprocessed [7]. However, as she begins to investigate, she realizes she has additional ad-hoc constraints:

- While examining the `debutanizer_flow` tag, she realizes that she should exclude all data from summer 2017. At that time, due to a petrochemical shortage caused by a hurricane, they had switched to a different infeed ingredient, which made that time period non-commensurate with the rest of the data.
- While examining the `volumetric_flow_17` tag, she realizes that she should exclude data from the first six months of the eight-year history. During that time range[1], they used a chemical foaming inhibitor [7], which was subsequently outlawed.

Here, we are finally defeated. While we can compute and incrementally maintain a Matrix Profile, we cannot "slice and dice" it to allow arbitrary exclusions of regions. While we have used an industrial example for clarity, nearly identical motivating remarks can be made for domains as diverse as seismology, medicine and entomology (Fig. 1).

The data shown in Fig. 1 motivates our work on a different time scale. Entomologists at the Mauck Lab record up to a dozen eight-hour sessions of Whitefly behavior each day, then compute the Matrix Profile overnight and examine the data for

[1] The world *range* has an unfortunate overload in this community. It often means the distance from some protype, as in range *query*. However, we use it in the sense of a *time* range, a week or a year etc.

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00185

motifs (and discords [10]) the next morning [6]. In the example shown in Fig. 1, there is a very low amplitude (but not *constant*) region in the first hour of data, and it causes most of the top motifs and discords to be found in that region ([4] explains *why*). The entomologists can guess the reason for this region; the insect may have exhausted a vein as a food source, and spent about twenty minutes looking for another one. They could delete the offending data, and recompute the Matrix Profile, but that would take on the order of an hour of CPU time. However, they really need to see the motifs *now*, in order to plan the next set of experiments. Like our oil and gas engineer, the entomologists need to be able to exclude arbitrary regions of the data in an ad-hoc manner, and examine the resulting matrix profile in interactive time.

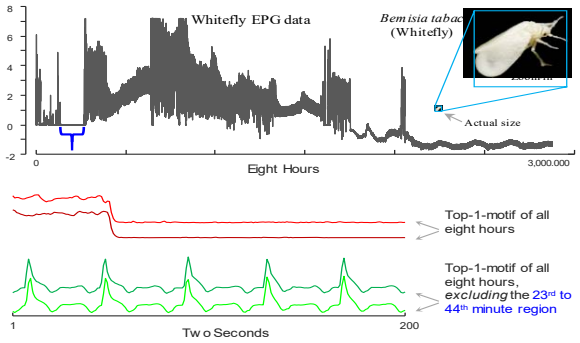

Fig. 1. *top*) Eight hours of data collected from a species of insect that causes billions of dollars of damage each year [9]. *bottom*) The Top-1-motif from all the data is an uninteresting pattern, probably caused by poor contact. Excluding the low amplitude region (blue bracket) during motif search produces a more interesting motif (probably *xylem-ingestion* behavior).

In this work we introduce STUMP, Scalable Time series Ubication Matrix Profile. The goal of STUMP is to produce a "meta" Matrix Profile which can be precomputed (and incrementality maintained), and then be used to allow quickly produce both:

- a standard Matrix Profile corresponding to an arbitrary region of a time series. This supports queries such as: *Find motifs in Google query volume for 'iPhone', but only look at the first eight months of 2016.*
- a standard Matrix Profile corresponding to a time series with an arbitrary region excluded. This supports queries such as: *Find motifs in Google query volume for 'Yahoo', but exclude all of Marissa Mayer's tenure.*

We will demonstrate that in order to support arbitrary range constraints, the memory and time overhead needed are just O($n$log(n)) and O($n$log(log(n)), respectively, which is inconsequential for the motivating domains outlined above.

II. PROBLEM DEFINITION

We follow the definition of the time series, the time series subsequence, the distance profile, the matrix profile and the matrix profile index in [11]. Our task at hand can be formally defined as follows.

Problem Definition: Given a time series $T = t_1, t_2, ..., t_n$ and a user supplied subsequence length m, compute a data structure X, that will allow the efficient creation of the following four types of matrix profiles (and indices) using any range (s, e):

- The *inclusion matrix profile* $P^{(s,e)}$ and *matrix profile index* $I^{(s,e)}$ corresponding to $T^{(s,e)}$. Here $T^{(s,e)}$ represents a section of T, starting from s and ending at e: $T^{(s,e)} = t_s, t_{s+1}, ..., t_{e-1}, t_e$. That is, for every subsequence of length m in $T^{(s,e)}$, we would like to efficiently find its nearest neighbor within $T^{(s,e)}$.
- The *exclusion matrix profile* $P^{\wedge(s,e)}$ and *matrix profile index* $I^{\wedge(s,e)}$ corresponding to $T^{\wedge(s,e)}$. Here $T^{\wedge(s,e)}$ represents the remaining parts of time series T after $T^{(s,e)}$ is excluded: $T^{\wedge(s,e)} = t_1, t_2, ..., t_{s-1}, t_{e+1}, t_{e+2}, ..., t_n$. That is, for every subsequence of length m in $T^{\wedge(s,e)}$ (this means every subsequence in $T^{(1,s-1)} = t_1, t_2, ..., t_{s-1}$ and in $T^{(e+1,n)} = t_{e+1}, t_{e+2}, ..., t_n$), we would like to efficiently find its nearest neighbor within $T^{\wedge(s,e)}$.

The problem has a simple brute-force solution. $P^{(s,e)}$ can be obtained by running the STOMP algorithm [11] directly on $T^{(s,e)}$, and $P^{\wedge(s,e)}$ can be obtained by ignoring the corresponding rows and columns of the distance matrix [11]. The solution has a low space complexity of $O(n)$, but the time complexity can be as high as $O(n^2)$. This is very undesirable, as the user may need to adjust the query range (i.e., the values of s and e) multiple times if she is not satisfied with the result. Can we do better?

III. THE STUMP ALGORITHM

The guiding principle of the STUMP algorithm is to use as little space as possible to store necessary information to allow fast arbitrary range queries of the matrix profile. To see how it works, let us first consider a toy example.

Assume that we have a subsequence $T_{i,m}$ (the subsequence of length m starting from position i) of time series T, and the distance profile D_i corresponding to $T_{i,m}$ is as follows:

1, 5, 4, 2, 3, 6, 7, 5, 1, 0, 2, 4, 2, 8, 3, 5, 9, 2, 3, 4

Assume i=10, m=4. To avoid trivial matches [10], we set the values within the exclusion zone of D_i to infinity:

1, 5, 4, 2, 3, 6, 7, 5, *inf*, *inf*, *inf*, 4, 2, 8, 3, 5, 9, 2, 3, 4

The nearest neighbor of $T_{i,m}$ within the *whole* time series T corresponds to the minimum value of D_i: the value 1 located at 1. This information is stored in the ith entry of the matrix profile and the matrix profile index: P_i=1 and I_i=1.

Now let us first consider arbitrary inclusion range queries. Assume that the user would like to obtain the inclusion matrix profile $P^{(s,e)}$, where s=7 and e=20. To find the nearest neighbor of $T_{i,m}$ within $T^{(7,20)}$, we need to look for the minimum value from entries 7 to 17 of D_i (the last entry is 17 instead of 20 since 20 is the last data point of $T_{17,4}$):

1, 5, 4, 2, 3, 6, [7, 5, *inf*, *inf*, *inf*, 4, 2, 8, 3, 5, 9], 2, 3, 4

We can see that the new minimum value is 2, located at 13. The brute-force algorithm would take an $O(n)$ time to find this minimum value; our algorithm can reduce this to O(log(log(n)).

As shown in Fig. 2, we divide D_i into two parts: the left part before location i (shown in Fig. 2.*middle*) and the right part after location i (shown in Fig. 2.*bottom*). We scan through the left part of D_i from location i backward (as shown in Fig. 2.*middle*), using

a vector SL_i to track the minimum-so-far values and a companion vector SLI_i to track their locations. Similarly, we use a pair of vectors SR_i and SRI_i to track the minimum-so-far updates through the right part of D_i from location i forward (as shown in Fig. 2.*bottom*).

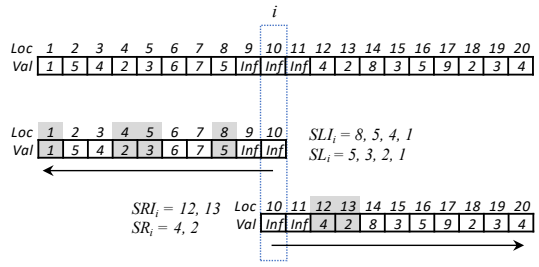

Fig. 2. *top*) The distance profile D_i in the toy example. *middle*) From location i backward, the minimum-so-far values in D_i are stored in SL_i and their locations in SLI_i. *middle*) From location i forward, the minimum-so-far values in D_i are stored SR_i and their locations in SRI_i.

Our claim is that that SL_i, SLI_i, SR_i and SRI_i provide us sufficient information to locate the left and right nearest neighbors (and thus also the general nearest neighbor) of $T_{i,m}$ for any arbitrary range inclusion matrix profile query. Consider our toy example, where we have the starting point as s=7 and the end point as e=20 (corresponding to locations 7 to 17 of D_i). The left nearest neighbor of $T_{i,m}$ corresponds to the last minimum-so-far update (located at 8 with a distance value of 5) before we reach the starting location 7. *Similarly*, the right nearest neighbor of $T_{i,m}$ corresponds to the last minimum-so-far update (located at 13 with a distance value of 2) before we reach the end location 17. Since SLI_i and SRI_i are both monotonic, we can easily locate these updates with binary search.

Next, let us consider the exclusion matrix profile queries. Assume that the user now sets the starting point as s=13 and the end point as e=17. The nearest neighbor of $T_{i,m}$ within $T^{\sim(13,17)}$ now corresponds to the minimum value of D_i, excluding entries 13 to 17:

$$[1, 5, 4, 2, 3, 6, 7, 5, \mathit{inf}, \mathit{inf}, \mathit{inf}, 4], 2, 8, 3, 5, 9, [2, 3, 4]$$

We can see the minimum value becomes 1, located at 1.

Different from the inclusion matrix profile case (Fig. 2), in which we scan through D_i from location i to both ends, this time we scan D_i from its left end through its right end, and then from the right end through the left end. As shown in Fig. 3.*top*, we use a pair of vectors LR_i and LRI_i to track the minimum-so-far value updates from left to right. Similarly, the right-to-left updates are stored in RL_i and RLI_i. As both LR_i and RL_i are monotonically decreasing, we can again use binary search to locate the left and right nearest neighbors of a subsequence given any exclusion range queries.

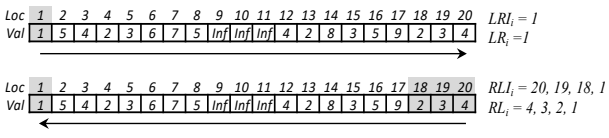

Fig. 3. *top) LR_i* and LRI_i store the minimum-so-far value updates and their locations as we scan through D_i from left to right. *bottom) RL_i* and RLI_i store the minimum-so-far value updates and their locations as we scan through D_i from right to left.

What is the time and space complexity of this approach? Actually, the expected length of SL_i, SLI_i, SR_i, SRI_i, LR_i LRI_i, RL_i and RLI_i are all $O(\log(n))$ (we refer interested reader to [14] for a detailed derivation). As we need to store n-m+1 such vectors in memory, the overall expected space complexity of STUMP is $O(n\log(n))$. As such, each binary search operation takes $O(\log(\log(n)))$ time, and the overall time complexity of each arbitrary-range Matrix Profile query is $O(n\log(\log(n)))$.

IV. Empirical Evaluation

To ensure that our experiments are reproducible, we have built a website [14] to archive all the dataset and code. Due to space limitations, we refer interested readers to [14] for a detailed speed analysis of the STUMP algorithm on various datasets. Once the minimum-so-far vectors are computed, we can obtain any arbitrary range Matrix Profile of a million-scale time series within less than a second [14]. This interactivity enables useful arbitrary range primitives and regularities to be discovered.

A. Case Study: Melbourne pedestrian counting data

One of the major applications of the Matrix Profile is discord discovery [10]. Here we consider the sensor data of the pedestrian counting system [5] developed by the City of Melbourne. Fig. 4.*top* shows a three-year snippet (Jan 1st 2015 to Jan 1st 2018) of the data recorded at the Collins Place (North) in the central business district.

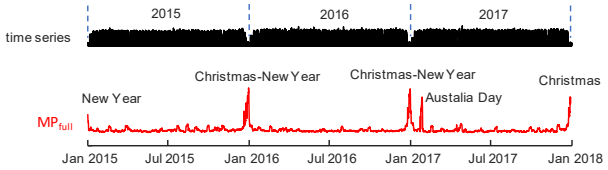

Fig. 4. Melbourne pedestrian counting data at the Collins Place (North) from Jan 1st, 2015 to Jan 1st, 2018.

The data is logged hourly, and we used a subsequence length of 120 (i.e., five days) to compute the Matrix Profile. As shown in Fig. 4.*bottom*, we can see five major peaks in the Matrix Profile, corresponding to five discord patterns through the three-year period. The 1st, 2nd, 3rd, and 5th peak from left to right indicate the unusual pedestrian traffic patterns at Christmas and New Year's Day of each year. The 4th peak indicates the Australia day in 2017: that is the only public holiday on a Thursday throughout this three-year period.

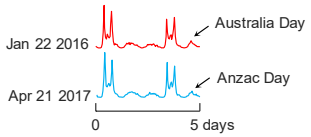

Fig. 5. The five-day subsequence ending at the Australia Day, Tuesday Jan 26th (top) is very similar to the five-day subsequence ending at the Anzac Day, Tuesday April 25th, 2017 (bottom). As a result, the Matrix Profile values in Fig. 4.*bottom* are low around the two holidays.

Though the Matrix Profile in Fig. 4.*bottom* successfully captured some unusual holiday traffic patterns, it apparently missed a lot of other important holidays and events. For

example, it failed to capture the unusual traffic pattern around the Australia Day in 2016. Fig. 5 shows the reason: the five-day subsequence starting at Jan 22th, 2016 and ending at the Australia Day is very similar to the subsequence starting at April 21st, 2017 and ending at the Anzac Day; both holidays are on Tuesday. As a result, the Matrix Profile in Fig. 4.*bottom* shows low values around both holidays, though we can rarely see such a pattern in a normal week.

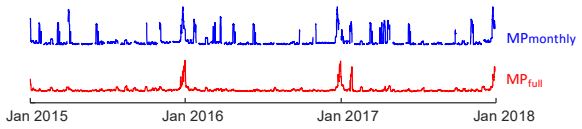

Fig. 6. The monthly Matrix Profiles (top) contains many more peaks than the full Matrix Profile (bottom).

One way to find such discords is through computing *monthly* Matrix Profiles. That is, instead of computing the full Matrix Profile, we divide the data into month-long sections and repeatedly compute the inclusion Matrix Profiles corresponding to each month. This seems to be very time-consuming, but with STUMP we can find *all* the monthly Matrix Profiles within less than a second. As shown in Fig. 6, we can see many more peaks in $MP_{monthly}$ (the combination of all the monthly Matrix Profiles) than in the full Matrix Profile (MP_{full}).

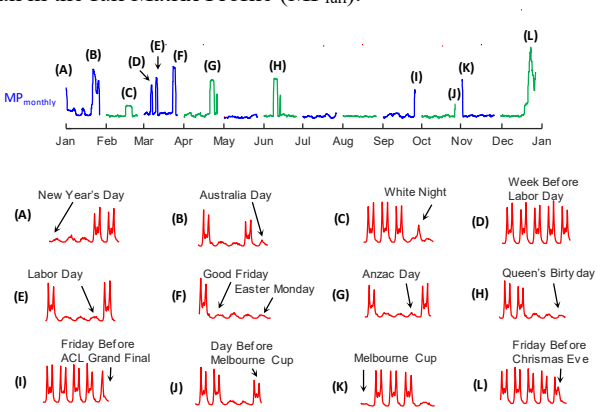

Fig. 7. The monthly Matrix Profiles (top) captured all the important holidays and events of Melbourne (bottom) in 2016.

Fig. 7.*top* provides a closer look at the monthly Matrix Profiles in 2016. There are about twelve major peaks in all these Matrix Profiles; their corresponding subsequences can be found in Fig. 7.*bottom*. We can see that all the important holidays and events of Melbourne in 2016 are captured as discords (corresponding to peaks) in the monthly Matrix Profiles. Every subsequence in Fig. 7.*bottom* is very different from other subsequences in the same month. For example, the subsequence in Fig. 7.*bottom.C* indicates more-than-usual pedestrian traffic on Saturday, Feb 20th 2016 due to White Night, a music and art festival held at the city center. We cannot discover such a pattern in the full Matrix Profile, as this is an annual event, but we can find it in the Matrix Profile of February 2016. Fig. 7.*bottom.D* shows an interesting pattern of five consecutive weekdays, which are supposed to be normal. However, note that these are the only five consecutive weekdays in March, as there are a lot of public holidays in the same month including Labor Day, Good Friday and Easter Monday. Though this is a usual pattern in general, it becomes a discord in that specific month.

This example shows that the discords discovered in a dataset are strongly dependent on the query range specified. With the proposed STUMP algorithm, the user can interactively compare the Matrix Profiles corresponding to various ranges and analyze the motifs/discords accordingly. More case studies on motif discovery and discord discovery with STUMP can be found on our project website [14]. This "what-if" type of interaction reflects our observations of how data analysts, including petrochemical engineers, entomologists and clinicians actually interact with large data archives.

V. Conclusions

We have introduced an algorithm that allow us to compute the Matrix Profile under a variety of temporal constraints. Moreover, we have shown that these constraints map to real-world needs in domains as diverse as science and industry. We have made all code and data freely availed to allow others to confirm and build on our results. In future work we plan to exploit the ideas in [3][13] and elsewhere to create a system that can interactively suggest regions to include/exclude for various analytic tasks.

References

[1] Barbosa, S. Exploiting the similarity of soil moisture time series based on the matrix profile. Time Series Analysis in the Geosciences - Concepts, Methods and Applications, EGU General Assembly 2018.

[2] Crowl, D.A., and Louvar, J.F., Chemical Process Safety: Fundamentals with Applications, Prentice Hall, New York (1999), pp 471–508.

[3] Gharghabi, S., et. al. Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels. IEEE ICDM 2017: 117-126.

[4] Dau, H.A., and Keogh, E. Matrix Profile V: A Generic Technique to Incorporate Domain Knowledge into Motif Discovery. KDD 2017: 125-134.

[5] Melbourne Pedestrian Counting System. URL: http://www.pedestrian.melbourne.vic.gov.au Accessed 06/01/2019.

[6] Mauck, K.E., Chesnais, Q., Shapiro, L.R. Evolutionary Determinants of Host and Vector Manipulation by Plant Viruses. Advances in Virus Research 2018.

[7] Qin, S., Hansen, B.B., Kiil, S. Effects of foaming and antifoaming agents on the performance of a wet flue gas desulfurization pilot plant AICHE J., 60 (2014): 2382-2388.

[8] Silver, N. The signal and the noise: Why so many predictions fail-but some don't. New York, NY: Penguin, 2015.

[9] Waldron, P. (2016) Why the whitefly is such a formidable threat to food security. URL: phys.org/news/2016-12-whitefly-formidable-threat-food.html

[10] Yeh, C.C.M., et. al. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. IEEE ICDM 2016: 1317-1322.

[11] Zhu, Y., et. al. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016: 739-748.

[12] Zhu, Y., et. al. Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds. IEEE ICDM 2018 (to appear).

[13] Zhu, Y., Imamura, M., Nikovski, D. and Keogh, E. Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining. IEEE ICDM 2017: 695-704.

[14] Project Website: https://sites.google.com/site/arbitraryrangemp/