Matrix Profile XXXI: Motif-Only Matrix Profile: Orders of Magnitude Faster

Maryam Shahcheraghi[1] and Eamonn Keogh

This is an informal author's copy of the final canonical Matrix Profile paper. I, Eamonn Keogh, am using this opportunity to offer some reflections and general comments. If you just want to read the "MOMP" paper, skip to page 3.

Dear Reader. This is a final canonical[2] Matrix Profile (MP) paper. The MP papers [15][16][17][18][19][20][21] [22][23][24][25][26][27][28][29][30][31][32][33][34][35][36][37][38][39][40][41][42][43][44][45] etc. are a sequence of papers in which I outlined my vision[3] for time series data mining. That vision is:

- Essentially all time series data mining problems can be solved by reasoning about the similarities (or lack thereof) between time series subsequences.
- The Distance Profile and the Matrix Profile are the fastest and most space efficient way to compute the needed similarities.
- Once you gain some intuitions for the MP, you can use them as higher-level abstractions to *think* about how to solve problems[4].

There are still some gaps remaining in this vision, however I will let the community fill these in (assuming they consider this a good use of their time).

In this preface to the final canonical Matrix Profile (MP) paper, I will list some open problems that the community may be interested in addressing and thank individuals that helped me realize this vison.

There is a famous debate in mathematical philosophy. Is mathematics an *invention* (a creation of the human mind) or a *discovery* (something that exists independently of us)? I am not qualified to speak on this, but in doing research on the Matrix Profile, I always felt like I was discovering *natural* beautiful existing structures, and my only task was to write them down and illustrate them with interesting examples. It was the most rewarding, fun and intellectually satisfying thing I have ever done. Dear Reader, whatever research you do, I hope you enjoy it as much as I have enjoyed creating this series of papers. Salutations!

Open Problems for the Matrix Profile

I) The MP, including minor variants such as the MP-join and the left-MP, can be used to create semantically meaningful primitives, including:

- Time Series Motifs
- Time Series Discords
- Time Series Chains
- Time Series Shapelets
- Time Series Consensus Motifs
- Time Series Joins
- Time Series Snippets
- Time Series Semantic Motifs
- Time Series Mplots
- Time Series Platos
- Time Series Novelets

Are there additional primitives waiting to be discovered?

II) Most MP algorithms only look at the subsequence *distances*, i.e. the MP itself. However, some MP algorithms, such as the FLOSS algorithm consider the *topological information* in the Matrix Profile Index [25]. Are the additional tasks for which the topological information in the Matrix Profile Index is useful?

III) Given the MP, and possibly a few additional calls to the Distance Profile, it is possible to compute any meaningful motif definition that currently exists or *could* exist, including top-K motifs, range motifs, motif sets etc. The "best" definition depends on the data, and the downstream use of motif discovery. Thus, I do not believe that there is any meaningful way to compare or rank motif *definitions* (except perhaps in very limited cases for specialized applications, for example, the best motif definition *just* for Carnatic Music [1]). Given this observation, I do not believe that is meaningful to compare motif *discovery* (really, motif *extraction*) algorithms.

However, there is clearly a need to establish benchmarks for the *speed* of motif discovery algorithms. This is complicated by two facts:

[1] There are additional authors, however I want to thank them here in the third person and offer some comments that they are not responsible for.
[2] By *canonical*, I am not including journal extensions of papers, or specialist uses of the MP, for music [46] or seismology [47] etc.
[3] Yes, "*my vision*" does sound like a pretentious and self-aggrandizing rhodomontade, but this is always how I have seen this.
[4] This is best exemplified in [48], and in Ryan Mercer's amazing papers [38][40].

- The speed of some motif discovery algorithms depends on the data. Thus, claims must be tested on a variety of datasets (or limited to a single datatype, for example "*a fast algorithm for motif discovery in classical music*").
- As noted in *this* paper and in [27], for large datasets, we almost always should use the *anytime* property of motif discovery.

Thus, I propose that a plot like Figure A is the best way to evaluate motif discovery algorithms (see also Figure 14).

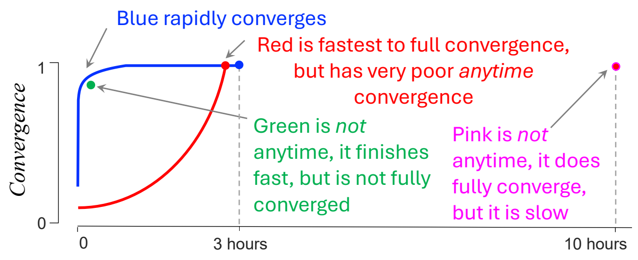


Figure A: Four hypothetical algorithms compared on a single dataset.

In this example we can completely dismiss the Pink algorithm. The Green algorithm is batch-only and fast. However, notice that if we had stopped the Blue algorithm at the amount of time that Green took, the Blue algorithm is closer to full convergence. Thus, we can dismiss the Green algorithm too.

The Red algorithm is the fastest to perfect convergence, but only a little bit better than the Blue algorithm, which has Hyper-anytime properties [45].

In my mind, a plot like this is the perfect way to evaluate motif discovery algorithms. There are only two issues to address.

- The Y-axis is *convergence*, but converge to what? I recommend convergence to the top-1 motif pair, if you can compute this fast, then any other definition of motifs can also be computed fast.
- Which dataset(s) should be used? It is very clear that different datasets favor different approaches. For example, Attimo [3] looks like the Green algorithm on some datasets, but like the Pink algorithm on others (See [10]). In this paper we consider thirty large diverse datasets. I am not suggesting that these datasets become the communities benchmark (people may reasonably argue that they are biased to MOMP), however some set of interested parties could clearly build a collection of say one hundred similarly ambitious datasets, to establish a community benchmark[5]. I would love to see this happen.

IV) I have done motif discovery for dozens of "customers"; both paying companies and free consulting for medical doctors, entomologists, seismologists, ornithologists, marine scientists, astronomers, roboticists, oil&gas engineers, neuroscientists etc. In almost every case, I exploit the anytime property of SCRIMP++/SCAMP/MOMP [27].

I believe that for almost any practical scientific or industrial application, we can and should use this anytime property. The MP algorithms are "superstars" among anytime algorithms; SCRIMP++ is (at least) *Super-Anytime*, and MOMP is *Hyper-Anytime* [45]. In almost every case, within 1% of the time needed for full convergence you can discover what is almost certainly the final motif or a very minor variant of it (see Figure 14).

However, it is not clear that the community understands this. Is what is needed simply some more education/advocacy for anytime algorithms, or do we need a formal framework for doing anytime motif discovery? I do have some ideas as to what the latter would look like, but I will deliberately avoid influencing anyone who may be interested in exploring this.

V) The highest values in the MP are *time series discords* [39][45]. In many (but by no means all) situations, time series discords are excellent anomaly detectors. In fact, there are dozens of papers that use "time series discord" as a synonym for "anomaly". I might have made more of an effort to correct this, if it was not so flattering.

Time series anomaly detection is a real black eye for the data mining community. In *just* the mainstream conferences (VLDB, SIGKDD etc.) have been hundreds of papers on the topic in the last decade, but there has been essentially zero progress. I have pointed this out in [49] and [50], so I will not repeat my arguments here. It would be wonderful to see *solid* work on creating benchmarks and scoring functions.

VI) This paper introduces the first lower bound for the Matrix Profile. Because it is the first, we have optimized our work for clarity of presentation. Two questions may have occurred to the reader:

- Is there an algorithm better than MOMP, to take advantage of the lower bounds? I think the answer is *no*.
- Are there better lower bounds than lbMP? Here I think the answer is *yes*.

While most papers claiming to do "fast motif discovery" have dubious utility, here there is a very simple and concrete test. Consider Figure 7, the x-axis can be expressed as wall-clock time instead. Then, given any proposed lower bound we could measure its TLB and the time it takes to compute it. If that

[5] Goodhart's Law states that "*when a measure becomes target, it ceases to be a good measure*". There are dangers in creating benchmarks, the UCR Classification Archive has been a mixed blessing to say the least. However, there are recent papers making incredible claims about the speed of motif discovery that only test on *tiny* datasets. A benchmark would act as some filter /barrier-to-entry for such poor ideas.

datapoint is within the Pareto frontier (the yellow region) it is worthless, if it is above the yellow region, it can be helpful. As noted in **III**, and in this paper, MOMP (unlike "pure" MP algorithms) is data dependent, so one would need to test over many datasets. I regard the task of finding tighter lower bounds as one of the most interesting open problems in time series motif discovery.

VII) The Matrix Profile has been used to solve at least two hundred real-world problems[6]. For example, [57] uses motifs for the characterization of Dansgaard-Oeschger events in paleoclimate time series. In [58] motifs are used for bridge health monitoring. Paper [59] uses motifs to classify motions in quadruped robots. The detection of major depressive disorders is facilitated by motifs in [60], in [62] the authors use motifs to study meiotic CMs in fission yeast. As I write this, I just got an email from a student whose PhD project is using the Matrix Profile to characterize Marsquakes! (Yes, "earthquakes" recorded on Mars).

However, I believe that there are thousands of researchers that could use the Matrix Profile in their work, but they are not aware of it. How can we make the broadest scientific community aware of the utility of the Matrix Profile?

VIII) The multiresolution computation of the Matrix Profile used by MOMP can be exploited by other algorithms for other tasks. In this work we show just one example; iMOMP for exploring *sets* of time series. Are there other uses for MOMP's multiresolution computation?

Acknowledging those that helped create the Matrix Profile

The Matrix Profile was created by a team of *incredible* students (all now graduated) and a handful of other collaborators from industry and academia. In the below, I will refer to these former students by their *first* name, as we always enjoyed this mutual level of informality.

The original MP was inspired by **Mueen**'s Distance Profile (MASS) and created by **Yan** and **Micheal**. My master's student **Kaveh** helped to optimize the algorithms and **Zach** optimized the code. **Anh** showed how to "nudge" the MP to find the domain/task specific motifs.

By coincidence, five compatriot females with the same first initial exploited the maturing MP algorithms to provide significant extensions. **Shaghayegh** exploited the MP to produce a superhuman semantic segmentation algorithm and a robust distance measure and **Shima** created two novel primitives that used the MP. **Sara** showed how to compute the MP under DTW and **Sadaf** showed how to compute the MP under Catch-22. In two different ways (including *this* paper), **(Seyedeh)Maryam** showed how the MP can be used with multiresolution data.

The late **Frank** selfishly made multiple minor contributions to others work on the MP and introduced the beautiful PAN-Matrix Profile. **Ryan**'s work perhaps best exemplified the spirit of the MP; that with just the MP and a handful of lines of additional code, you can do amazing things. **Audrey** created a distance measure for comparing *long* time series. **Yue** used the MP to create *extraordinary* anomaly detectors, parameter-free, ultra-fast and ultra scalable. **Renjie** and **Akhil** helped several MP first authors solve important subproblems.

Diego, a visiting researcher from Brazil, made several contributions, including showing how the MP could be applied to music (he is the real inventor of the wonderful Queen|Vanilla-Ice example [61]).

Daniel Nikovski, **Makoto** Imamura and **Takaaki** Nakamura of Mitsubishi Labs provided useful ideas, applications and problems.

UCR professors **Gareth** Funning, **Evangelos** Papalexakis, **Samet** Oymak and **Philip** Brisk provided advice and UCR entomologists/professors **Quentin** Chesnais, **Kerry** Mauck and **Amy** Murillo provided amazing datasets and interpretation of findings. **Anthony** Bagnall helped with evaluation on (downstream to MP) classification algorithms.

It is almost impossible to do research in the USA without funding. Many thanks to the **NSF**, **Toyota**, **Oracle**, **VISA Research**, **Mitsubishi**, **Amazon** and **Google**. It is impossible to do research without *data*, many thanks to all the donors of datasets, who are too numerous to list individually.

Sean Law has always been a big "booster" for the Matrix Profile, as have **Francisco** Bischoff, **Frankie** Cancino, **Tyler** Marrs and **Andrew** Van Benschoten.

The University of California Riverside is an amazing place to do research, thanks to all staff, administrators and faculty.

Finally, thanks to all the editors, program chairs and reviewers, even though sometimes their questioning felt like the Spanish Inquisition[7]. Peer review is painful, and it can be very discouraging for young researchers. I know of no magic bullet, but if you do good work and your ideas will eventually find a home.

STOP PRESS: Just before we released this paper, I discovered that the Matrix Profile will be an official primitive in Matlab starting with version R2024b [63]. I was not involved in this, and it is too early to speak of the quality of the implementation. However, I am delighted that another step has been taken to make the Matrix Profile even more accessible to the broadest possible community.

[6] Scroll to the bottom of www.cs.ucr.edu/~eamonn/MatrixProfile.html

[7] Nobody expects the Spanish Inquisition!

Matrix Profile XXXI: Motif-Only Matrix Profile: Orders of Magnitude Faster

Maryam Shahcheraghi, Chin-Chia Michael Yeh, Yan Zheng, Junpeng Wang, Zhongfang Zhuang, Liang Wang, Mahashweta Das, and Eamonn Keogh

***Abstract*—Approximately repeated subsequences in a longer time series, i.e., *time series motifs*, are important primitive in time series data mining. Motifs are used in dozens of downstream tasks, including classification, clustering, summarization, rule discovery, segmentation etc. Time series motif discovery is a notoriously computationally expensive task. Some motif discovery algorithms are fast in the *best* case, but in other datasets, even if both the data and motif lengths are held the same, both their time and space complexity can explode. The Matrix Profile has the nice property that its time and space complexity are independent of the data. Moreover, the Matrix Profile is fast enough for datasets with about one million datapoints, which covers a large fraction of user cases. However, there are situations where we may wish to consider datasets which are much larger. In this work, we introduce the first lower bound for the Matrix Profile and an algorithm that exploits that lower bound to allow orders of magnitude speed up for *exact* motif search on real-world datasets. We demonstrate the utility of our ideas with the largest and most ambitious motif discovery experiments ever attempted.

Keywords—Time Series; Motifs

I. Introduction

Much of science reduces to finding conserved structure, then reasoning about the mechanism that causes the conservation. Approximately conserved patterns in time series are called *time series motifs* [4][15]. Figure 1 shows a motif discovered in an EOG (eye movement) dataset.

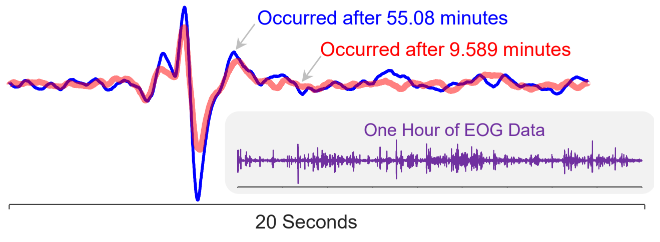


Figure 1: The Top-1 twenty-second long motif found in an EOG dataset. *inset*) The full EOG (sc410e) dataset (cf. Table 7).

Broadly speaking, there are two main approaches to finding time series motifs. The first is to attempt to find a good *best-so-far* motif quickly by using hashing [3][4] or searching reduced dimensionally data [9], etc. Then, exploit this *best-so-far* to prune off some calculations with a handful of classic techniques such as lower bounding, triangular inequality or early abandoning. These approaches can work well when the data is "cooperative." However, in other cases the data can force these algorithms to explode in memory and/or time requirements. For example, Quick-Motif [9] works well if there is a distinct motif of say length 250 in a reasonably smooth dataset. But, if the motif is less than strongly conserved, or the motif length grows beyond ~1,000, or the data is noisy, the algorithm crawls to a speed slower than brute force search. Worse, the memory footprint grows dramatically. Even on a 32 Gb machine, a noisy dataset with a mere one million datapoints will force Quick-Motif to give an `out-of-memory` error.

The second main approach to finding time series motifs is the Matrix Profile (MP) and the various algorithms to compute it, including STAMP, STOMP, SCAMP and SCRIMP [15][17][18]. All these MP algorithms have a time and space complexity that is independent of the structure/noise-level of data. While the computation of MP is quadratic in the length of the time series, it has two properties that make it the best algorithm for motif discovery in most cases. First, the computation of the MP is O(1) in the motif length m. Note that the user-specified m can be a 1,000 or greater. If the length of the data n, is in the low millions, then $O(n^2)$ will be faster than a say $O(n \times \text{sqrt}(n) \times m)$ algorithm. This is especially true as the constant factors for the MP algorithms have been optimized to be incredibly low [15][17][18].

There is another reason why the MP is typically preferred. Some of the state-of-the-art MP algorithms, including SCRIMP, are *anytime* algorithms. In practical applications, a user may be satisfied with the tentative results produced in the first two to three percent of the algorithm's run.

To summarize, we have satisfactory motif discovery algorithms for datasets whose lengths are in the low millions, but beyond that, motif discovery rapidly becomes impractical. Several research groups have pushed this limit by exploiting high performance computing paradigms [12][17], however not everyone has access to such hardware.

In this work, we show that we can combine the two paradigms: the stochastic "*try to find the motif early then prune*" with the MP's "*optimize the computation independent of the data*." We do this by defining a parametrizable lower bound for the MP, and creating an algorithm that exploits it to dramatically accelerate motif discovery. As we will show, this allows us to consider datasets that are two orders of magnitude larger than anything that has been attempted before. For some of the more challenging datasets we consider, a direct use of the SOTA MP algorithms would take decades to finish.

The rest of this paper is organized as follows. In Section II we offer some geometric intuitions for the concepts behind our lower bound. Section III introduces our definitions and notations, allowing us to introduce our lower bound in Section IV. In Section V, we introduce a multiresolution search algorithm that exploits this lower bound for motif discovery. In Section VI, we conduct the most ambitious experimental evaluation of motif discovery ever attempted, before offering brief conclusions in Section VII.

II. Geometric Intuitions

Before introducing our definitions and notation in the next section, we will take a moment to visually review the ideas behind lower bounding, the triangular inequality and the combination of these two techniques. While the triangular inequality is a commonly used tool for indexing and similarity search, our work is unusual in that it exploits triangular inequality *twice*. Hence this review may help sharpen the readers' intuition for the contributions in Section IV.

In Figure 2.*left*, we show two points, R_1 and B_1 that are 9.0 units apart in Euclidean space.

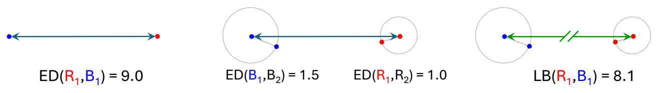


Figure 2: *left*) We know points R_1 and B_1 are 9.0 units apart. *center*) We further know that B_2 is 1.5 units from B_1, and that R_2 is 1.0 units from R_1. *right*) Here we assume we do not know the true distance between R_1 and B_1 but we know a *lower bound* for it, 8.1.

Further assume that we know B_2 is 1.5 units from B_1 and that R_2 is 1.0 units from R_1, as is illustrated in Figure 2.*center*. From this we can derive a new fact: a lower bound of the distance between B2 and R2 is 6.5. This we computed as a **L**ower **B**ound with *two* applications of **T**riangular **I**nequality:

$$\text{LBTI}(B_2,R_2) = \text{ED}(R_1,B_1) - [\text{ED}(R_1,R_2) + \text{ED}(B_1,B_2)] = 6.5$$

Note that while the LBTI here is positive, if the "circles" shown in Figure 2.*center* were relatively large, then this function could be negative, which we "snap" to zero.

We can generalize this idea to the case where we do not know the *exact* distance $\text{ED}(R_1,B_1)$ but as shown in Figure 2.*right*, we only have some lower bound for it, i.e. $\text{LB}(R_1,B_1) \leq \text{ED}(R_1,B_1)$. We can still compute a (now *weaker*) lower bound between B_2 and R_2 by changing a term in LBTI.

$$\text{LBTI}(B_2,R_2) = \text{LB}(R_1,B_1) - [\text{ED}(R_1,R_2) + \text{ED}(B_1,B_2)] = 5.6$$

To make it clear that how we can exploit this information imagine that we know "for free" the information in Table 1.

Table 1: Incomplete information about pairwise distances between objects in R & B

	R_1	R2	B_1	B2
R_1		1.0	?	?
R2			?	?
B_1				1.5
B2	$\text{LB}(R_1,B_1) = 8.1$			

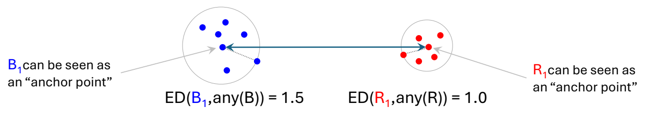
Further imagine we are tasked finding the *closest pair* between *any* R point and *any* B point. Naively, we could compute the four missing values in Table 1 and select the minimum. Instead, we leverage the intuitions presented in Figure 2 to find the answer while doing less work.

Suppose we randomly choose one of the four missing values to compute, say $\text{ED}(R_2,B_2)$ and discover it is 6.5, establishing a *best-so-far* (*bsf*). In this case we can see that:

- $\{R_1,B_1\}$ cannot be a closer pair because we have:
  $\text{ED}(R_1,B_1) \geq \text{LB}(R_1,B_1)$ which is 8.1
  Since 8.1 is greater than our *bsf*, $\{R_1,B_1\}$ is pruned.
- $\{R_1,B_2\}$ cannot be a closer pair because we have:
  $\text{ED}(R_1,B_2) \geq \text{LB}(R_1,B_1) - \text{ED}(B_1,B_2)$, which is 8.1 – 1.5
  Since 6.6 is greater than our *bsf*, $\{R_1,B_2\}$ is pruned.
- $\{R_2,B_1\}$ cannot be a closer pair because we have:
  $\text{ED}(R_2,B_1) \geq \text{LB}(R_1,B_1) - \text{ED}(R_1,R_2)$, which is 8.1 – 1.0
  Since 7.1 is greater than our *bsf*, $\{R_2,B_1\}$ is pruned.

Thus, at least in this case, we only had to compute one new distance calculation, not four.

There is an obvious generalization of these ideas. Suppose, as shown in Figure 3, we have *many* points in R and B.



B1 can be seen as an "anchor point"
ED(B1,any(B)) = 1.5
ED(R1,any(R)) = 1.0
R1 can be seen as an "anchor point"

Figure 3: We can generalize the triangular inequality pruning to include a cohort of points, simply by recording the distance between an anchor point and the most distant member of the cohort.

We could record each of these points' distance from its "anchor point" R_1 or B_1. However this would require significant memory overhead. It is sufficient to record just the largest distance and use that as a bound for all its cohort.

III. Defniitions and Background

We begin by outlining the definitions and notations used in this work. We start with the data type of our interest, which is *time series*.

Definition 1: A *time series* $\mathbf{T} = t_1, t_2, \dots, t_n$ is a sequence of real-valued numbers.

We are not interested in the global properties of a time series but rather in shorter regions called *subsequences*.

Definition 2: A *subsequence* $\mathbf{T}^{(i,m)}$ is a contiguous subset of values from T starting at index i with length m.

The Z-normalized nearest neighbor distance for all subsequences of length *m* in T can be stored in a meta time series, known as the *matrix profile* [15].

Definition 3: A *matrix profile MP* of time series T is the vector of the z-normalized Euclidean distance between all subsequence $\text{T}^{(i,m)}$ and their nearest neighbor $\text{T}^{(j,m)}$ in T.

When finding the nearest neighbor to each subsequence, we enforce an *exclusion zone* of *m*/2 before and after location *i* to avoid considering trivial matches [4]. The lowest values in a Matrix Profile (there will always be a tied pair) correspond to the Top-1 motif, i.e. the pair of subsequences that have the lowest mutual Euclidean distance.

IV. Lower Bounding the Matrix Profle

As the examples in the previous section hinted at, we plan to create a *lower bound* for the MP. This lower bound will be the same length as the true MP, but faster to compute. As shown in Figure 4 our starting point is to use the Piecewise Aggregate Approximation (PAA) [8] to downsample T.

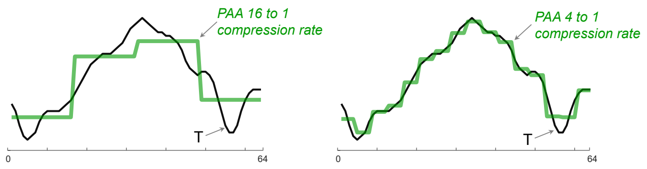


Figure 4: *left*) A course 16-to-1 approximation of a time series T. *right*) A fine 4-to-1 approximation of the same time series.

The PAA can be defined as [8]:

Definition 4: The PAA of time series **T** of length *n* can be calculated by dividing T into k equal-sized windows and computing the mean value of data within each window. More specifically, for each window *i*, the appropriate value is calculated by the following equation:

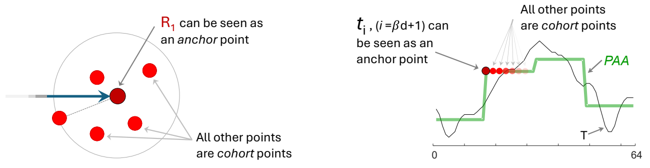
$$\bar{t_i} = \frac{k}{n} \sum_{j=\frac{n}{k}(i-1)+1}^{\frac{n}{k}i} t_j$$

Note that the examples in Figure 4 and all subsequent examples assume that the length of the time series is a power-of-two. This is not a requirement; the PAA is defined when m and/or n are arbitrary integers, this just simplifies exposition.

Assume we apply PAA to T using a downsample rate (dsr), producing an output T′ with length n/dsr. If we compute the MP on T′, then the computation is dsr^2 times faster than computing the MP on T. However, this "proxy" MP has two issues. First it is too short by a factor of dsr. This is easily fixed by upsampling—,simply repeating each value dsr times. The second issue is that the proxy MP is "weak". The true MP records the distances between subsequences of length m, but the proxy MP considers distances between subsequences of length just m/dsr, which generally have lower values. To correct this, we multiply T′ by $\sqrt{dsr}$ [8]. We call the result the *Approximate Matrix Profile*.

Definition 6: The approximate *Matrix Profile* AMP_{dsr} is the MP vector computed on T′. The computed MP is then upsampled by the same factor by repeating every value for dsr times and multiplied by $\sqrt{dsr}$.

Note that a special case of the AMP is $AMP_{1\text{-to-}1}$, which is simply the original Matrix Profile. In Figure 5, we show how this notation harkens back to the examples in the previous section. Note that each t_i ($i = \beta$d+1) is an *Anchor* point, and the following dsr-1 values are called *cohort* points.


R1 can be seen as an anchor point
All other points are cohort points
ti , (i =βd+1) can be seen as an anchor point
All other points are cohort points
PAA
T
0
64
Figure 5: *left*) (cf. Figure 3) The geometric example considered in the previous section has a perfect analog with the AMP_{dsr} (*right*).

Why did we make this connection? As shown in Figure 6, if we plot AMP_{dsr} and MP together it *appears* that the AMP_{dsr} *lower bounds* the MP, however this is not the case!

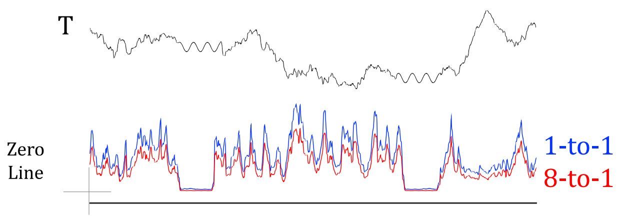


Figure 6: The MP and the $AMP_{8\text{-to-}1}$ for time series T. Visually the AMP_{dsr} appears to lower bound the MP, but this is *not* the case.

The AMP_{dsr} is only guaranteed to be a lower bound for MP at locations that are aligned to anchor points, and even then, it is only lower bounding to other subsequences that *also* happen to align to anchor points.

To define a true lower bound for all values in MP, including all the cohort subsequences, will take additional work. We begin by introducing the *K-Triangular Inequality Profile*.

Definition 7: A *K-Triangular Inequality Profile KTIP*, is defined as $d^{(i,x)}$ for x ∈ (i, j), where t_i and t_j are two consecutive anchor points and t_x is t_i's furthest cohort point.

The individual values in the KTIP can be seen analogous to the circle shown in Figure 5.*left*. They define a region of variability or uncertainty of subsequence shape around the anchor point. To make a true lower bound, we must "compensate" for this uncertainty by subtracting it from the true lower bounding information that we do know. In the example in Section II, we did this with

$$LBTI(B_2,R_2) = LB(R_1,B_1) – [ED(R_1,R_2) + ED(B_1,B_2)]$$

In the notation below, we show a perfect analogue of this: a final lower bound that comprised of a lower bound, minus the sum of two triangular inequality-based compensations.

Based on the intuition explained in Section II, we propose using KTIP to define the *Lower Bound Matrix Profile*.

Definition 8: A *Lower Bound Matrix Profile* $lbMP_{dsr}$, is a vector of distance values, $lb_{dsr} = [lb_1, \ldots, lb_{n-m+1}]$, where $lb_i = \max[amp^i_{dsr} - (ktip^i + ktip^j)]$ for $j \neq i$.

$lbMP_{dsr}$ is a parameterizable lower bound for the MP. Note that it has an interesting special case: when dsr is 1 we have $lbMP_{1\text{-to-}1}$ = MP. Thus the MP is a special case of $lbMP_{dsr}$.

More generally, as dsr is set to larger values, the time needed to compute it decreases, but the tightness also decreases. To see this, we need a formal metric. We define TLB, the *Tightness of the Lower Bound* as:

$$TLB = 1 - [\ ED(MP, lbMP_{dsr}) / ED(MP, ZeroLine)\ ]$$

As illustrated in Figure 7.*right*, TLB ranges from zero to one, with zero being a useless lower bound, and becoming more effective as it approaches one.

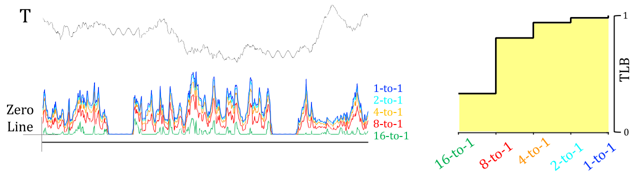


Figure 7: *left*) Five lbMPs of time series T for various levels of downsampling. Note that the special case of $lbMP_{1\text{-to-}1}$ is just the normal Matrix Profile. *right*) The Pareto frontier of the time needed to compute each $lbMP_{dsr}$ vs. its tightness.

How can we exploit this lower bound? All Matrix Profile algorithms initialize a *best-so-far* variable (*bsf*) to infinity, and then incrementally reduce it until it is the true distance of the Top-1 motif pair. This suggests a pruning rule: any region of the time series corresponding to a section of the lbMP that is greater than the current *bsf* can be admissibly pruned.

As shown in Figure 7, we have a *parametrizable* lower bound. This suggests the need for careful consideration of the trade-off involved. We could invoke a fast computation, but then only obtain a weak lower bound. Alternatively, we could spend additional computational resources to obtain a tighter lower bound. This will almost certainly allow us to prune more, but will this more aggressive pruning pay for itself? This is a very difficult thing to optimize, as the minimum useful tightness of a lower bound depends on the current value of the *bsf* variable. However, the current value of the *bsf* variable will change as the algorithm runs. Moreover, the current *bsf* variable itself depends on two things:

- The final true distance of the Top-1 motif pair. Clearly, that is a lower bound for the best *bsf* variable we can see.
- How fast the algorithm finds at least a good (i.e. *low*) *bsf* variable. This, in turn, depends on the algorithm's search strategy and the data itself.

To make this concrete, consider the two time series shown in Figure 8. In Figure 8.*left*, a lbMP$_{8\text{-to-}1}$ is sufficient to prune almost all the data. A lbMP$_{2\text{-to-}1}$ used here would take sixteen times longer to compute but is no more effective at pruning. In contrast, in Figure 8.*right*, a lbMP$_{8\text{-to-}1}$ can prune almost nothing and is simply a waste of computation.

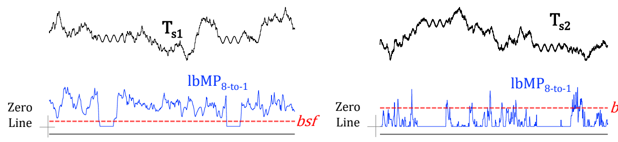


Figure 8: In these examples, the pruning algorithm has a *bsf* that is currently ~8% greater than the true final motif distance. *left*) A 8-to-1 AMP can prune almost all the data. *right*) A 8-to-1 AMP cannot prune almost no data.

In the next section, we solve this issue with MOMP (**M**otif **O**nly **M**atrix **P**rofile). MOMP begins with a coarse lbMP$_{dsr}$ and then iteratively passes any surviving (i.e. unpruned) data to finer lbMP$_{dsr}$ steps. In the limit, the finest lbMP$_{dsr}$ is the lbMP$_{1\text{-to-}1}$, which is just the classic Matrix Profile.

This strategy has two notable effects: it removes the "guesswork" of choosing the right down-sampling level and tends to quickly reduce the *bsf* variable. This, in turn, has two positive effects. It maximizes the pruning effectiveness, hence speeding up the algorithm, and it makes the algorithm strongly *anytime*.

V. MOMP

In this section, we introduce the MOMP algorithm. For simplicity of presentation, we consider only the case of Top-1 motif, self-join, and *m* being a power-of-two. However, all the generalizations of these assumptions are trivial and have already been implemented [10].

A. Introducing MOMP

We begin by explaining the intuition behind MOMP. The core idea is to attempt to prune as much of the time series as possible with the coarsest (and therefore *cheapest*) lower bound. Any time series that survives that pruning is then considered at a sampling rate that is twice as fine. The fact that the data is twice as fine means that the Matrix Profile computation would be four times as slow *if* all the data were unpruned. However, if any amount of data was pruned, this finer computation will have been accelerated. We iteratively continue this *prune-then-upsample* step until the unsampled data is at the original (i.e., 1-to-1) sample rate. At this point, the Matrix Profile algorithm searches the remaining data and returns the true best motif pair.

This algorithm is formalized in Table 2.

Table 2: The MOMP algorithm

```
Function:     MOMP(T, m)
Input:        T : Input time series
              m:  Subsequence length
Output:       momp_out: Distance matrix
1   T0 = T
2   dsr = m/32 # Set initial coarse down sample rate
3   bsf = inf
4   full_ktip = computeKTIP(T0, m, dsr)#Table 3
5   while true
6     ip = full_ktip(:, log2(dsr))
7     lbMP, local_bsf = computeLBMP(T, m, dsr, ip)#Table 4
8     bsf = refineBSFloc(T0, m, dsr, local_bsf, bsf)#Table 5
9     prnT = prune(T0, m, lbMP, bsf) #Table 6
10    T = prnT
11    dsr = dsr/2
12    if dsr == 1
13       mp, motifloc = SCAMP(T, m) #Or any other MP algorithm
14       return min(mp), prnT.indices(motifloc)
```

The algorithm starts by taking in the input time series, T and the user's choice of subsequence length *m*. In line 1, the input time series is assigned to T_0 as the original input. Then in line 2, the initial downsampling rate (*dsr*) is set. In line 3 the *bsf* value is initialized to infinity. Line 4 computes the full KTIP matrix explained in Table 3 on T_0. Each column of this matrix is later used at the appropriate downsampling level.

The *prune-then-upsample* loop starts at Line 5. Line 6 selects the corresponding KTIP array, then uses it in line 7 to compute the lbMP. The lower bound algorithm outlined in Table 4 returns the lbMP$_{dsr}$ along with a *bsf* value. We call this the local *bsf*. This local *bsf* only represent motifs that start at an anchor point (recall Figure 5.*right*). However, a tighter motif pair might be possible if we also consider the cohort points around the anchor point. Thus line 8 executes a quick local tuning on T_0 to refine the local *bsf* value and return the (generally smaller) current *bsf*.

In line 9, the algorithm admissibly prunes any section of the time series that has a corresponding region of the lbMP$_{dsr}$ that is greater than the *bsf*, and in line 10, the pruned time series is promoted to the next iteration. In line 11, the downsampling rate is divided by two so that the next iteration is working with data that is twice as finely sampled. Line 12 checks to see when downsampling rate reaches one; at that point, the exact MP computation is done on whatever regions of the time series have survived pruning up to that point. Line 13 computes the final MOMP results, and line 14 returns the output.

Note that there are no parameters for MOMP. We do need to pick a starting downsample rate. As shown in line 2 of Table 2, we use (*m*/32)-to-1, but replacing the 32 with 128, 64 or 16 makes no measurable difference. The number of times the loop beginning at line 5 iterates is log$_2$(*m*/32).

This outline explains MOMP. We can now consider it subroutines in more detail, by further examining the details of KTIP computation (Table 3), lbMP computation (Table 4), refinement (Table 5) and pruning (Table 6).

In line 2, the ktip matrix is initialized as all NaN values. This matrix is the length of the MP array, and its column count is set to the number of MOMP steps. A temp variable is defined in line 3 for temporary storage of minimum values.

Table 3: K-Triangular Inequality Profile Algorithm

```
Function:    computeKTIP(T, m, $dsr_0$)
Input:       T: Input time series
             m:  Subsequence length
             $dsr_0$: Initial downsampling rate
Output:      ktip: lower bound Matrix Profile
1   n = len(T)
2   ktip = nan(n-m+1, log2($dsr_0$))
3   temp = nan(n-m+1, 1)
4   for diag = 1:$dsr_0$
5       for rr = 1:n-m-diag+2
6           cc = row + diag - 1
7           dist ← ED(T(rr:rr+m-1), T(cc:cc+m-1))
8           if dist < temp(rr)
9               temp(rr) = dist
10          if dist < temp(cc)
11              temp(cc) = dist
12      if ispow2(diag)
13          ktip(:, log2(diag)) = temp
14  return ktip
```

Then lines 4 to 13 compute the minimum distances, and in line 12 and 13 the required ktip values are stored. Finally, line 14 returns the full KTIP matrix. This KTIP is then used in computing the lbMP using the algorithm in Table 4.

Table 4: Lower Bound Matrix Profile Algorithm

```
Function:    computeLBMP(T, m, dsr, ip)
Input:       m:  Subsequence length
             dsr: Downsampling rate
             ip: ktip for dsr step
Output:      $lbMP_{dsr}$: lower bound Matrix Profile
1   lbMP = nan(size(amp))
2   dT = PAA(T, dsr)
3   amp = SCAMP(dT, m/dsr)
4   for i = 1:len(amp)
5       lbMP(i) = $\max_{j\in[1:len(amp)], j\neq i}$(amp(i)-ip(i)-ip(j))
6   $lbMP_{dsr}$ ← upsample(lbmp, dsr)
7   return $lbMP_{dsr}$, min($lbMP_{dsr}$)
```

In line 1, the lbMP array is initialized as NaN values. Then in line 2, the input time series is downsampled by dsr-to-1 using PAA (recall Definition 4). Line 3 computes the MP on the downsampled time series allowing lines 4 and 5 to compute the lower bound for every value in *amp*. Line 6 upsamples the computed lbMP by factor dsr to maintain the original length. Finally line 7 returns the $lbMP_{dsr}$ and its minimum value which is an approximate *bsf*. Recall that, as shown in Figure 5.*right*, this *bsf* is limited to representing motifs that start at an anchor point. In Table 5, we use a refinement function to adjust the approximate *bsf* locally, by considering the cohort points to the right of the anchor points that currently have the *bsf* value.

Table 5: Best-so-far Local Refinement

```
Function:    refineBSFloc(T, m, dsr local_bsf, bsf)
Input:       T: Input time series
             m:  Subsequence length
             dsr: Downsampling rate
             local_bsf: bsf found by $lbMP_{dsr}$
             bsf: current bsf value
Output:      bsf: updated bsf value
1   i,j ← local_bsf.loc
2   segA = T(i: i+m+dsr-1)
3   segB = T(j: j+m+dsr-1)
4   [mp, minloc] = SCAMP([segA, segB], m)
5   If min(mp) < bsf
6       bsf, bsf.loc ← min(mp), T.indices(minloc)
7   return bsf
```

In line 1, the locations of the local motifs found by $lbMP_{dsr}$ is assigned to i and j. Then, in lines 2 and 3, two small segments of time series are made, covering the approximate motif locations and the dsr cohort points to the right. Then line 4 computes a classic matrix profile on this tiny subset of T. Lines 5 and 6 updates the *bsf* value if a better motif is found, and Line 8 returns the *bsf*. Then we use the approximate and refined *bsf* values to prune T, using the algorithm in Table 6 and continue to the next iteration.

Table 6: Pruning Algorithm

```
Function:    prune(T, m, lbMP, bsf)
Input:       T: Input time series
             m:  Subsequence length
             bsf: refined bsf value
Output:      prnT: pruned time series
1   prnT = []
2   tgts ← locate(lbMP ≤ bsf)
3   for t in tgts
4     prnT.concatenate(T(t:t+m-1))
5   return prnT
```

In line 1, the pruned time series is initialized as an empty array. The target subsequences are found in line 2 by applying the *bsf* threshold to lbMP values. Then, in lines 3 and 4, the target subsequences are concatenated. The overlaps and alignment of concatenation are handled in the original code, but this is glossed over here for brevity. Line 5 returns the pruned time series.

B. *MOMPs Cost Model*

Since the affairs of men rest still uncertain, Let's reason with the worst that may befall. William Shakespeare

The cost for MOMP is almost completely dictated on the final prune rate at the time we finish computations on the 2-to-1 down-sampled data and prepare to pass the non-pruned data at full resolution to the classic MP algorithm in line 13 of Table 2. Thus, the speed achieved over classic MP can be approximated as: $1/\,(1 - prune\ rate\ at\ DSR_{1-in-2})^2$.

However, this model ignores the overhead cost of lines 1 to 12 of Table 2. Here there is some variability. In the best case, the coarsest down-sampling might prune almost everything, and this overhead would be almost immeasurably small. However, let us consider the worst case, in which nothing is pruned. The overhead would consist of fruitless computation of the SCAMP algorithm at down-sampled data with dsr of 2-to-1, 4-to-1, 8-to-1 etc. These would have a cost of ${1/2}^2$, ${1/4}^2$, ${1/8}^2$ etc., which sums to an overhead of ~33%.

There is also a small amount of overhead for down-sampling/up-sampling, computing the KTIP, refining the *bsf* etc. Empirically, the total overhead is typically less than a factor of two. Thus, as shown in Figure 9, the acceleration achieved by MOMP over MP is well modeled as: Speedup = $(1/\,(1 - prune\ rate\ at\ DSR_{2-to-1})^2)/2$

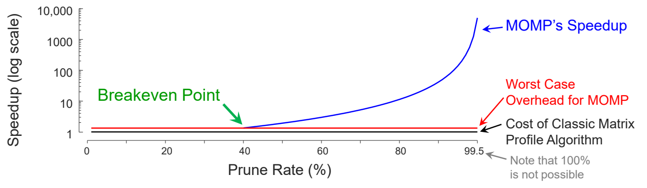


Figure 9: The cost model for MOMP

As shown in Figure 9 this implies that MOMP "breaks even", that is to say, covers it's cost overhead, once the prune rate reaches about 40%. As the prune rate surpasses this threshold, the speed up rapidly makes the overhead inconsequential. For example, if we can prune 90% of the data, the speed-up is about forty-one times faster; and if we prune 99% of the data, the speed-up is about 1,250 times faster. Remarkably, as we will show in Section VI, such prune rates and speedups are observed in real-world datasets.

C. *Observations about MOMP*

Here we discuss some of the properties of MOMP.

- MOMP is *exact*, it produces identical motifs to the classic MP algorithms, thus we do not need to measure accuracy, precision or F1 etc., we only need to report speedup.
- MOMP is independent of the base Matrix Profile algorithm used. It can use STAMP, STOMP, SCAMP, SCRIMP, SCRIMP+, or their GPU multi-threaded versions, or their reduced precision versions [12], etc.
- Some of the original MP algorithms, notably SCRIMP++ are strongly *anytime* algorithms. As we show in Figure 14, MOMP not only inherits this property, but it can convert the currently batch-only MP algorithms such as STOMP or SCAMP into strongly anytime algorithms.
- MOMP is independent of the lower bound used. In principle, MOMP could be used with any new lower bound invented in the future. There is a simple test to see if any newly proposed lower bound could help accelerate MOMP: it needs to be *above* the Pareto frontier shown in Figure 7. For the simpler problem of *time series similarity search*, there is a rich history of alternative lower bounds [8]; we suspect this will be a fruitful area for future research.
- Since the introduction of the Matrix Profile, the community has produced many extensions including Motif-Joins, Consensus Motifs, Chains, K-motifs, multi-dimensional motifs, etc. MOMP can be used to accelerate all these primitives. For brevity, we confine our demonstration to Motif-Joins in Section V.D. Note that MOMP does not accelerate *discord* discovery, but there is already DAMP, an ultra-fast *upper* bound method for this task.
- MOMP is more *sampling-rate invariant* than classic MP algorithms. Imagine Alice and Bob both record the exact same hour of EOG data using a Edanusa X12 device [6], but Alice at the lowest setting of 50 Hz, and Bob at the highest setting of 400 Hz. As shown in Figure 1, they both find the same top motif of length twenty seconds, but while Alice's search take 109 seconds, Bob's search takes 117 minutes. This is about sixty-four times longer, as we expect from the O(n^2) time complexity of SCAMP. However, MOMP is only 5.1 times slower when using the finer resolution data[8]. This is because the performance of MOMP is more influenced by the *intrinsic* dimensionality of the data, not the actual sampling rate. This is a very useful property, as many datasets are arguably greatly oversampled.
- The worst-case *time* overhead for MOMP as shown in Figure 9 is remarkable. Most algorithms based on lower bounding in the dimensionality reduced space have good *best* cases, but in the worst case they can be orders of magnitude slower than a simple brute force search. For example, the recently introduced Attimo[9] has a good best case for motif discovery [3], but as we show in [10], in the worst case it is hundreds of times slower than brute-force.
- The worst-case *space* overhead for MOMP is just O(n). Once again, many algorithms have good *best* cases but untenable worst cases. For example, the space complexity for QuickMotif is O(m + *survivors*) [9]. The number of *survivors* might be small, but in the worst case there can be O(n^2) *survivors*. Previous work noted that "*the memory footprint for Quick-Motif tends to be very large*" [16], and that was when considering datasets that are less than one-hundredth the size of the datasets we consider in this work.

D. *Limitations of MOMP*

There are two bad cases for MOMP. Interestingly, they are essentially two opposite cases:

- ***Nothing* is a Motif**: If the data is just random noise, then the *bsf* variable will never be significantly smaller than any part of LB (see line 2 of Table 6) and no pruning will take place. Of course, in such a situation, there are really no semantically meaningful motifs, because the closest pair of subsequences will only be slightly closer than any random pair of subsequences [1].
- ***Everything* is a Motif**: Suppose the data is a sine wave with a little noise. Here there *are* visually satisfying motifs, but any random subsequence with its nearest neighbor will almost be as similar. Because of this, all lower bounds will be close to zero everywhere, and no pruning is possible.

Obviously, we do not normally expect to work with such pathological datasets; however, real datasets may approach either case. There are two other scenarios where MOMP does not offer significant improvements. First, if n is small, SCAMP is so fast that there is little room for improvement. The second case is if m is small. Given that MOMP is doing a multi-resolution search, if m is very short, there simply are not multiple resolutions to search over.

VI. Experimental Evaluation

To ensure that our experiments are reproducible, [10], which contains all data/code for the results, in addition to many experiments that are omitted here for brevity.

Unless otherwise stated, all experiments were run on a Dell XPS 8920, with Intel Core i7-7700 CPU @ 3.6GHz and 64GB RAM. While there is great interest in using HPC for time series motif discovery [12][17][18], here we intentionally use a dated and underwhelming machine.

Note that we mostly consider motif lengths that are a power-of-two. This is *not* a limitation of our work; the PAA representation is defined for arbitrary lengths [8]. However, later research may wish to build alternative lower bounds based on wavelets or DFT, and both those representations have their best cases for n and/or m equals a power of two [8].

[8] MOMP's subquadratic scaling is due to its improving pruning ability in the finer data. For the low-sample-rate data MOMP prunes 91.4% of the data, but for the high-sampling-rate data it prunes 96.9% of the data. Full details in [10]

[9] Attimo is not an *exact* time series motif discovery algorithm. It finds the best motif with a user specified probability [3].

We use a highly optimized MATLAB version of batch-only SCAMP as both the base algorithm for MOMP and our rival strawman [18]. Because both approaches use the same base algorithm on the same hardware, any speed differences can be solely attributed to our algorithm.

There is almost no other algorithm we can compare to. The recent Attimo [3] and HIME [7] are *approximate* algorithms, but we are only interested in *exact* search. Quick-Motif [9] is an exact algorithm, and we do compare to it.

There are dozens of papers that expand on the Matrix Profile. However most, such as *Motiflets* [13], use a standard algorithm such as SCAMP, but offer different ways to extract the motifs once the Matrix Profile has been computed.

A. *Establishing Two Important Facts*

We begin by establishing two facts that we will use for the rest of our empirical evaluations.

- We can perfectly predict how long SCAMP will take, given only length of the input time series (see Appendix B).
- The cost model introduced in Section V.B is reasonable.

To see this, we can conduct an experiment to measure the speed up obtained by MOMP in the face of increasing noise. We created a random walk of length 2^{20}, and using SCAMP we measured how long it takes to find the Top-1 motifs of length 2^{11}. As shown in Figure 10.*right*, it takes SCAMP about 4,976 seconds (about 1.38 hours).

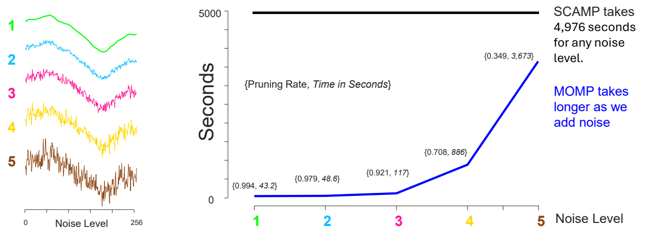


Figure 10: *left*) The first 256 datapoints of the increasingly noisy datasets considered. *right*) The time for SCAMP is independent of noise level. However, while MOMP is initially two orders of magnitude faster, it slows down in the face of increasing noise.

We can see that MOMP is impressively fast, about 115 times faster than a direct use of SCAMP. However, as we add noise to the dataset, MOMP begins to slow down. This is to be expected. In the limit, a very noisy dataset puts us in the "*nothing is a motif*" situation explained in Section V.D.

Now let us consider the *raison d'etre* for this experiment.

- In this case, it took 4,976 seconds for SCAMP to process $n = 2^{20}$ datapoints. How long would it take to process say $n = 654,321$? Using the formula in Appendix B we predict it would take 1,937 seconds. Empirically measuring it, it takes 1,939 seconds, a prediction error of less than 1%.
- Consider the speed up obtained for the noisiest experiment. Our cost model predicts we should take 84.8% of the time for SCAMP. We actually took 73.8% of the predicted time. This suggests that our cost model is a little conservative.

These observations tell us that we can report either the wall-clock time or just the cost model's predicted speedup. This is useful for two reasons. For some of the experiments we wish to do, SCAMP would take years or longer, which is clearly impractical. Secondly, the speed up numbers are independent of the MP algorithm, the quality and optimization of the algorithm, and the hardware used. We use the most appropriate of the two measures on a case-by-case basis below, with *all* measures archived at [10].

B. *MOMP's Speed Up on Diverse Datasets*

In Table 7, we evaluate MOMP on a collection of diverse datasets from science, industry and medicine. The times for MOMP, and the times for SCAMP that are less than 12 hours are measured wall-clock times. Longer times are extrapolated as discussed in Section VI.A.

Table 7: SCAMP vs. MOMP on various datasets.

Name	**Length *n***	**SCAMP**	**MOMP**	**SpeedUp**
EOG400 Hz (Figure 1)	1,439,997	2.03 hours	0.03 hours	60.7
SWaT Attack 7	449,919	12.0 min	2.3 min	5.0
Human$_{\text{Y-chromo}}$ [3][7]	26,415,043	28.6 days	0.11 days	259.3
EEG P_2O_2	2,472,001	6.1 hours	1.75 hours	3.3
SleepElectromyography	5,983,000	35.5 hours	0.85 hours	41.1
EOG (during sleep study)	8,490,000	71.05 hours	4.82 hours	14.7
Respiration (Challenge)	1,799,997	3.19 hours	0.15 hours	20.4
Insect EPG$_{\textit{Kryder trifoliata}}$	7,583,000	2.36 days	0.64 days	3.6
Insect EPG$_{\textit{Poncirus trifoliata}}$	7,583,000	2.36 days	0.033 days	231.5
Chicken Behavior	8,595,817	3.02 days	0.107 days	28.1
Kittiwake (Flying wild bird)	1,288,330	1.63 hours	0.11 hours	13.6
HAR Ambient Sensor	1,875,227	3.27 hours	0.098 hours	35.4
Electroencephalography	6,375,000	1.67 days	0.01 days	144.3
WaterDemand	2,100,777	4.35 hours	0.131 hours	32.8
Micro PMU	62,208,000	158.9 days	0.19 days	800.3
Stator Winding	1,330,816	1.74 hours	0.033 hours	52.4
Household elec' demand	5,153,051	1.09 days	0.0087 days	125.6
Wind Turbine	5,231,008	1.12 days	0.068 days	16.4
NASA SolarWind	8,066,432	2.67 days	0.012 days	220.0
ECoG (Finger Flexion)	23,999,997	23.65 days	0.126 days	188.2
SpanishEnergy	25,232,401	26.15 days	0.51 days	51.3
EEG CinC (Figure 13)	1,593,750	2.50 hours	0.013 hours	181.2
WearablesLab(Figure 16)	4,031,969	16.0 hours	0.097 hours	165.4
Physical Activity NIH	4,741,248	22.2 hours	0.17 hours	123.8
OSA Sleep Apnea	8,340,000	68.3 hours	2.2 hours	30.9
Random Walk	67,108,864	164.64 days	0.064 days	2565.8

(Because Random Walk is the only *synthetic* dataset, we exclude it from summary statistics below)

While *negative* speedups are logically possible (see Figure 9), none are observed. The speedups range from 3.3 (EEG P_2O_2) to 800.3 (Micro PMU) with a mean of 113.9. It is difficult to see any obvious predictors of speedup (beyond the factors discussed in Section V.D). The two insect EPG datasets are not visually distinct yet have dramatically different speedups. It is only with a *post-hoc* analysis do we see that Insect EPG$_{\textit{Poncirus trifoliata}}$ happened to have a better conversed motif, allowing for more aggressive pruning.

We do not have space to provide detailed provenance for each dataset, but this information is archived at [10]. However, in most cases, the motifs make intuitive sense to the domain experts we asked to review the findings. For the chicken behavior dataset, the motif reflects *dustbathing*. For small values of *m*, *pecking* is the most conspicuous and frequent motif in chickens. However, here we take advantage of the scalability of MOMP to look for much longer motifs in

24-hour period. Healthy chickens normally only dustbathe every two days [11], so this discovery of two bouts in a single day offers evidence of possible infestation by parasitic mites.

For the Kittiwake dataset, the motif reflects a bird (*Rissa tridactyla*) transitioning from gliding behavior to flapping flight. For EOG data, the motif represents corneal reflex blinks. For WearablesWetLab, it signifies the act of *pouring*.

In Figure 11, we show representative examples of some of the motifs found during these experiments.

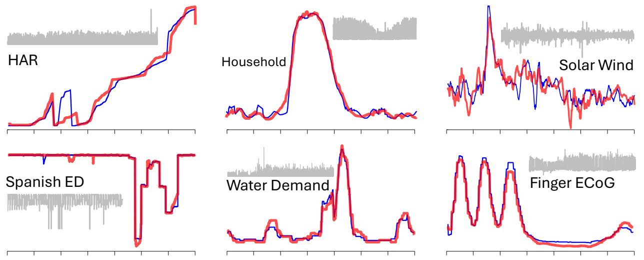


Figure 11: Six representative motifs found during the experiments described in Table 7. Inset in gray are the entire original datasets.

Note that we included one tiny dataset, SWaT-Attack-7 in Table 7, as it is the only dataset Quick-Motif could process without running out of memory [9]. When we use the original m = 16,384, Quick-Motif *still* ran out of memory (Quick-Motif's memory problems are also noted in [17]). However, by reducing m to 4,096 we got it to work. Quick-Motif was about seventeen times slower than SCAMP. Quick-Motif has three parameters: w/ℓ, λ and ε. It is possible that better settings of these parameters could close the time gap, but its memory footprint issue seems insurmountable.

C. Searching The Entire Human Genome

In Table 7, we converted the Human Y chromosome (H_Y) to a time series using the algorithm in Appendix A and searched it for motifs. This dataset's length is 26,415,043 bp (base pairs), and it was used as a capstone experiment in two papers that could only search it *approximately* [3][7]. Note that it is by far the smallest of the 23 human chromosomes.

To demonstrate the scalability of MOMP, we considered an experiment that would be otherwise untenable. We began by converting the entire human genome (2,727,047,427 bp) into 23 time series representing all the human chromosomes. We searched H_1 for the best motif of length 65,536. We now can ask the following question: *Which of the remaining 22 chromosomes have a motif as well conserved as H_1?*

To answer this question, we simply placed MOMP in a loop that ran 22 times. The only difference from normal MOMP is that instead of initializing the *bsf* in line 3 of Table 2, we initialize a global *bsf* outside of MOMP, and pass that value into MOMP. We can then exploit the anytime algorithm property of MOMP by stopping the search of a chromosome, if we find a qualifying small motif distance.

The largest chromosome is H_2 with a n = 238,357,386. To find the exact motif in H_2, we must compare or prune 25,734,779,520,915,612 candidate subsequence pairs of length 65,536. If each comparison takes one microsecond, this requires 815 years. SCAMP cleverly reduces the factor in m to just O(1), and thus takes only 5.07 years. Here, MOMP took 1.08 hours, about 41,000 times faster.

In five of the twenty-two Chromosomes, *anytime* MOMP's early abandoning did work. For example, on H_9 (n = 121,526,601), anytime MOMP found a qualifying motif in 12.4 minutes, whereas SCAMP takes 1.32 years.

Note that the H_1 motif does not have zero distance, as zero-distance motifs can be found in linear time. Consider these excerpts we mapped into the original DNA string:

```
Motif-1 (103,286,375 bp omitted) ..TGGCACAATGTCAC..
                                    ||||||  |||||||
Motif-2 (103,426,107 bp omitted) ..TGGCAC-ATGTCAC..
```

We can see that there are small local micromutations (either the insertion of **A** into Motif-1 *or* the deletion of **A** from Motif-2) that happened after the transposition event. By counting these micromutations it is possible to approximately predict when the transposition event occurred.

D. AB Join MOMP

The genomes of the Human and Chimp are nearly identical in structure, except for five large-scale inversions (and one chromosome fusion, which we ignore [14]). Let us use a *join* to find one such large-scale inversion. Here we follow the notation of the original MP paper [15] and generalize the MP to consider J_{AB}, which is simply a motif consisting of the subsequence in **B**, that is closest to *some* subsequence in **A**. We denote this generalization J_{AB}-MOMP.

A DNA inversion appears as a time series subsequence reversed in time. So, if the i^{th} chromosome does not have an inversion, we should expect that the Euclidean distance of the Top-1 join-motif between the Human's i^{th} chromosome (Hereafter H_i) and C_i is much smaller than the Top-1 join-motif distance between the H_i and `Backwards`(C_i), as the reversal will not reveal hidden structure.

If, however, the chromosome does have an inversion, we should expect the Euclidean distance of the top-1 join-motif between the H_i and `Backwards`(C_i) to be about the same as the top-1 join-motif distance between original chromosomes. This is because our *digital* reversal undoes a *biological* inversion that took place sometime after the Human|Chimp split six million years ago.

We first consider H_{21} and C_{21} which have lengths of size 35,186,393 and 32,724,802 respectively. Here the reversed search found a motif pair with a distance of 164.49, which is dramatically (and *visually*, see Figure 12) further apart than the normal motif pairs distance of just 27.33.

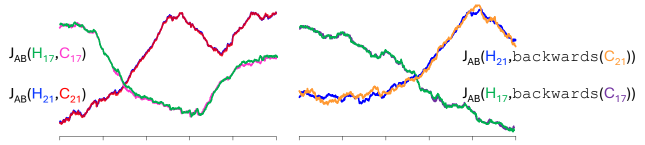


Figure 12: Top-1 motifs pairs from the four AB-join experiments. The pair corresponding to the reversal of C_{21} is visibly less conserved than the others, looking no better conserved that random chance, suggesting that this chromosome is *not* the source of an inversion.

We then considered H_{17} and C_{17}, which have lengths of size 79,910,446 and 81,665,723 respectively, finding that the reversed search found a motif pair with a distance 54.45, which is actually slightly less than the normal motif pairs distance of 76.87. This strongly (and *correctly*, see [14]) indicates that this pair of chromosomes *has* a large-scale inversion.

The average pruning rate for all four joins was 0.934, meaning that our experiments ran about 139 times faster than J_{AB}-SCAMP. Note that the pruning rates are not particularly high here because for most of the joins, we are in the "*everything is a motif*" situation as described in Section V.C. Furthermore, note that the anytime convergence is so fast here, that as a practical matter we could have answered the original question in a few minutes on a typical laptop.

This is not an original biological finding, and it could be done better/faster in the original DNA representation. Our example simply shows that MOMP allows accelerated time series similarity joins between massive datasets. To the best of our knowledge, these joins are, by two orders of magnitude, the largest time series joins ever attempted [15].

E. *MOMP as an AnyTime Algorithm*

Consider the 8.8 hour-long EEG CinC example in Table 7. As shown in Figure 13.*top* the discovered motif is unusually well conserved and perfectly periodic, something we do not expect to see in real medical data. A few seconds of introspection by a medical technician would have them realize that the motif does not reflect a biological signal. After about 6.69 hours, the electrode patch that holds the sensor to the patient's skin became loose. When this happens the sensor does not "flatline" but instead emits a calibration signal. this is what this motif represents.

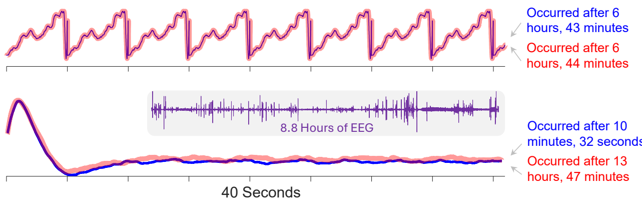


Figure 13: *top*) The Top-1 motif discovered in the EEG CinC dataset is just a calibration signal. It looks too well conserved to be natural, given how noisy the data is (gray *inset*). After removing the calibration signal, the Top-1 motif *is* a biological signal (*bottom*).

When a user sees this, she can delete the calibration signals and rerun motif discovery. However, consider the three following scenarios. If we had used SCAMP to find this motif, this cycle would have taken 2.50 hours. Using MOMP as a batch algorithm, it takes about 79 seconds. However, using MOMP as any anytime algorithm and visually inspecting the current *bsf* motif, we would have seen this calibration motif after about eight seconds.

This observation motivates the use of an anytime framework for motif discovery. Although the full algorithm may take minutes or hours to finish, often a visual inspection in the first few seconds will allow the user to stop because:

- It is clear that the motif that will be returned is pathological, as in the EEG case above, or
- The current best-so-far motif is already interesting enough to warrant investigation, so there is no point in continuing. Perhaps the current run can be run in the background while the user examines the current motif.

A useful property of MOMP is that it is an *anytime* algorithm, even if the underlying MP algorithm is not. To see this, let us conduct an experiment. We created a random walk of length 2^{20} and using SCAMP (a *batch-only* algorithm), we measured how long it takes to find the Top-1 motif $m = 2^{12}$. Figure 14 shows that this takes 34.7 minutes.

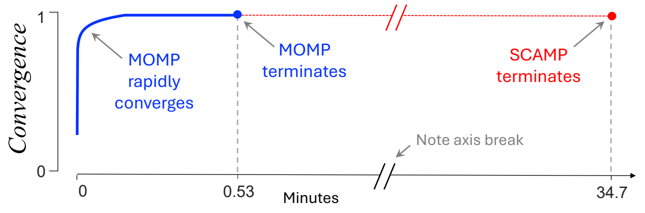


Figure 14: The time needed for SCAMP and MOMP to find the Top-1 Motif. For MOMP, we also plot its anytime convergence rate.

We repeated the experiment with MOMP, this time also measuring how fast it converges. This *convergence* is the normalized difference between the final motif distance and the *bsf* distance recorded each time through the loop in line 8 of Table 2. We defined a score to evaluate the convergence rate. The score is: $Convergence = 1 - \frac{currentBSF - finalBSF}{initialBSF}$.

Here, we define *initialBSF* as the average distance between two random subsequences (empirically determined). Because MOMP is data dependent, we averaged over one hundred runs. As Figure 14 shows, MOMP is an ideal anytime algorithm, quickly converging on high-quality motifs.

To make the speed of convergence more visceral, let us visualize the convergence of motif discovery on a challenging dataset of length 13.4 million datapoints.

Here SCAMP takes 7.4 days. MOMP is able to reduce this to just 7.1 hours. However, consider Figure 15 which shows the *best-so-far* motif available at two intermediate times. In the first two minutes the motif is relatively low quality, although the pair *are* somewhat similar, with the first 90% being a noisy constant region, and the last 10% having a dramatic increase in mean value.

However, after about two minutes, MOMP has converged on what will be the final answer returned in another seven hours (if we indeed let the algorithm run to full convergence).

This result is typical. In virtually all practical applications of time series motif discovery, regardless of the domain or downstream task, we should exploit the Hyper-anytime properties [45] of MOMP, STAMP, SCAMP or SCRIMP++.

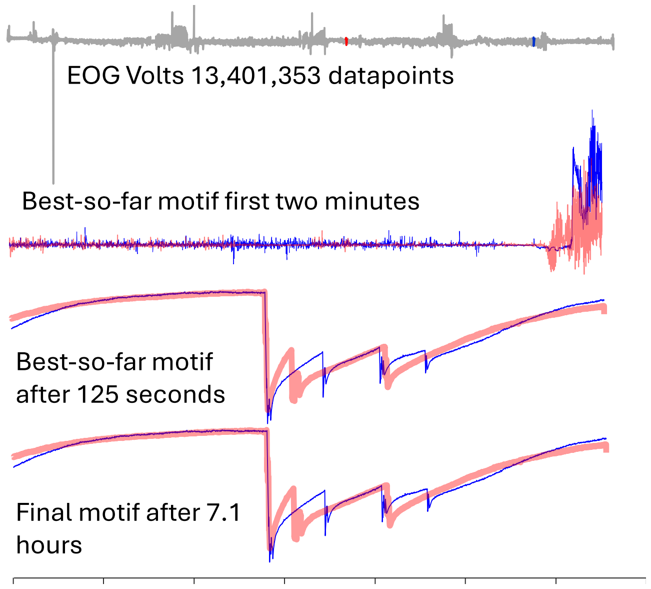


Figure 15: *Top-to-Bottom*). An EOG dataset of length 13.4 million datapoints. The *bsf* motif returned by MOMP in the first two minutes is poor, but after 125 seconds MOMP has already found what will be the final motif when MOMP fully converges 7 hours later.

Several papers published since the introduction of the Matrix Profile have proposed approximate motif discover algorithms. However, MOMP, STAMP, SCAMP and SCRIMP++ are their *own* approximation algorithms.

It would be foolish to suggest that these MP approximate algorithms are the best approximate motif discovery possible. However:

- They are clearly very *highly* competitive.
- They have an advantage over all proposed approximate algorithms. All currently known approximate algorithms run a batch algorithm to produce an approximation, then the user must examine the results and decide to "*take it or leave it*". If the user is *not* satisfied with the result, what should they do? Perhaps they could rerun the algorithm with different settings and hope for the best. In contrast, if the user of MOMP is not satisfied with the result after one minute, doing nothing and waiting another minute will either improve the results, or by *not* improving the result, offer more evidence that the dataset examined is bereft of well conserved motifs.

In summary, we strongly argue for the utility of anytime motif discovery.

F. *The Utility of Motif Discovery*

While the utility of motif discovery as a subroutine in downstream analysis is well established, for completeness we will show two concrete examples.

Suppose we wish to annotate a long video. If we have a companion time series, it may be easier to find motifs in the time series, and visually check if the corresponding video snippets show semantically similar behaviors. As shown in Figure 16, this idea does bear fruit.

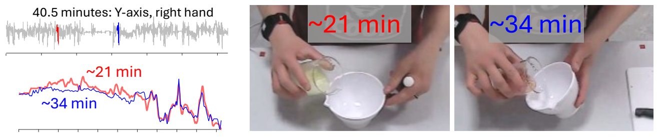


Figure 16: *left*) We searched for motifs in time series recorded in parallel to a 40.5 minute video showing benchwork in a wet lab. *right*) The motifs do correspond to a repeated behavior, *pouring*.

Here, MOMP took 5.8 minutes, much faster than real time. Our cost model predicts SCAMP would take 16.0 hours. We took advantage of the relatively small data size to run SCAMP to completion, finding it actually took 15.8 hours.

Similarly, for a doctor to annotate a sleep study, they must examine about eight hours of sleep telemetry. Working with Dr. Greg Mason we performed a simple experiment to see if motif discovery could reduce this labeling burden. The task was to annotate a dataset for an Obstructive Sleep Apnea (OSA) study. OSA is characterized by "*repetitive episodes of intermittent hypoxemia*" [5], and of course "repetitive" strongly suggests "motifs". As shown in Figure 17 we searched a seven-hour respiration dataset for motifs of length 80 seconds (the length was suggested by Dr. Mason).

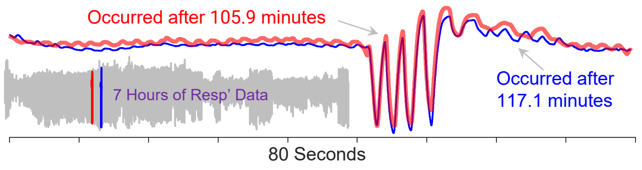


Figure 17: This motif corresponds to "*cycles of intermittent hypoxemia with cyclical desaturation-reoxygenation*". *inset*) the full seven-hour respiration dataset.

Having found a motif of interest, we can simply search for other occurrences to help label the dataset.

Here, MOMP took 2.21 hours, which is faster than real time. Our cost model predicts SCAMP would take 68.5 hours. We again took advantage of the small data size to run SCAMP to completion, finding it actually took 68.3 hours.

VII. MISTS: *M*OTIF DISCOVERY *IN S*ETS OF *T*IME *S*ERIES

We have waited until the reader's intuition for the utility and scalability of our multiresolution approach was fully developed to introduce a novel problem in data exploration and show that MOMP is ideally suited to solve it.

To motivate this problem, let us consider a dataset donated to the community by a research group from the University of Western Ontario [64]. The dataset contains the mitochondrial DNA (mDNA) of various groups of organisms. The mDNA of most familiar animals is about the same size, approximately 16,500 bp. Because this is so short, even a large collection of such DNA (for example, there are only 5,500 mammal species), could be brute-forced by SCAMP in minutes. However, as Figure 18 shows, the mDNA of plants is extraordinary variable, and includes some much longer time series. The mDNA of plants can range from about 20,000 to almost 2,000,000 bp[10].

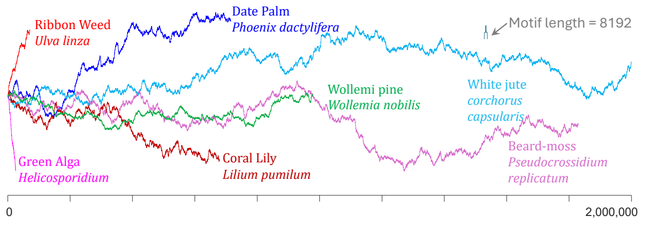


Figure 18: A random sample from the 174 plant mDNA dataset provided by [64].

Do *any* of these mDNA time series have high-quality motifs of length 8,192? In Figure 19 we show the Top-1 motif from two time series from this collection.

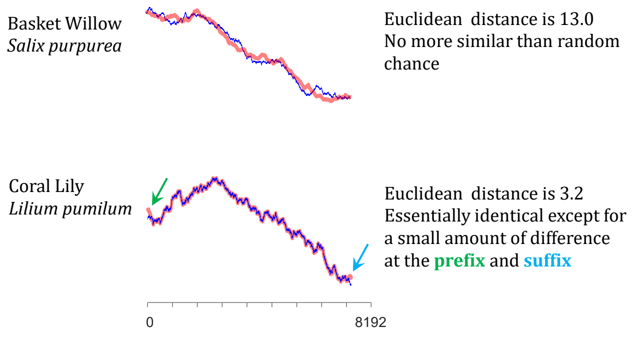


Figure 19: *top*). The best motif discovered in the Basket Willow mDNA looks no more similar than chance. *botttom*) In contrast, the best motif discovered in the Coral Lily mDNA is near perfectly preserved, suggesting a duplication event occurred.

The Top-1 motif found in the Basket Willow ($\mathbf{T}_{\text{BW}}$, with 598,970 bp) looks like the motif we might expect to find in a random walk time series of a similar length; it does not appear to be conserved above chance level. In contrast, in the Coral Lily time series ($\mathbf{T}_{\text{CL}}$, with 678,653 bp), the motif is conserved almost "bit-for-bit", except for two small regions at the beginning and the end.

Using SCAMP to find the motifs in these two time series takes 21.2 minutes and 27.2 minutes respectively. Unsurprisingly, MOMP is significantly faster, taking 36.5 seconds and 9.6 seconds respectively. The individual timings for MOMP may initially be surprising; the (slightly) longer dataset was more than three times faster. However, the final prune rate for $\mathbf{T}_{\text{BW}}$ was 0.930, whereas $\mathbf{T}_{\text{CL}}$ final prune rate was 0.963. This may seem like a small difference but recall from Figure 9 that the speed-up of MOMP is superlinear in the prune rate.

In fact, there is an opportunity to further improve MOMPs performance here. Because we wish to find the best motif in *either* time series, we can create a *global* best-so-far. We will denote this as *gbsf*, to distinguish from the *local bsf* discussed in Section V. After we compute the Top-1 motif search in the first time series we search, we can set the *gbsf* to the Euclidean distance of the discovered motif. Then, when we continue our search the second time series, we can modify line 3 in Table 2 to be initialized with `bsf = gbsf`, instead of `bsf = inf`.

As noted in Section III the speedup for MOMP depends strongly on the value of the *bsf*. The smaller this is, the more pruning that takes place, and the faster the algorithm is.

The reader will perhaps already appreciate that this idea has a problem, it is highly order dependent. If we search in the order $\mathbf{T}_{\text{BW}}$ *then* $\mathbf{T}_{\text{CoralLily}}$, there is no difference in the total time taken. But had we searched in the order $\mathbf{T}_{\text{CL}}$ *then* $\mathbf{T}_{\text{BaW}}$, the first search takes 9.6 seconds as before, but the second search goes from 36.5 seconds to just 8.8 seconds.

In general, to exploit the most benefit from the *global* best-so-far we should always search the time series with the best motif first. However, this seems to open an insurmountable chicken-and-egg paradox, as we do not know which time series has the best motif; that is what we are trying to discover.

Here we can exploit the unique multiresolution property of MOMP. Instead of using MOMP in a *serial* fashion over both time series, we can use MOMP in a *parallel* fashion. It is really a *pseudo* parallel algorithm, perhaps better seen as an *interleaved* algorithm. We can search *both* time series at the coarsest level, then set the local *bsf* for each to be the global *bsf*, which is just *gbsf* = min($\mathbf{T}_{\text{BW}}$*bsf*, $\mathbf{T}_{\text{CL}}$*bsf*). We can then promote both time series to the next finer resolution and repeat until we are working with the original resolution. Doing this, the total time is 18.5 seconds, which is essentially identical to the time needed for oracle search order (8.8 + 9.6 seconds).

The reader will appreciate that this idea works just as well for an arbitrary number of time series. In fact, we might expect it to work better as we see more time series, because as we see more time series, we increase the chance that at least one of them will have a very well conserved motif, and this will help pruning overall. In the next section we will formalize these intuitions.

A. iMOMP: interleaved MOMP

For simplicity we assume that all the time series are the same length. If that is not the case (as in Figure 18), we pad the empty slots with NaNs and strip them out before processing. We can solve MISTS with a sequential search, using any motif algorithm as the subroutine. While this is obvious, for concreteness we outline the algorithm in Table 8.

Table 8: Sequential Algorithm for MISTS

```
Function:     sMISTS(TS_SET,m)
Input:        TS_SET: Set of input time series
              m:  Subsequence length

Output:       top_1_motif: Top-1 motif
              top_1_index: T index including the best motif
1   top_1_motif, top_1_index = inf, NaN
2   for Tᵢ in TS_SET:
3      bsf_motif = compute_best_motif(Tᵢ, m) #MOMP or SCAMP
4      If bsf_motif < Top_1_motif:
5         top_1_motif, top_1_index = bsf_motif, i
```

[10] Some plant species have enormous mitochondrial genomes, with the Striped corn catchfly (*Silene conica*) mDNA containing as many as 11,300,000 base pairs. However, here we confine our interests to the data from [64].

In line 1, we initialize the Top-1 motif to infinity and set the time series containing the best motif to NaN. The main loop, starting in line 2, iterates over all the timeseries in the given set. In each iteration as in line 3, the best motif is computed for the target time series using any algorithm, such as MOMP or SCAMP. Line 4 compares the current best motif with the Top-1 motif. If a better motif is found, line 5 updates the Top-1 motif.

In Table 9 we outline iMOMP, or interleaved MOMP, an algorithm inspired by the observations in the last section.

Table 9: The iMOMP Algorithm

```
Function:      iMOMP(TS_SET,m)
Input:         TS_SET: Set of input time series
               m:  Subsequence length

Output:        top_1_motif: Top-1 motif
               top_1_index: T index including the best motif
```

```
1    top_1_motif, top_1_index = inf, NaN
2    BSF_SET = inf(timeseries_count ,1)
3    While dsr >= 1:
4       for T_i in TS_SET:
5          pruned_T_i, T_i_bsf = MOMP_STEP(T_i, m , dsr)
6          top_1_motif = min([top_1_motif, T_i_bsf])
7          top_1_index = i
8          update(TS_SET, pruned_T_i)
9       sort(TS_SET, 'ascending', 'bsf')
10      dsr = dsr /2
```

In line 1, we initialize the Top-1 motif and its corresponding time series. In line 2, we define a set of best-so-far motif values all set to infinite value. The main loop begins in line 3 and continues until the downsampling ratio reaches one. The inner loop starts in line 4, where we iterate over the time series in the set. Line 5 computes the MOMP step for the current downsampling ratio. A MOMP step refers to each iteration of the main loop in the MOMP algorithm (Table 2).

Lines 6 and 7 update the Top-1 motif value and its corresponding time series index. In line 8, the input set is updated with its pruned version. If a time series is fully pruned, line 8 removes it from the rest of computations. In line 9, the timeseries are sorted by their best-so-far motif values in ascending order, so the first time series now has the best motif. Finally, in line 10, the downsampling ratio is divided by a factor of 2 so we can repeat the process at the next finer data resolution.

B. *Emperical Evaluatiuon of iMOMP*

To evaluate iMOMP we will test on the plant mDNA data shown in Figure 18. We created a dataset by considering the ten longest plant mDNA time series from [64], sorted from shortest (704,100bp) to longest (1,999,595). We searched for the best motif of length 65,536.

We compare three approaches:

- Sequential search with SCAMP (Table 8).
- Sequential search with MOMP (Table 8).
- The proposed iMOMP algorithm (Table 9).

We evaluate the three approaches by measuring their (anytime) convergence over time, similar to the example shown in Figure 14. In Figure 20 we summarize the results.

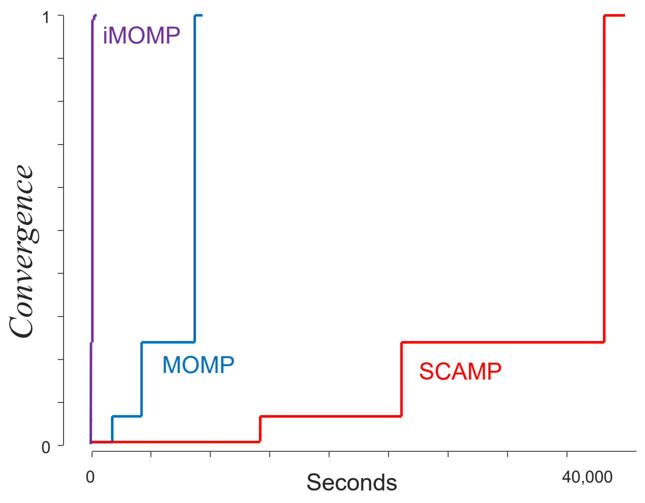


Figure 20: The convergence of three algorithms to solve the MISTS problem for the Plant-DNA dataset. The iMOMP algorithm converges so fast it might be mistaken for a vertical line when plotted at this scale. However, if you examine the top of the line carefully, you can see some short "steps" as it refines its answer.

Here SCAMP takes 12.4 hours. Simply replacing SCAMP with MOMP takes only 2.6 hours, however both algorithms slowly converge to the correct answer in a "stepwise" fashion. In contrast, iMOMP finishes in on 7.33 minutes, and converges very rapidly.

If we examine the first nine time series in the Plant-mDNA dataset, we find that the average motif distance is 71.2 with a standard deviation of 19.6. However, the best motif discovered, which happened to be the tenth place, has a distance of just 0.066, about 3.6 standard deviations less than the mean. This suggests that this motif is unusually well conserved, and Figure *21* visually confirms this.

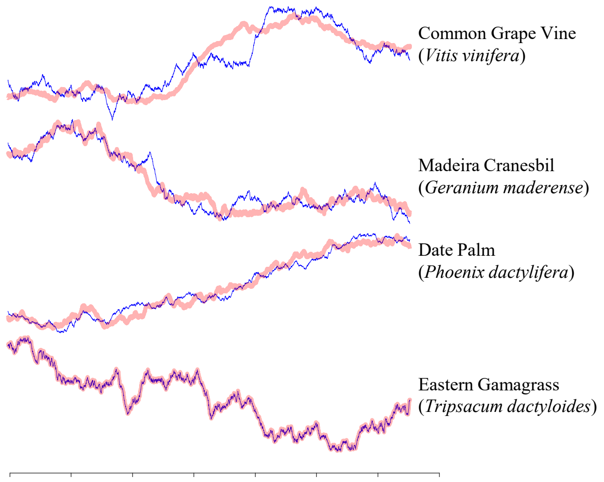


Figure 21:The The motifs discovered in the four longest Plant-mDNA datasets. Only the motif discovered in the Easten Gamagrass is conversed beyond chance level.

The plant in question is Eastern Gamagrass (*Tripsacum dactyloides*), and we find it was explicitly singled out for having long motifs in a paper titled "*Repeats of Unusual Size in Plant Mitochondrial Genomes: Identification, Incidence* *and Evolution*" (our emphasis) [65]. In Figure 21 we show the motifs discovered in the four longest Plant-mDNA datasets, the reader will appreciate that the discovered motif is visually much more conversed than we might expect by chance (although it is not perfectly conserved).

VIII. Related Work

We have relegated the discussion of related work to the end of our paper, so the reader has more context to appreciate our contributions. There has been a significant increase of interest in time series motif discovery in recent years.

The recent Attimo [3] is an *approximate* algorithms that uses hashing to attempt to find motifs. While *hashing* for time series motifs dates back to 2003 [51], Attimo offers probabilistic "guarantees", however:

- As with all probabilistic models, there are assumptions about the data. However it is not clear if these assumptions are *ever* true on real data, much less *generally* true.
- It is not clear how users can interpret, set or interact with the "probabilistic guarantees". Humans (that includes data scientists!) are notoriously bad with probabilities [52].
- However, the two above issues are moot, because as we show in [10], on some datasets it takes Attimo several orders of magnitude longer to converge on a "99%" solution than simply using SCAMP to compute to the 100% correct solution! Again, to be clear, the *approximate* algorithm can be much slower than the *exact* algorithm.
- It might be argued that Attimo has a place in finding approximate solutions in easy datasets. However, recall that the anytime property of the STAMP/ SCAMP/SCRIMP++ algorithms means that we *already* have faster converging (and parameter-free) algorithms for quickly finding approximate solutions!

While Attimo does not make any contribution time series motif discovery, it is to be commended for being an intellectually honest attempt at making a scientific contribution.

Motiflets [13] promises "*Simple and Accurate Detection of Motifs in Time Series*", however, since that paper's only correct claims are restatements of known results, it need not detain us here.

FRM-Miner vows "*our method is highly scalable*" [53], however the method can only find motifs in the pathological case where both occurrences that have the same mean and standard deviations *while within the larger dataset that the appear in*[11]. This is due to the unorthodox use of SAX to symbolize the *entire* data, not the *individual subsequences* (like [4][7] etc.). To see how brittle FRM-Miner is, consider the following. Suppose it did *find* a motif shared between two time series **A** and **B**, because it just happened to be conserved under *both* the z-normalized and non-normalized Euclidean distance. It is possible that you could add or delete a single point from the end or beginning of **A** or **B** (far away from the motif's location) that would change the non-normalized Euclidean distance of motifs, and thus make FRM-Miner fail to find the motif. Note that in these circumstances, any MP algorithm, which all operate in the z-normalized space, would still correctly report the motif. Thus FRM-Miner is an approximation to an approximation to a very limited subset of the expressiveness of classic motif discovery [15][16].

LoCoMotif[12] proports to find motifs "*faster than its competitors*" by creating recurrence matrix and mining the visual patterns therein. It also proports to find more expressive patterns [54]. However, this basic idea is at least twenty years old [55], and the Matrix Profile version of this [41][56] is O(m) times faster[13], and more expressive, being able to discover motifs of different lengths, rescaled motifs, occluded motifs, warped motifs, rare motifs etc. [56].

While there has been little progress in *finding* motifs in recent years, in contrast there has been significant progress in *using* motifs to solve interesting problems. For example, [57] uses motifs for the characterization of Dansgaard-Oeschger events in paleoclimate time series. In [58] motifs are used for bridge health monitoring. Paper [59] uses motifs to classify motions in quadruped robots. The detection of major depressive disorders is facilitated by motifs in [60]. In [62] the authors use motifs to study meiotic CMs in fission yeast. We look forward to seeing future creative uses of time series motifs.

IX. Conclusions

We introduced lbMP, the first lower bound to the Matrix Profile, and we further introduced MOMP, a parameter-free algorithm that exploits lbMP to accelerate *exact* motif discovery. We have shown that MOMP accelerates motif discovery by up to two orders of magnitude. We have further shown that it is possible to exploit MOMP to efficiently solve more general tasks, with the example of iMOMP for motif discovery within time series sets.

Moreover, even in the cases where MOMP does *not* produce any speed up over classic MP algorithms, we argue that it is still a desirable algorithm as it is hyper-anytime, and in 99.9% of real-world situations, we should be using anytime algorithms.

There are many opportunities for future work. In the past motif discovery was always compute-bound. The use of MOMP is bringing us close to the point were dealing with secondary memory will become an interesting research challenge. In addition, precedent in time series *similarity search*, suggests that once a lower-bounding framework for an important problem is introduced, the community is very creative in inventing increasingly tighter lower bounds [8].

[11] Note that we *plot* motifs after z-normalization in Figure 11, Figure 12, Figure 17 etc. to emphasis their shape similarity. However, in general they do not have the same mean and standard deviation in within the raw data.

[12] The paper ends by thanking "*Eamonn Keogh for providing valuable feedback*." To be clear, Keogh's modest contribution was to point out that most of their paper is just restating existing and (still) uncited work.

[13] Recall that m can be in the thousands or tens of thousands, so the speed up is significant. In fact, Mplots are so efficient that [56] creates and searches recurrence matrices that would cover several football fields if printed out.

ACKNOWLEDGEMENTS

We thank the donors of datasets, who in many cases were also kind enough to interpret the motifs. We thank Dr. Mason and his team for helping to interpret some of the medical motifs. Finally, we thank Kaveh Kamgar of the UCR Lab for producing and explaining the highly optimized SCAMP code used as the base algorithm in our work.

REFERENCES

[1] Nuttall, T., Plaja-Roglans, G., Pearson, L., Serra, X. (2023). The Matrix Profile for Motif Discovery in Audio - An Example Application in Carnatic Music. In: Aramaki, M., Hirata, K., Kitahara, T., Kronland-Martinet, R., Ystad, S. (eds) Music in the AI Era. CMMR 2021. Lecture Notes in Computer Science, vol 13770 . Springer, Cham.

[2] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is 'Nearest Neighbor' Meaningful?," *ICDT 1999. LNCS*, vol. 1540.

[3] M. Ceccarello and J. Gamper, "Fast and Scalable Mining of Time Series Motifs with Probabilistic Guarantees," *Proceedings of the VLDB Endowment*, vol. 15, pp. 3841–3853, May 2023.

[4] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in 2003, ACM SIGKDD, pp. 493-498.

[5] N. Dewan, F. Nieto, V. Somers. Intermittent hypoxemia and OSA: implications for comorbidities. Chest. 2015 Jan;147(1):266-274.

[6] Edanusa-X12 Tech Sheet., *edanusa.com/wp-content/uploads/sites/D12/2020/11/x8-x12-PM-spec-sheet.pdf*, Apr. 29, 2024.

[7] Y. Gao and J. Lin, "HIME: discovering variable-length motifs in large-scale time series," *Knowl Inf Syst*, vol. 61, May 2019.

[8] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Data Bases," in 2001, KAIS vol. 3, no. 3, pp. 263-286.

[9] Y. Li, et. al., "Quick-motif: An efficient and scalable framework for exact motif discovery" in *ICDE*,. 2015, pp. 579–90.

[10] MOMP, "https://sites.google.com/view/momp2024," Jun. 01, 2024.

[11] A. Olsson and L. Keeling, "Why in earth? Dustbathing behaviour in jungle and domestic fowl reviewed from a Animal Welfare perspective," *Appl Anim Behav Sci*, vol. 93, pp. 259–282, May 2005.

[12] A. Raoofy, R. Karlstetter, M. Schreiber, C. Trinitis, and M. Schulz, "Overcoming Weak Scaling Challenges in Tree-Based Nearest Neighbor Time Series Mining," 2023, pp. 317–338.

[13] P. Schäfer and U. Leser, "Motiflets: Simple and Accurate Detection of Motifs in Time Series," *Proceedings of the VLDB Endowment*, vol. 16, pp. 725–737, May 2023, doi: 10.14778/3574245.3574257.

[14] M. Suntsova, and A. Buzdin. Differences between human and chimpanzee genomes. BMC Genomics 21 (Suppl 7), 535 (2020).

[15] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, Eamonn J. Keogh: Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. ICDM 2016: 1317-1322.

[16] C.-C. M. Yeh *et al.*, "Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile," *Data Min Knowl Discov*, vol. 32, May 2018, doi: 10.1007/s10618-017-0519-9.

[17] Y. Zhu *et al.*, "Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins," in *2016 IEEE 16th ICDM*, 2016, pp. 739–748.

[18] Z. Zimmerman *et al.*, "Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond," in *Proceedings of the ACM SoCC*, 2019, pp. 74–86.

[19] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth J. Funning, Abdullah Mueen, Philip Brisk, Eamonn J. Keogh: Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. ICDM 2016: 739-748.

[20] Chin-Chia Michael Yeh, Helga Van Herle, Eamonn J. Keogh: Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series. ICDM 2016: 579-588.

[21] Chin-Chia Michael Yeh, Nickolas Kavantzas, Eamonn J. Keogh: Matrix Profile IV: Using Weakly Labeled Time Series to Predict Outcomes. Proc. VLDB Endow. 10(12): 1802-1812 (2017)

[22] Hoang Anh Dau, Eamonn J. Keogh: Matrix Profile V: A Generic Technique to Incorporate Domain Knowledge into Motif Discovery. KDD 2017: 125-134.

[23] Chin-Chia Michael Yeh, Nickolas Kavantzas, Eamonn J. Keogh: Matrix Profile VI: Meaningful Multidimensional Motif Discovery. ICDM 2017: 565-574

[24] Yan Zhu, Makoto Imamura, Daniel Nikovski, Eamonn J. Keogh: Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining (Best Student Paper Award). ICDM 2017: 695-704

[25] Shaghayegh Gharghabi, Yifei Ding, Chin-Chia Michael Yeh, Kaveh Kamgar, Liudmila Ulanova, Eamonn J. Keogh: Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels. ICDM 2017: 117-126.

[26] Yan Zhu, Abdullah Mueen, Eamonn J. Keogh: Matrix Profile IX: Admissible Time Series Motif Discovery With Missing Data. IEEE Trans. Knowl. Data Eng. 33(6): 2616-2626 (2021)

[27] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, Eamonn J. Keogh: Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds. ICDM 2018: 837-846.

[28] Shaghayegh Gharghabi, Shima Imani, Anthony J. Bagnall, Amirali Darvishzadeh, Eamonn J. Keogh: Matrix Profile XII: MPdist: A Novel Time Series Distance Measure to Allow Data Mining in More Challenging Scenarios. ICDM 2018: 965-970.

[29] Shima Imani, Frank Madrid, Wei Ding, Scott E. Crouter, Eamonn J. Keogh: Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining. ICBK 2018: 382-389.

[30] Kaveh Kamgar, Shaghayegh Gharghabi, Eamonn J. Keogh:Matrix Profile XV: Exploiting Time Series Consensus Motifs to Find Structure in Time Series Sets. ICDM 2019: 1156-1161

[31] Frank Madrid, Shailendra Singh, Quentin Chesnais, Kerry Mauck, Eamonn J. Keogh: Matrix Profile XVI: Efficient and Effective Labeling of Massive Time Series Archives. DSAA 2019: 463-472

[32] Zachary Zimmerman, Nader Shakibay Senobari, Gareth J. Funning, Evangelos E. Papalexakis, Samet Oymak, Philip Brisk, Eamonn J. Keogh: Matrix Profile XVIII: Time Series Mining in the Face of Fast Moving Streams using a Learned Approximate Matrix Profile. ICDM 2019: 936-945

[33] Shima Imani, Eamonn J. Keogh: Matrix Profile XIX: Time Series Semantic Motifs: A New Primitive for Finding Higher-Level Structure in Time Series. ICDM 2019: 329-338

[34] Frank Madrid, Shima Imani, Ryan Mercer, Zachary Zimmerman, Nader Shakibay Senobari, Eamonn J. Keogh: Matrix Profile XX: Finding and Visualizing Time Series Motifs of All Lengths using the Matrix Profile. ICBK 2019: 175-182

[35] Makoto Imamura, Takaaki Nakamura, Eamonn J. Keogh: Matrix Profile XXI: A Geometric Approach to Time Series Chains Improves Robustness. KDD 2020: 1114-1122

[36] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Eamonn J. Keogh: Matrix Profile XVII: Indexing the Matrix Profile to Allow Arbitrary Range Queries. ICDE 2020: 1846-1849

[37] Sara Alaee, Kaveh Kamgar, Eamonn J. Keogh: Matrix Profile XXII: Exact Discovery of Time Series Motifs under DTW. ICDM 2020: 900-905.

[38] Ryan Mercer, Sara Alaee, Alireza Abdoli, Shailendra Singh, Amy C. Murillo, Eamonn J. Keogh: Matrix Profile XXIII: Contrast Profile: A Novel Time Series Primitive that Allows Real World Classification. ICDM 2021: 1240-1245

[39] Yue Lu, Renjie Wu, Abdullah Mueen, Maria A. Zuluaga, Eamonn J. Keogh: Matrix Profile XXIV: Scaling Time Series Anomaly Detection to Trillions of Datapoints and Ultra-fast Arriving Data Streams. KDD 2022: 1173-1182.

[40] Ryan Mercer, Eamonn J. Keogh: Matrix Profile XXV: Introducing Novelets: A Primitive that Allows Online Detection of Emerging Behaviors in Time Series. ICDM 2022: 338-347.

[41] Maryam Shahcheraghi, Ryan Mercer, João Manuel De Almeida Rodrigues, Audrey Der, Hugo Filipe Silveira Gamboa, Zachary Zimmerman, Eamonn J. Keogh: Matrix Profile XXVI: Mplots: Scaling

Time Series Similarity Matrices to Massive Data. ICDM 2022: 1179-1184.

[42] Audrey Der, Chin-Chia Michael Yeh, Renjie Wu, Junpeng Wang, Yan Zheng, Zhongfang Zhuang, Liang Wang, Wei Zhang, Eamonn J. Keogh: Matrix Profile XXVII: A Novel Distance Measure for Comparing Long Time Series. ICKG 2022: 40-47.

[43] Sadaf Tafazoli and Eamonn Keogh: Matrix Profile XXVIII: Discovering Multi-Dimensional Time Series Anomalies with K of N Anomaly Detection. SIAM SDM 2023.

[44] Sadaf Tafazoli, Yue Lu, Renjie Wu, Thirumalai Vinjamoor Akhil Srinivas, Hannah Dela Cruz, Ryan Mercer, Eamonn J. Keogh:Matrix Profile XXIX: C22MP, Fusing catch 22 and the Matrix Profile to Produce an Efficient and Interpretable Anomaly Detector. ICDM 2023: 568-577

[45] Yue Lu, Thirumalai Vinjamoor Akhil Srinivas, Takaaki Nakamura, Makoto Imamura, Eamonn J. Keogh: Matrix Profile XXX: MADRID: A Hyper-Anytime and Parameter-Free Algorithm to Find Time Series Anomalies of all Lengths. ICDM 2023: 1199-1204

[46] Diego Furtado Silva, Chin-Chia Michael Yeh, Yan Zhu, Gustavo E. A. P. A. Batista, Eamonn J. Keogh: Fast Similarity Matrix Profile for Music Analysis and Exploration. IEEE Trans. Multim. 21(1): 29-38 (2019).

[47] Shabikay Senobari, N., Shearer, P. M., Funning, G. J., Zimmerman, Z., Zhu, Y., Brisk, P., & Keogh, E. (2024). The matrix profile in seismology: Template matching of everything with everything. Journal of Geophysical Research: Solid Earth, 129, e2023JB027122.

[48] Yan Zhu, Shaghayegh Gharghabi, Diego Furtado Silva, Hoang Anh Dau, Chin-Chia Michael Yeh, Nader Shakibay Senobari, Abdulaziz Almaslukh, Kaveh Kamgar, Zachary Zimmerman, Gareth J. Funning, Abdullah Mueen, Eamonn J. Keogh: The Swiss army knife of time series data mining: ten useful things you can do with the matrix profile and ten lines of code. Data Min. Knowl. Discov. 34(4): 949-979 (2020)

[49] Renjie Wu, Eamonn J. Keogh: Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. IEEE Trans. Knowl. Data Eng. 35(3): 2421-2429 (2023)

[50] Eamonn Keogh (2024). Problems with TSAD www.dropbox.com/scl/fi/cwduv5idkwx9ci328nfpy/Problems-with-Time-Series-Anomaly-Detection.pdf?rlkey=d9mnqw4tuayyjsplu0u1t7ugg&dl=0

[51] Bill Yuan-chi Chiu, Eamonn J. Keogh, Stefano Lonardi: Probabilistic discovery of time series motifs. KDD 2003: 493-498

[52] Tversky, A., & Kahneman, D. (1983). Extensional versus intuitive reasoning: The conjunction fallacy in probability judgment. Psychological Review, 90(4), 293–315.

[53] Rotman, S.J., Cule, B., Feremans, L.: Efficiently mining frequent representative motifs in large collections of time series. In: 2023 IEEE International Conference on Big Data (BigData), pp. 66–75 (2023)

[54] Van Wesenbeeck, D., Yurtman, A., Meert, W. *et al.* LoCoMotif: discovering time-warped motifs in time series. *Data Min Knowl Disc* (2024).

[55] Dragomir Yankov, Eamonn J. Keogh, Stefano Lonardi, Ada Wai-Chee Fu: Dot Plots for Time Series Analysis. ICTAI 2005: 159-168

[56] Shahcheraghi, M., Mercer, R., Rodrigues, J.M.d. et al. Introducing Mplots: scaling time series recurrence plots to massive datasets. J Big Data 11, 96 (2024).

[57] Barbosa, S., Silva, M. E., and Rousseau, D.-D.: Characterisation of Dansgaard-Oeschger events in palaeoclimate time series using the Matrix Profile, Nonlin. Processes Geophys. Discuss. [preprint], https://doi.org/10.5194/npg-2024-13, 2024.

[58] M. Makki Alamdari, An evolutionary vehicle scanning method for bridges based on time series segmentation and change point detection, Mechanical Systems and Signal Processing. Volume 210, 2024, 111173, ISSN 0888-3270

[59] C. Allred, J. Pusey and M. Harper, "Detecting Ballistic Motions in Quadruped Robots: A Boosted Tree Motif Classifier for Understanding Reinforcement Learning," *2023 Seventh IEEE International Conference on Robotic Computing (IRC)*, Laguna Hills, CA, USA, 2023, pp. 143-151,

[60] Tuuli Uudeberg, Juri Belikov, Laura Päeske, Hiie Hinrikus, Innar Liiv, Maie Bachmann, In-phase matrix profile: A novel method for the detection of major depressive disorder, Biomedical Signal Processing and Control, Volume 88, Part C, 2024,105378.

[61] Farrokh Bulsara (1981). Under Pressure www.youtube.com/watch?v=89DBhoia9XI&

[62] León-Periñán D, Fernández-Álvarez A. ChroMo, an Application for Unsupervised Analysis of Chromosome Movements in Meiosis. Cells. 2021 Aug 6;10(8):2013. doi: 10.3390/cells10082013. PMID: 34440781; PMCID: PMC8392469.

[63] https://www.mathworks.com/help/predmaint/ref/matrixprofile.html

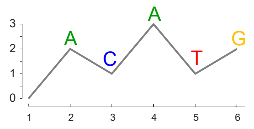
[64] Randhawa, G., Hill, K. & Kari, L. ML-DSP: Machine Learning with Digital Signal Processing for ultrafast, accurate, and scalable genome classification at all taxonomic levels. BMC Genomics 20, 267 (2019).

[65] Emily L Wynn, Alan C Christensen, Repeats of Unusual Size in Plant Mitochondrial Genomes: Identification, Incidence and Evolution, G3 Genes|Genomes|Genetics, Volume 9, Issue 2, 1 February 2019, Pages 549–559, https://doi.org/10.1534/g3.118.200948

Appendix A: DNA to Time Series

We use the classic algorithm below to perform a lossless transformation of DNA strings to time series.

```
$T_1 = 0$;
for i = 1 to length(DNAstring)
  If DNAstring$_i$ = A, then $T_{i+1} = T_i + 2$
  If DNAstring$_i$ = G, then $T_{i+1} = T_i + 1$
  If DNAstring$_i$ = C, then $T_{i+1} = T_i - 1$
  If DNAstring$_i$ = T, then $T_{i+1} = T_i - 2$
end
```



Appendix B: Predicting the time needed for SCAMP

SCAMP has the interesting property that we can predict exactly how long it will take given only *n*. The value of *m*, and data's other properties are completely irrelevant. This requires just one calibration run to learn an empirically determined constant **C**.

$$\text{TimeRequired}_n = n^2/\ \mathbf{C}\ , \quad \text{giving us:} \quad \mathbf{C} = n^2/\ \text{TimeRequired}_n$$

For concreteness, let us consider a worked example with a calibration run with the length of 131,072 or 2^{17}.

```
>> T=cumsum(randn(1,2^17));, tic; SCAMP(T, 100,100); , toc
Elapsed time is 60.9706 seconds.
```

How long would it take to compute the MP for a length of say, 333,333? Let us begin by computing **C** for our machine.

```
    C = (2^17)^2/ 60.9706 =  2.8177e+08
```

So, $\text{TimeRequired}_{333333} = (333333^2)/\ 2.8177\text{e}{+}08 =$ 394.3 seconds. How good was our prediction?

```
>> T=cumsum(randn(1,333333));, tic; SCAMP(T,100,100); , toc
Elapsed time is 392.9724 seconds.
```

It was almost perfect. Thus, although we must give extrapolated times for SCAMP with the larger datasets we compare to in this work, those times are within 1% of the true value.