Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds

Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar and Eamonn Keogh
Department of Computer Science and Engineering
University of California, Riverside
{yzhu015, myeh003, zzimm001, kkamg001}@ucr.edu, eamonn@cs.ucr.edu

***Abstract*— Time series motif discovery is an important primitive for time series analytics, and is used in domains as diverse as neuroscience, music and sports analytics. In recent years, algorithmic advances (coupled with hardware improvements) have greatly expanded the purview of motif discovery. Nevertheless, we argue that there is an insatiable need for further scalability. This is because more than most types of analytics, motif discovery benefits from interactivity. The two state-of-the-art algorithms to find motifs are STOMP, which requires O(n^2) time, and STAMP, which, despite being an O(log$n$) factor slower, is the preferred solution for most applications, as it is a fast converging anytime algorithm. In favorable scenarios STAMP needs only to be run to a small fraction of completion to provide a very accurate approximation of the top-k motifs. In this work we introduce SCRIMP++, an O(n^2) time algorithm that is *also* an anytime algorithm, combining the best features of STOMP and STAMP. As we shall show, SCRIMP++ maintains all the desirable properties of the original algorithms, but converges much faster, in almost all scenarios producing the correct output after spending a tiny fraction of the full computation time. We argue that for many end-users, this allows motif discovery to be performed in interactive sessions. Moreover, this interactivity can be game changing in terms of the analytics that can be performed.**

Keywords*— *Time Series, Anytime Algorithms, Motif Discovery

I. Introduction

In domains as diverse as seismology [16], neuroscience [1][2] and entomology [9], time series motif discovery is emerging as one of the most important analytic primitives. The recently introduced *Matrix Profile* has been shown to be a flexible and generic data tool to solve a host of time series data mining problems, including motif discovery. There are two algorithms to compute the Matrix Profile, STOMP [16], which requires O(n^2) time, and STAMP [15], which is an O(log$n$) factor slower. In spite of being slower, STAMP is actually the preferred solution for most applications, as it is a fast converging *anytime* algorithm, and in favorable scenarios it needs only to be run to about 5% of completion to provide a very accurate approximation of the top-k motifs [15]. Note that at first blush, the O(n^2log$n$) STAMP algorithm may not seem that scalable, even if run to only 5% of completion. However, both STAMP/STOMP have time and space complexities that are independent of the dimensionality, the *length* of the motifs, m. Because this length can be in the many thousands, in practice O(n^2) is significantly faster than *apparently* faster algorithms that scale with m [15][16].

In this work we introduce SCRIMP++, an O(n^2) time algorithm that is *also* an anytime algorithm, combining the best features of STOMP and STAMP. As we shall show, SCRIMP++ maintains all the desirable properties of the original algorithms, including invariance to the curse of dimensionality, a low memory footprint and the ability to trivially exploit High Performance Computing platforms such as GPUs. We will demonstrate that SCRIMP++ further expands the purview of the Matrix Profile and allows us to consider even larger datasets. More critically however, SCRIMP++ allows us to perform motif discovery *interactively*, rather than the typical offline *batch* processing that is the norm [2].

Fig. 1 outlines the relationship between all these algorithms. While it is a symbolic plot, we will show empirically in Section IV.A that these relationships are correct. Note that we have been deliberately vague about what the Y-axis represents. It could be 1 minus the difference between the true Matrix Profile and the current estimate (as measured by the root-mean-squared error), or it could be the probability that the best motif has been discovered, or essentially any quality measure of the approximate Matrix Profile.

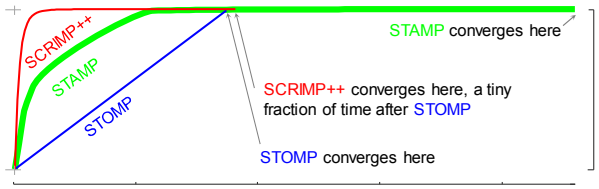

Fig. 1 The convergence behaviors of algorithms for computing the Matrix Profile (cf. Fig. 14). STOMP (not a true anytime algorithm) converges in O(n^2) time. STAMP converges faster, but takes O(log$n$) more time to completely converge. SCRIMP++ has both O(n^2) time complexity and diminishing returns convergence. Note that SCRIMP++ takes very slightly more time than STOMP to fully converge, but this difference is inconsequential.

A. Motif Analytics: An Insatiable Need for Speed

While all data mining algorithms benefit from improvements in speed, here we argue that for the particular case of motif discovery, improvements in speed are game-changing. Motif discovery benefits from *interactivity* more than most data mining processes. To see this, consider the following analytics session scenario, which while slightly fictionalized, is based on an ongoing project supporting data-intensive entomology [14].

An entomologist wants to examine a five-hour, 1,080,000-point time series (as shown in Fig. 2) she recorded overnight. From her previous experience, she suspects that a motif length of 100, corresponding to one-second, is about the right scale for this insect's behavior to be manifest. However, because she notices the motifs discovered are so well conserved at this scale, she decides to consider two-second long motifs. When she sees these new motifs, she realizes that they correspond to snippets from the *setup* time, when her assistant was adjusting the conductive glue on the insect's back. She therefore crops off the first few minutes and runs motif discovery again. She then…

Fig. 2. A five-hour sample of Electrical Penetration Graph (EPG) data hints at the difficulty of motif search. See also Fig. 17/Fig. 18.

If the entomologist was to use STOMP [16], the state-of-the-art exact motif discovery algorithm[1], then on a modern desktop each run would take about 0.7 hours. This is an important data resource, and a diligent entomologist may find it worth the effort to visit her machine every hour or so, but clearly such long cycle time dashes any hope of interactively. As [11] notes *"In interactive data analysis processes, the dialogue between the human and the computer is the enabling mechanism that can lead to actionable observations. It is of paramount importance that this dialogue is not interrupted by slow computation*".

As we will show in this work, SCRIMP++ allows us to perform the above analytic workflow *interactively*; in the above scenario, we can reduce the cycle time to just a few seconds.

Beyond the above anecdote that reflects *our* research interests, the literature is replete with examples that suggest the need for faster motif discovery. A recent paper considering several fundamental questions in neuroscience notes that some such questions reduce to determining if neural activity "repeats" happen more than expected by chance [2]. As Fig. 3 suggests, these *repeats* are simply time series motifs.

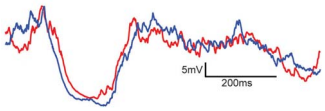

Fig. 3. Adapted from [2]. "Repeats" in the neuroscience literature are simply time series motifs.

To find such motifs in even a minute's worth of data, the authors resorted to various approximations to "*increase processing speed*." For example, they downsampled their data by 1 in 10, and rather than use a *sliding* window, they use a "*jumping*" window to reduce the number of comparisons. Even then, the authors noted that to obtain timely answers their "*repeat-finding algorithm was parallelized and performed on a high-performance computing* (HPC) *cluster*." [2].

However, consider their 2-kHz data, and further assume that we search for their longest motif length of 2.7 seconds (5,400 datapoints), and test *all* possible subsequences (not just "jumping" overlaps) in their largest dataset, which is 8,258,064 data points corresponding to 68.8 minutes of wall clock time.

With an off-the-shelf desktop we can run SCRIMP++ to 1%, in 27.4 minutes, and reproduce their quality of results (cf. [18]). Note that even here, with the original authors' most challenging task, we can still process the data *faster than they can collect it* [2]. The authors go on to bemoan the fact that even with their approximations and use of HPC, that their findings "*represent a lower limit on the duration and prevalence of motifs which might be observed if longer segments of intracellular dynamics could be analyzed*". The algorithm presented in this paper will trivially allow this possibility to be explored, not with batch processing on an HPC, but in real-time interactive sessions on a laptop.

[1] We justify this claim in Section II.C.

Before moving on, we note that the Matrix Profile has implication for other time series tasks, including discord discovery [15], chain discovery [17], semantic segmentation [15], etc. While SCRIMP++ can benefit these tasks, for simplicity and concreteness we only consider motif discovery in this work.

We conclude this section with a statement of contributions:

- Users of the Matrix Profile must currently choose between the *batch* $O(n^2)$ STOMP algorithm, or the *anytime* $O(n^2 \log n)$ STAMP algorithm. We introduce SCRIMP, the first "best of both worlds" algorithm for computing the Matrix Profile that is *both* anytime and $O(n^2)$.
- We introduce PreSCRIMP, a novel ultrafast *approximate* algorithm to compute the Matrix Profile. The output of PreSCRIMP can be seamlessly passed to SCRIMP, which can take the *approximate* solution and refine it until it is *exact*. This combination, of PreSCRIMP and SCRIMP is called SCRIMP++, and it represents the state-of-art in motif discovery.
- We show that SCRIMP++ is "game changing" by vastly increasing the space of problems that can be processed *interactively*.

The rest of this paper is organized as follows. In Section II we introduce definitions and consider related work. In Section III we introduce SCRIMP++. Section IV sees an extensive empirical evaluation of the algorithms, including some case studies. Finally, in Section V we offer conclusions and directions for future work.

II. RELATED WORK AND BACKGROUND

In this section, we first introduce all necessary definitions before considering related work.

A. Definitions

We begin by defining the data type of interest, *time series*:

Definition 1: A time series T is a sequence of real-valued numbers t_i: $T = t_1, t_2, ..., t_n$ where n is the length of T:

We are typically interested not in *global*, but *local* properties of a time series. A local region of a time series is called a *subsequence*:

Definition 2: A subsequence $T_{i,m}$ of a time series T is a continuous subset of the values from T of length m starting from position i. Formally, $T_{i,m} = t_i, t_{i+1}, \ldots, t_{i+m-1}$, where $1 \leq i \leq n\text{-}m+1$.

Given a query subsequence $T_{i,m}$ and a time series T, we can compute the distance between $T_{i,m}$ and *all* the subsequences in T. We call this a *distance profile*:

Definition 3: A *distance profile* D_i corresponding to query $T_{i,m}$ and time series T is a vector of the Euclidean distances between a given query subsequence $T_{i,m}$ and each subsequence in time series T. Formally, $D_i = [d_{i,1}, d_{i,2}, ..., d_{i,n-m+1}]$, where $d_{i,j}$ $(1 \leq j \leq n\text{-}m+1)$ is the distance between $T_{i,m}$ and $T_{j,m}$.

We assume that the distance is measured by Euclidean distance between z-normalized subsequences [12]. Once we obtain D_i, we can extract the nearest neighbor of $T_{i,m}$ in T. Note that if the query $T_{i,m}$ is a subsequence of T, the i^{th} location of distance profile D_i is zero (i.e., $d_{i,i} = 0$) and close to zero just to the left and right of i. This is called *trivial match* in the literature. We avoid such matches by ignoring an "exclusion" zone of length $m/4$ before and after i, the location of the query. In practice, we simply set $d_{i,j}$ ($i-m/4 \leq j \leq i+m/4$) to infinity, and the nearest neighbor of $T_{i,m}$ can thus be found by evaluating min(D_i).

We wish to find the nearest neighbor of every subsequence in T. The nearest neighbor information is stored in two meta time series, the *matrix profile* and the *matrix profile index*:

Definition 4: A *matrix profile* P of time series T is a vector of the Euclidean distances between every subsequence of T and its nearest neighbor in T. Formally, $P = [\min(D_1), \min(D_2), \ldots, \min(D_{n-m+1})]$, where D_i ($1 \leq i \leq n-m+1$) is the distance profile D_i corresponding to query $T_{i,m}$ and time series T.

The i^{th} element in the matrix profile P tells us the Euclidean distance from subsequence $T_{i,m}$ to its nearest neighbor in time series T. However, it does not tell us the *location* of that nearest neighbor; this is stored in the companion *matrix profile index*:

Definition 5: A *matrix profile index* I of time series T is a vector of integers: $I=[I_1, I_2, \ldots I_{n-m+1}]$, where $I_i=j$ if $d_{i,j} = \min(D_i)$.

Fig. 4 illustrates the relationship between distance matrix, distance profile (Definition 3) and matrix profile (Definition 4). Each element of the distance matrix $d_{i,j}$ is the distance between $T_{i,m}$ and $T_{j,m}$ ($1 \leq i, j \leq n-m+1$) of time series T.

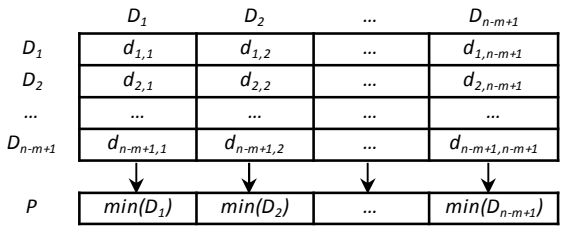

Fig. 4. The relationship between the distance matrix, distance profile and matrix profile. A distance profile is a column (also a row) of the distance matrix. The matrix profile stores the minimum (off diagonal) value of each column of the distance matrix; the location of the minimum value within each column is stored in the companion matrix profile index.

Fig. 5 shows a visual example of a distance profile and a matrix profile created from the same time series T.

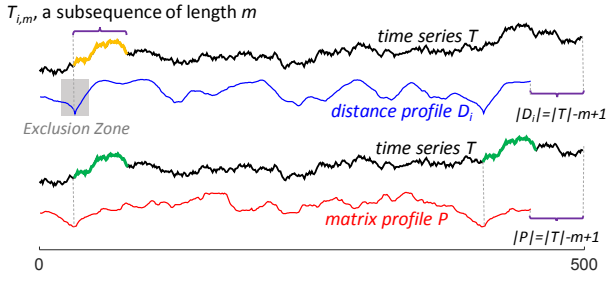

Fig. 5. *top*) A distance profile D_i created from $T_{i,m}$ shows the distance between $T_{i,m}$ and all the subsequences in T. The values in the dark zone are ignored to avoid trivial matches. *bottom*) The matrix profile P is the element-wise minimum of all the distance profiles (D_i is one of them). Note that the two lowest values in P are at the location of the 1st motif in T [15][16].

Note that as we presented it, the matrix profile is a *self-join* [15]: for every subsequence in a time series T, it records information about its (non-trivial-match) nearest neighbor in the same time series T. However, we can trivially generalize it to be an AB-join [15]; for every subsequence in a time series A, record information about its nearest neighbor in time series B. Note that A and B can be of different lengths, and that in general, AB-join ≠ BA-join.

B. Related Work: Matrix Profile Based

In a flurry of recent papers [15][16], it has been shown that one can trivially compute all top-k motifs (for any k), range motifs (for arbitrary ranges), and a host of other useful time series primitives [17], if one has access to the *matrix profile*. Thus, fast motif discovery simply reduces to fast computation of the matrix profile.

To date there are two algorithms to compute the matrix profile, STAMP [15] and STOMP [16].

The STAMP algorithm [15] evaluates the distance profiles (Definition 3; the columns/rows in Fig. 4) in random order. Each distance profile D_i is evaluated by the MASS algorithm [5], which exploits Fast Fourier Transform (FFT) to calculate the dot product between $T_{i,m}$ and every subsequence in T. The evaluation of a distance profile thus takes O($n$log$n$) time where n is the length of time series T, and the overall process takes O(n^2log$n$) time.

In contrast to STAMP, the STOMP algorithm [16] evaluates the distance profiles in Fig. 4 in-order by exploiting the computation dependency between consecutive distance profiles. The algorithm only costs O(n^2) time, an O(log$n$) factor faster than STAMP. STOMP algorithm was forcefully demonstrated as more efficient than the previous state-of-the-art motif discovery algorithms, the Quick-Motif algorithm [4] and the MK algorithm [6] in both time and space [16].

Both STAMP and STOMP maintain the element-wise minimum-so-far values of the evaluated distance profiles in a running matrix profile. Note that although STAMP is an O(log$n$) factor slower than STOMP, it shows better interactivity. As shown in Fig. 6, STAMP is able to locate the highlighted motifs in the time series T when it is only 10% completed, as the running matrix profile already contains two deep valleys at the vicinity of the motifs. In contrast, STOMP cannot locate the motifs even when it is 50% completed (no deep valleys show up), because the running matrix profile converges to the oracle from left to right in order.

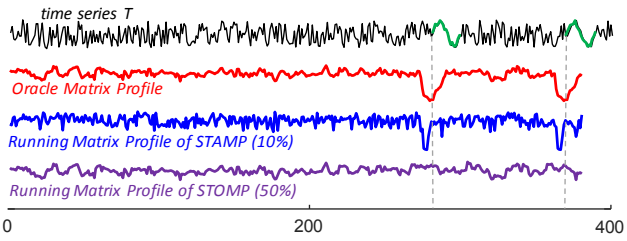

Fig. 6. STAMP is able to detect the motifs located towards the right side of a time series when it is only 10% completed due to its random computation order. In contrast, STOMP's left-to-right sequential computation means it cannot detect them even when 50% completed.

However, when the time series is very long and motifs are rare, the probability of STAMP finding the top-k motifs within 10% of its computation greatly decreases. Furthermore, as STOMP is a factor of O(log$n$) faster, by the time STAMP has completed 10% of its computation, STOMP may already converge to the exact solution. These conflicting strengths of the two algorithms require careful reasoning by the analyst, based on her goals and her tentative knowledge of the data. SCRIMP++ eliminates any dilemma, by combining the speed of STOMP with the anytime convergence property of STAMP.

C. Related Work: General Motif Search

It is important to make the distinction between *approximate* algorithms (of which there are many, see [10] for a survey) and *anytime* algorithms for motif discovery [15]. Suppose a user runs a fast, but approximate algorithm on a large dataset. It is possible that when the motifs are returned, she is satisfied. However, suppose that the motifs are *not* as well conserved as she expected, given her domain knowledge and her intuitions for the data. She is now in a quandary, are the expected motifs simply not there, or did the algorithm fail to find them? The problem is compounded by the fact that no approximate motif discovery algorithm we are aware of come with any kind of probabilistic guarantees, and all require at least three unintuitive parameters to be set [10]. What can our user do? If the approximate algorithm was stochastic, she can run it again, and/or change the parameters, but she may repeatedly face the same problem. Otherwise, she is condemned to run the fastest *exact* algorithm she has access to (which is STOMP [16]).

If the approximate algorithms took a tiny fraction of the time of the best exact algorithm, this issue would require some careful reasoning about trade-offs. However, as we will show in Section IV, all approximate algorithms take a large fraction of the time needed by SCRIMP++, especially for longer motifs.

For this reason, we argue that an *anytime* algorithm is necessary. In most cases, in a few seconds the user has acceptable results. If she has any doubts, she can simply let the algorithm run a little longer. There is no need to start the fastest *exact* algorithm, because it is already running!

Finally, we need to qualify the claim that STOMP is the fastest exact algorithm for motif discovery. On "cooperative data" (relatively smooth data, motifs highly conserved relative to the rest of the data, short motif lengths etc.), other exact algorithms such as Quick-Motif [4], IMD [3], or MK [6] *can* be fast. But in less-than-cooperative data (e.g., seismology data [16]) these algorithms degenerate to O(n^2m), with very high constant factors. The authors of [3] are to be commended for stating this explicitly "…*in the worst case, the algorithm still has a time complexity of* O(n^2m)".

As we show in our case studies (see Fig. 17), m can be as large as 15,000 or greater for real-world problems. In contrast STOMP (and SCRIMP++) takes O(n^2) time, completely independent of the data and the value of m. Thus, for realistic problems with high dimensionality, STOMP can be thousands of times faster than Quick-Motif [4], IMD [3], or MK [6].

III. ALGORITHMS

The SCRIMP++ Algorithm consists of two parts: PreSCRIMP and SCRIMP (as shown in Fig. 7). In this section, we will first introduce the SCRIMP algorithm, which is an O(n^2) anytime algorithm with better convergence characteristics than STOMP [16]. We will then further extend SCRIMP to SCRIMP++, a robust anytime algorithm which, thanks to the addition of an ultra-fast preprocessing algorithm PreSCRIMP, is capable of detecting essentially all the motifs within a time series at an early stage, even when the motifs are subtle and/or extremely rare. For simplicity we only consider self-join here; however, all the algorithms introduced can be easily extended to AB-join [15].

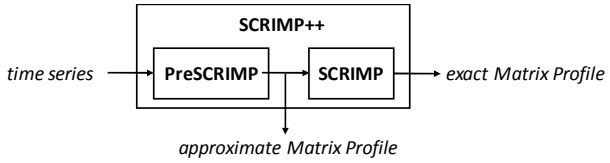

Fig. 7. The SCRIMP++ algorithm consists of an ultra-fast preprocessing algorithm, PreSCRIMP, and an O(n^2) anytime algorithm, SCRIMP. PreSCRIMP provides a very accurate approximation of the matrix profile at an early stage; SCRIMP further refines the approximate matrix profile until it becomes the exact/final solution. The user can interupt the algorithm at any time (during either PreSCRIMP or SCRIMP) to inspect the current approximate solution. Thus overall, SCRIMP++ is also an anytime algorithm.

A. Our Initial Solution: The SCRIMP Algorithm

Before we introduce the SCRIMP algorithm, let us first review the basics of the STOMP algorithm [16].

The z-normalized Euclidean distance $d_{i,j}$ of two time series subsequences $T_{i,\mathrm{m}}$ and $T_{j,m}$ can be evaluated as follows:

$$d_{i,j} = \sqrt{2m\left(1 - \frac{Q_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j}\right)} \quad (1)$$

Here m is the subsequence length, $Q_{i,j}$ is the dot product of $T_{i,m}$ and $T_{j,m}$, μ_i is the mean of $T_{i,m}$, μ_j is the mean of $T_{j,\mathrm{m}}$, σ_i is the standard deviation of $T_{i,m}$, and σ_j is the standard deviation of $T_{j,\mathrm{m}}$.

We can precompute the means and standard deviations for all subsequences in the time series in O(n) time by applying the technique introduced in [7]. Once that is done, the means and standard deviations in (1) can all be obtained in O(*1*) time. Furthermore, it is demonstrated in [16] that $Q_{i,j}$ can also be evaluated in O(*1*) time once $Q_{i-1,j-1}$ is given:

$$Q_{i,j} = Q_{i-1,j-1} - t_{i-1}t_{j-1} + t_{i+m-1}t_{j+m-1} \quad (2)$$

Based on (1) and (2), the STOMP algorithm [16] evaluates the distance matrix in Fig. 4 row-by-row in-order and updates the matrix profile accordingly, rendering an O(n^2) time complexity. However, as indicated in Fig. 6, this in-order computation prevents motifs at the end of a time series from being discovered at an early stage. Can we fix this undesirable property?

Note that (2) also implies that we can evaluate the *diagonals* of the distance matrix in Fig. 4 in random order. The SCRIMP algorithm (Algorithm 1) exploits this, evaluating the matrix

profile in an anytime fashion while keeping the same $O(n^2)$ time complexity.

Algorithm 1: ***The SCRIMP Algorithm***

Input: *A time series T and a subsequence length m*
Output: *Matrix profile P and matrix profile index I of T*

```
1  n ← Length(T)
2  μ, σ ← ComputeMeanStd(T, m) // see [7]
3  P ← infs, I ← ones     // initialization
4  Orders ← RandPerm(m/4+1 : n-m+1) // randomize evaluation order
5  for k in Orders        //evaluating diagonals in random order
6     for i ← 1 to n-m+2-k
7        if i=1 do q ← DotProduct(T_{1,m}, T_{k,m})
8        else q ← q - t_{i-1} t_{i+k-2} + t_{i+m-1} t_{i+k+m-2}   // see (2)
9        end if
10       d ← CalculateDistance(q, μ_i, σ_i, μ_{i+k-1}, σ_{i+k-1}) // see (1)
11       if d < P_i do P_i ← d, I_i ← i+k-1 end if
12       if d < P_{i+k-1} do P_{i+k-1} ← d, I_{i+k-1} ← i end if
13    end for
14 end for
15 return P, I
```

Line 2 precomputes the means and standard deviations of all subsequences in T. The matrix profile P and matrix profile index I are initialized in line 3. In lines 5-14, we iteratively evaluate the diagonals of the distance matrix in Fig. 4 in random order. Fig. 8 visualizes this. The distance values $d_{1,k}$, $d_{2,k}$, …, $d_{n-m+2-k,n-m+1}$ are calculated one by one; if $d_{i,i+k-1}$ (denoted as d in line 10, $1 \le i \le n-m+2-k$) is smaller than P_i (line 11) or P_{i+k-1} (line 12), we update the corresponding matrix profile (and index) values. At any time, the user can interrupt the algorithm to inspect the current P and I.

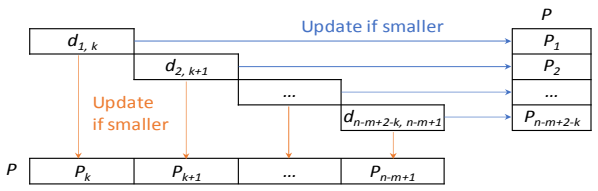

Fig. 8. A single iteration of SCRIMP evaluates a randomly selected diagonal in Fig. 4, thus updating the matrix profile in an *anytime* fashion.

B. Limitations of the SCRIMP Algorithm

As motifs in the time series correspond to the minimum points of the *oracle* (or *exact*) matrix profile (indicated in Fig. 9.*top*), we hope that SCRIMP could "focus" on these minimum points rather than at other locations. This has an element of a *chicken-and-egg* paradox to it, we want the algorithm to focus on where the motifs are, but we are using the algorithm to discover where the motifs are.

Recall that in each iteration of SCRIMP (as shown in Fig. 8), we evaluate a random diagonal of the distance matrix. To locate the motifs of time series T in Fig. 9.*top*, we need to evaluate the diagonal starting from $d_{1,126}$ (126-1=137-12) as early as possible. As shown in Fig. 9.*middle*, if SCRIMP evaluates that diagonal in its first iteration, the running matrix profile already overlaps perfectly with the oracle at the minimum points. However, if SCRIMP does not evaluate that diagonal until its very last iteration (Fig. 9.*bottom* shows the running matrix profile before the last iteration), we need to wait until the algorithm is 100% completed to locate the motifs. In fact, the probability to evaluate the diagonal of $d_{1,126}$ before the kth iteration is $k/(n-m+1)$. While SCRIMP has a chance to find the motif early no matter where they are located (which is its advantage over STOMP), that probability is not high.

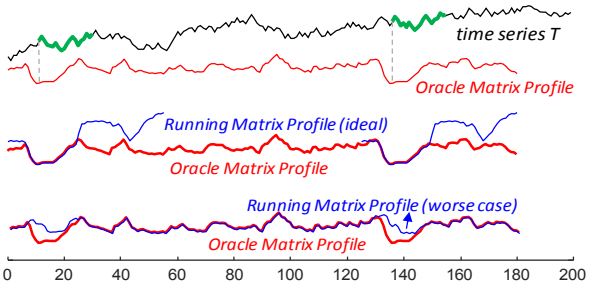

Fig. 9. *top)* Motifs (highlighted, located at 12 and 137) correspond to the minimum values of the matrix profile. *middle)* Ideally, SCRIMP can locate the motifs after its first iteration. *bottom)* In the pathological worst case, SCRIMP cannot locate the motifs until fully completed.

However, note that Fig. 9.*top* shows the hardest possible scenario for motif discovery; there is only a single pair of motifs in the time series. When the data contain more motifs, SCRIMP will perform much better. This is much like how the famous birthday paradox has an unexpectedly fast converge to probability 1 as we consider more individuals. The chance of SCRIMP making an early discovery of some pair from a motif set, increases dramatically if there are more members in that motif set. In the next section, we will introduce SCRIMP++, an extended version of SCRIMP which has a much higher probability of discovering not *some*, but *all* the true motifs at an early stage, even when the motifs are very rare.

C. Our Ultimate Solution: The SCRIMP++ Algorithm

The SCRIMP++ Algorithm is simply the SCRIMP algorithm (Algorithm 1) augmented by an additional preprocessing stage called PreSCRIMP (recall Fig. 7). We begin by introducing the Consecutive Neighborhood Preserving Property of time series subsequences, upon which PreSCRIMP is based.

Let us reexamine the matrix profile index of the example time series T in Fig. 9.*top*. Fig. 10 shows its first 25 entries.

Index	1	2	3	4	…	7	8	9	…	24	25	…
I	56	57	112	113	…	116	133	134	…	149	150	…

Fig. 10. The matrix profile index of T.

Here *Index* = [1, 2, 3, …, n-m+1] is the locations of all the subsequences in T, I is the matrix profile index (Definition 5) of T. We can see that the matrix profile index can be divided into multiple sections of consecutive values: within each section, a set of consecutive subsequences find another set of consecutive subsequences as their nearest neighbors. We call this the *Consecutive Neighborhood Preserving (CNP) Property* of time series subsequences.

With a little introspection, one can see that the CNP property *should* exist: since consecutive subsequences overlap by a large portion, if the i^{th} subsequence is very similar to the j^{th} subsequence, then there is a very high probability that the $(i+1)^{th}$

subsequence is also very similar to the $(j+1)^{th}$ subsequence. In Fig. 11, we can see that the 11th, 12th, 13th, and 14th subsequences find the 136th, 137th, 138th and 139th subsequences as their nearest neighbors, respectively; the subsequence-neighbor pairs remain a constant location difference of 125.

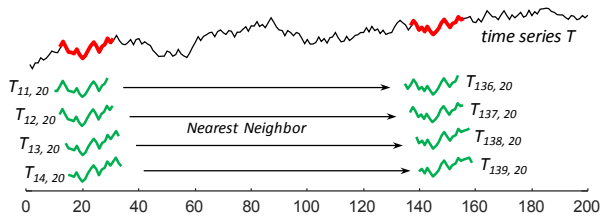

Fig. 11. Visualizing the CNP property of time series subsequences at the vicinity of the 1st motif pattern.

Exploiting the CNP property, we propose a preprocessing algorithm *PreSCRIMP*, that produces a very close approximation of the oracle matrix profile while costing only a tiny fraction of its $O(n^2)$ computation time. Essentially, we sample subsequences from the time series with a fixed interval s (Fig. 12.*top* shows the starting locations of these sampled subsequences). For each sampled subsequence, we find its exact nearest neighbor. Assume that $T_{i,\mathrm{m}}$ is a sampled subsequence, and its nearest neighbor is $T_{j,m}$, then according to the CNP property, there is a high probability that the nearest neighbor of $T_{i+k,\mathrm{m}}$ is $T_{j+k,m}$ (k=-s+1, -s+2, …, -2, -1, 1, 2, …, s-2, s-1). We compute the distances between these pairs of subsequences and update the matrix profile if a smaller distance value shows up.

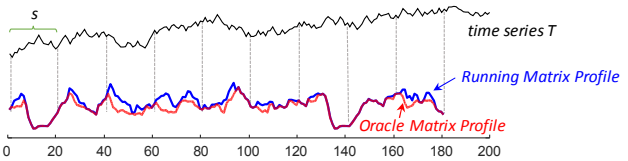

Fig. 12. *top*) Subsequences are sampled from time series T with a fixed interval s. *bottom*) After running PreSCRIMP, the running matrix profile becomes *very* similar to the oracle matrix profile, especially at the low values we care about.

The overall algorithm is outlined in **Algorithm 2**. Line 2 precomputes the means and standard deviations of all subsequences in T. In line 3, we sample subsequences from time series T with a fixed interval s (Fig. 12.*top* shows their starting position), then process these subsequences in random order. Each sample subsequence is processed with two stages (lines 4-22).

In the first stage (lines 4-7), we evaluate the distance profile corresponding to the current sample subsequence $T_{i,m}$ with the MASS algorithm [5], then update the running matrix profile (and index) if we find a smaller distance value. Note that after this stage, we already know the nearest neighbor of $T_{i,m}$ (assume it is $T_{j,m}$), and the matrix profile and matrix profile index are exact at the ith entry. As a result, we can see from Fig. 12.*bottom* that the running matrix profile aligns perfectly with the oracle matrix profile at the sampled locations.

In the second stage (lines 8-22), we refine the running matrix profile (and index) near the i^{th} entry by exploiting the CNP property. Starting from the current sample subsequence $T_{i,m}$ and its nearest neighbor $T_{j,m}$, we move forward to evaluate the pairwise distances between ($T_{i+1,m}$, $T_{j+1,m}$), ($T_{i+2,m}$, $T_{j+2,m}$), …, until we reach the next sampled location or the end of the time series (lines 10-15). After that, we traverse backward from $T_{i,m}$ and $T_{j,m}$ to evaluate the pairwise distance between ($T_{i-1,m}$, $T_{j-1,m}$), ($T_{i-2,m}$, $T_{j-2,m}$), …, until we reach an earlier sampled location or the beginning of the time series (lines 17-22). The corresponding running matrix profile (and index) entries are updated once we find a smaller distance value.

Algorithm 2: *The PreSCRIMP Algorithm*

Input: *A time series T, a subsequence length m and a sampling interval s.*

Output: *The running matrix profile P and matrix profile index I of T*

```
1  n ← Length(T), P← infs, I← ones // initialization
2  μ, σ ← ComputeMeanStd(T, m)  // precomputation, see [7]
3  for i ← RandPerm( 1 : s : (n-m+1) ) do  //sampling with interval s
4    seq ← T_{i,m}  //obtain a sample subsequence
5    D ← MASS (T, seq)  // evaluate a distance profile, see [5]
6    P, I ← ElementWiseMin (D, P, i)
7    P_i , I_i ← min(D)
8    j ← I_i  // the nearest neighbor of the sample subsequence
9    q ← CalculateDotProduct (P_i, μ_i, σ_i, μ_j, σ_j), q' ← q  // see (1)
10   for k ← 1 to min (s-1, n-m+1- max(i,j) ) do
11     q ← q - t_{i+k-1} t_{j+k-1} + t_{i+k+m-1} t_{j+k+m-1}  // see (2)
12     d ← CalculateDistance (q, μ_{i+k}, σ_{i+k}, μ_{j+k}, σ_{j+k}) // see (1)
13     if d < P_{i+k} do P_{i+k} ← d, I_{i+k} ← j+k end if
14     if d < P_{j+k} do P_{j+k} ← d, I_{j+k} ← i+k end if
15   end for
16   q ← q'
17   for k ← 1 to min(s-1, i-1, j-1) do
18     q ← q - t_{i-k+m} t_{j-k+m} + t_{i-k} t_{j-k}  // see (2)
19     d ← CalculateDistance (q, μ_{i-k}, σ_{i-k}, μ_{j-k}, σ_{j-k}) // see (1)
20     if d < P_{i-k} do P_{i-k} ← d, I_{i-k} ← j-k end if
21     if d < P_{j-k} do P_{j-k} ← d, I_{j-k} ← i-k end if
22   end for
23 end for
24 return P, I
```

The overall time complexity of the algorithm is $O(n^2 \log n/s)$, where n is the length of the time series and s is the sampling interval. The space complexity is $O(n)$. From Fig. 12.*bottom*, we can see that after running PreSCRIMP, the running matrix profile aligns very well with the oracle matrix profile, especially at the minimum points, which for motif discovery, are all we care about.

The reader may wonder how we determine the sampling interval s. Note that any unsampled subsequence must overlap with one of the sampled subsequences by at least $1-s/(2m)$. Therefore, the smaller s is, the more accurate is our running matrix profile (and the longer PreSCRIMP takes to compute it). As a practical matter (as we will demonstrate later in Section IV), we set $s=m/4$, which guarantees that all the subsequences overlap with at least one sampled subsequence by at least 87.5%. This setting renders PreSCRIMP an $O(n^2 \log n/m)$ time complexity. As the subsequence length m is normally much larger than $\log n$, the time needed for PreSCRIMP is a tiny fraction required for SCRIMP/STOMP.

After running PreSCRIMP, we continue to refine the matrix profile with SCRIMP, until it converges to the exact solution. We call the augmentation of SCRIMP with PreSCRIMP, *SCRIMP++* (recall Fig. 7). Note that SCRIMP++ can be interrupted at any stage (including during the PreSCRIMP stage), to produce an approximate solution.

IV. Empirical Evaluation

To ensure that our experiments are reproducible, we have built a website which contains all data/code/raw spreadsheets for the results, in addition to many experiments that are omitted here for brevity [18]. All experiments were run on a Dell XPS 8920, with Intel Core i7-7700 CPU @ 3.6GHz and 64GB RAM.

A. Comparing Convergence Behaviors

We begin by comparing the convergence behavior of STAMP [15], STOMP [16] and SCRIMP++. Note that STOMP is not regarded as a true anytime algorithm but is included for completeness.

To stress-test these algorithms with different circumstances (different numbers and locations of motifs, different data type, etc.), we created four different synthetic datasets. Fig. 13 shows one example from each of the four datasets.

Each dataset includes 100 time series of length 40,000. Within each time series we embed various numbers of motif patterns of length m=400 *at random locations*. The first dataset (Fig. 13.*a*) is a set of random-walk time series; within each of these time series we embed a single pair of random-walk motif patterns (they are similar, but not identical). The second dataset (Fig. 13.*b*) is also random-walk data, but contains 10 pairs of different random-walk motif patterns. The third dataset (Fig. 13.*c*) is adapted from [16], where we have a continuous recording of seismograph background noise, and embed in it one pair of repeated earthquake signals (similar but not identical) at random locations. The fourth dataset (Fig. 13.*d*) is random noise time series; we embed no motifs in it, but regard its natural top-1 motif pattern of length 400 as target.

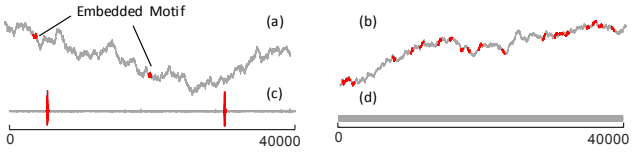

Fig. 13 *a*) Random-walk data with one pair of embedded random-walk motif patterns. *b*) Random-walk data with 10 embedded random-walk motif pairs. *c*) Seismology data with two repeated earthquake signals. *d*) Random noise without any embedded motif patterns.

As the algorithms evaluate the matrix profile of a time series, we constantly interrupt it, mark the current runtime t, then extract the top-k motif patterns (we set $k = 10$ for Fig. 13.*b*; $k = 1$ for Fig. 13.*a*, Fig. 13.*c* and Fig. 13.*d*) from the running matrix profile and check to see if the embedded motif patterns have been discovered. We regard an embedded motif pattern as discovered if it overlaps with one of the k extracted motif patterns by at least 95%. We use a value p to represent the percentage of embedded motif pairs discovered at each time instant t.

We first consider Fig. 13.*b*, where the random-walk time series includes 10 pairs of embedded motifs. Fig. 14 shows the average value of p as the three algorithms search for motifs.

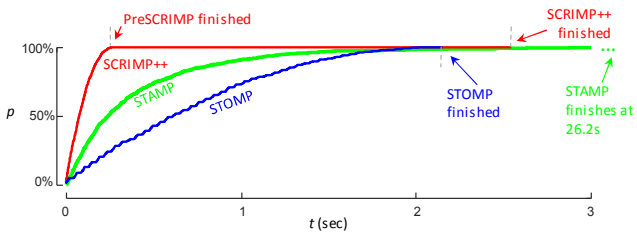

Fig. 14 The average percentage of embedded motif pairs discovered at each time instant for the dataset shown in Fig. 13.*b*. Note that the time for STAMP's convergence is truncated.

We can see that SCRIMP++ shows much faster convergence characteristics than STAMP or STOMP in locating the top 10 motif pairs. After the PreSCRIMP phase (requiring only 0.26 seconds) finishes, all the 10 embedded motifs randomly located in all 100 random-walk time series are successfully discovered. In contrast, to be just 99% sure that we have discovered all the true motifs, STAMP takes about 8 times longer and STOMP needs to almost run to completion (about 9 times longer).

Now let us consider the harder scenarios in Fig. 13.*a*, Fig. 13.*c* and Fig. 13.*d*, where there are only one pair of motif patterns in the data. We experimented in these scenarios because: 1) The top-1 motif in these datasets are hard to locate as they are rare. 2) The seismology data in Fig. 13.*c* is a typical example of "less-than-cooperative" data discussed in Section II.C, which would degenerate rival motif discovery methods such as Quick-Motif [4] or MK [6] to their worst case time complexity [16]. 3) The random-noise data in Fig. 13.*d* shows an extremely hard case for motif discovery, as essentially all pairs of time series subsequences are approximately equidistant. Nevertheless, as shown in Fig. 15, SCRIMP++ shows a very fast convergence characteristic in all these datasets. After the PreSCRIMP phase is completed (0.26 seconds), all the top-1 motifs in all the time series within all three datasets are already successfully discovered, costing only a tiny fraction of time needed by STOMP or STAMP. Note that here STAMP does not perform as well as in Fig. 14, as the motifs are very rare.

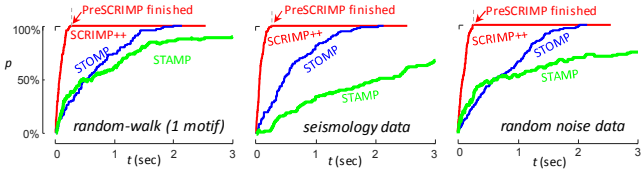

Fig. 15 *left-to-right)* The observed probability for the top-1 motif discovered at each time instant for the dataset shown in Fig. 13.*a*, Fig. 13.*c* and Fig. 13.*d*. Note that the full time for STAMP's convergence is truncated.

As we show in the next section, SCRIMP++ maintains this advantage over different lengths of time series and motif lengths. We chose to consider 40,000 data points here, because based on our informal survey of practitioners that *use* motif discovery, this is about the median size of datasets[2] considered [1][9]. Here we can find such motifs in just ¼ of a second, truly interactive time [11].

[2] To be clear, many biologists produce terabytes of data, but often each "run" or "treatment" is only of the order of tens to hundreds of thousands in length.

B. Runtime Comparison of SCRIMP++ and STOMP

In this section, we compare the run time of SCRIMP++ with the state-of-the-art exact motif discovery algorithm, STOMP [16]. The time measurements are based on the C++ implementation of both algorithms. Note that the runtime for both algorithms is invariant to the type of time series data we are using. TABLE I shows the time required by both algorithms with a fixed subsequence length m, on random noise time series with increasing length n.

TABLE I. TIME NEEDED FOR MOTIF DISCOVERY WITH $m = 4096$, VARYING n

Algorithm \ n		2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
STOMP		22.5s	1.78m	7.37m	37.1m	2.22h
SCRIMP++	PreSCRIMP	0.51s	2.33s	17.2s	1.52m	6.83m
	SCRIMP	23.9s	1.94m	7.96m	40.9m	2.46h

We can see that the runtime of the SCRIMP Algorithm is similar to the STOMP algorithm, as they vary only in evaluation order. The PreSCRIMP algorithm consumes only a very small fraction (less than 6%) of their time[3].

In TABLE II, we fixed the time series length n and vary subsequence length m. We can see that the runtime of STOMP and SCRIMP are essentially invariant to the subsequence length m. PreSCRIMP, with a time complexity $O(n^2 \log n/m)$, costs less and less time while we increase m. As m can be in the thousands for real-world problems (cf. Sections IV.D-IV.F), this is a desirable feature.

TABLE II. TIME NEEDED FOR MOTIF DISCOVERY WITH $n = 2^{18}$, VARYING m

Algorithm \ m		1024	2048	4096	8192	16384
STOMP		1.83m	1.78m	1.78m	1.8m	1.67m
SCRIMP++	PreSCRIMP	9.22s	4.81s	2.33s	1.23s	0.58s
	SCRIMP	2.17m	2.12m	1.94m	2.05m	1.96m

Furthermore, as PreSCRIMP is based on iterative vector operations, the computation process is highly parallelizable. Implementing PreSCRIMP with high-performance computing platforms such as GPU is trivial, and we make the GPU version freely available at [18].

C. Comparison to Rival Methods

We have argued that there is a critical difference between *approximate* algorithms and *anytime* algorithms for motif discovery. By definition, anytime algorithms are also approximate algorithms (if stopped early), but the converse is not true. If the motifs returned by an approximate algorithm are not satisfactory, the user has no recourse but to adjust parameters and try again, or resort to the fastest exact algorithm [16].

Nevertheless, it may be instructive to compare our proposed algorithm to the state-of-the-art *approximate* algorithm. But which algorithm *is* state-of-the-art for this task? A recent survey reviews more than a dozen algorithms without explicitly answering that question [10]. Fortunately, we can bypass this issue, and effectively compare to *all* of them. All such algorithms, whether they use hashing, grammars, Markov models, suffix trees etc. [10], must first convert the data into a symbolic representation. The time taken to do this is clearly a lower bound on the time to produce any motifs. Note that we cannot bypass this time requirement with any precomputation/indexing, as this is only possible if one knows the length of motifs, but as we have shown, this can be changed in an ad-hoc manner during the user's interactive session.

We used the code written by L. Wei [13] (which is the code used by the majority of papers reviewed in [10]), to discretize increasingly long time series, while keeping m fixed to 4,096 and a dimensionality of 8 and cardinality of 5 (typical values for most research efforts [10]). As Fig. 16 shows, we compare the runtime of this preprocessing discretization step of the rival algorithms to that of PreSCRIMP.

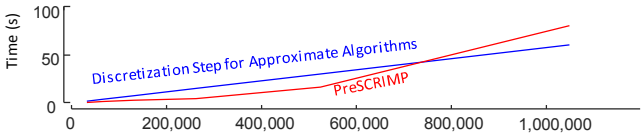

Fig. 16. The time needed to discretize data and the time need to perform PreSCRIMP for increasingly long data.

We can see that when the time series length is smaller than 2^{19}, SCRIMP++ has already reported a very high-quality solution with PreSCRIMP, before any approximate algorithm is even in a position to finally start the hashing or suffix tree construction that they hope will yield an approximate answer.

Note that this experiment offers an extremely weak lower bound for the cost of the rival approximate algorithms. In practice, the searching such algorithms take is 3 to 20 times longer than this preprocessing [10]. Finally, all these methods are reporting motifs found in a lossy data representation with the inherent error that produces, whereas SCRIMP++ is searching the *original* data.

D. Case Study: Multiscale-Motifs

We believe that the extraordinary speed of PreSCRIMP will allow the community to invent novel time series primitives. To give an example, we consider a question suggested by an entomologist collaborator: are there any *multiscale-motifs* in the EPG datasets previously discussed in Section I.A? We informally define a multiscale-motif as a pair of patterns that are very similar to each other but differ by at least a factor of two in length.

Clearly finding multiscale-motifs is computationally challenging, because beyond comparing all pairs of subsequences, we must now compare all pairs of subsequences, *and* at all possible combinations of scales. It may be possible to create a scalable novel algorithm to find multiscale motifs, but the speed of PreSCRIMP suggests a very easy "fast-enough" method that we can implement in a handful of lines of code, given PreSCRIMP as a primitive.

[3] Note that though we could further speed up PreSCRIMP with multi-threading or piece-wise FFT, we reported its run time here without any of these optimizations.

Recall we can use PreSCRIMP to do self-joins or AB-joins. Suppose we set B = rescale(A,300%) and compute an AB-join. The resulting motifs discovered will reflect a short pattern in A that matches a much longer pattern in B, after the patterns are scaled to a common length. In this case, we do not know what the "right" rescaling length is, but PreSCRIMP is fast enough to allow us to run it fifty times and simply test all possible scalings from 200% to 300%, in 2% increments. We have done this for a 1.8 hour (650,000 datapoints) long trace of Asian citrus psyllid (*Diaphorina citri*) feeding on Citrangor, a subspecies of orange. Fig. 17 shows that the best multiscale-motif occurs for a rescaling of 218%.

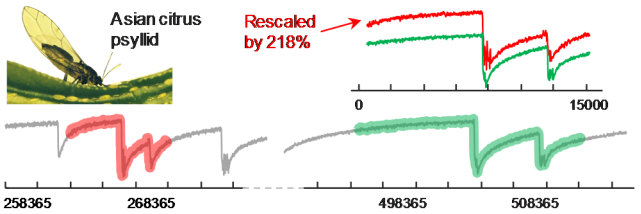

Fig. 17. *top.left*) An Asian citrus psyllid feeding on a citrango leaf. *top.right*) The top-1 multiscale-motif discovered. *bottom*) the two motif occurrences in context.

Note that one may wish to normalize the Euclidian distance for length when comparing multiscale-motifs (it happens to make no difference in this case). Further note that we are not claiming any particular entomological significance here, although it is interesting that this insect has behaviors that manifest themselves at such different scales. Our point is simply to show that PreSCRIMP is fast enough to be considered as a primitive we can call multiple times for higher-level analytics. The time taken for this entire experiment was just 84 seconds (m=15,000).

This ability to handle motifs that occur at different lengths may also be of interest to the neuroscience/neuroinformatics community, which has recently adopted time series motifs as one of their most used analytic tools [1][9]. However, some of these authors have criticized current motif discovery algorithms because they "*consider only exactly equal duration sequences as potential matches*" [9]. The authors of [9] note that motifs of "turning maneuvers" of *Drosophila* larval have a variable length scale, with μ = 0.83s and σ = 0.27s. Using the simple algorithm described above, we can find multiscale motifs in the range of $\mu \pm 2\sigma$ in a dataset of 40,000 points, searching *all* rescalings in 5% increments ([35%, 40%, … ,160%, 165%]) in just 17 seconds.

E. Case Study: Motif Joins

The EPG domain considered in the previous section is a rich source of fundamental problems that can be addressed with motif discovery, below we consider another such problem.

As shown in Fig. 18.*top*, we consider three datasets, each of length 7,560,000, representing 21-hours of insect behavior. One of them, in which the insect was feeding on Valencia (a type of orange), we designated as reference sample, $\text{Valencia}_{\text{Ref}}$. We are interested to know if any elements of this reference behavior are to be found in the two other datasets, in one of which an insect was feeding on a Yamaguchi (a different type of citrus), and the other in which a different insect is feeding on a Valencia. We hope to understand what elements of the Asian citrus psyllid may be attributed to the type of plant it is feeding on, and what may be attributed to simple differences between individuals. Such studies have implications for breeding resistant strains and hybrids.

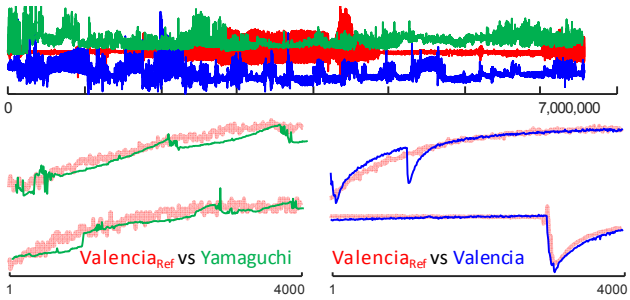

Fig. 18. *top*) The three EPG time series under investigation. *bottom-left to right*) There is little evidence of conserved patterns when the insects are feeding on different citrus plants, but there are strongly conserved patterns when feeding on a single plant type.

It is instructive to think of the cost of a brute-force-search here. The motifs are of length 4,000, requiring (at least) 4,000 real-valued operations. Each AB-join requires about $5.71 * 10^{13}$ pairwise comparisons of subsequences, requiring $2.28 * 10^{17}$ real-valued operations. Even at one hundred gigaFLOPS, this would require 26.4 days. In contrast, SCRIMP++ took just 2 hours.

F. Case Study: Electrical Power Demand

As a final example of the scalability of SCRIMP++, and the potential actionability of motif discovery, we examined the electrical power demand dataset of [8]. Each trace corresponds to two calendar years or 8,198,756 datapoints, sampled once every 8 seconds. As shown in Fig. 19, a pair of motifs from trace 3 of House-5 caught our attention.

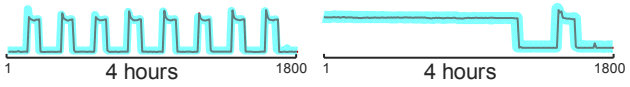

Fig. 19. The top two motifs in an electrical power data set.

The first motif is the (near) binary switching on-and-off of a freezer compressor at very regular intervals. This unusually "perfect" motif has dozens of occurrences, almost all at night when there is no kitchen actively that would cause the compressor to "kick-in" after the freezer was opened and disrupt the perfect spacing. The second motif is more interesting. It suggests that the compressor was running continuously for at least three hours. Two common causes of a freezer motor running for a long time are a faulty thermostat, or the more prosaic explanation, the homeowner not fully closing the door. In either case this is clearly a low-hanging fruit for energy conservation.

SCRIMP++ allows us to find such patterns in real-time interactive sessions, something that no other tool allows [10].

G. *When can PreSCRIMP fail?*

The previous sections have shown the extraordinary alacrity and effectiveness of PreSCRIMP. To explore the limits of PreSCRIMP, in section IV.A, we considered all the possible worst-case scenarios: when the motifs are very rare, when the dataset is of very high intrinsic dimensionality, when all the subsequence pairs are equidistant, etc. Nevertheless, PreSCRIMP succeeded in quickly locating all the true motifs in all these scenarios. It is natural to ask, can PreSCRIMP ever fail? Do we *ever* need to resort to running the SCRIMP phase of the SCRIMP++ algorithm, to refine the PreSCRIMP answer?

In spite of a diligent search of over 100 diverse datasets, we could not find any *real* dataset that prevents PreSCRIMP from quickly discovering motifs. However, with careful introspection, we can *create* a pathological example that is difficult for PreSCRIMP. As shown in Fig. 20.*top*, we created a synthetic random walk time series of length 40,000, with a pair of motifs embedded at fixed locations ($T_{21842,400}$ and $T_{24871,400}$, shown in red and yellow respectively). We edited the first/red motif pattern such that *just* before and after the pattern, the level of the time series dramatically changes. In this scenario, the CNP property no longer hold at locations around the motif patterns. Though $T_{21842,400}$ is very similar to $T_{24871,400}$, $T_{21842+k,400}$ is very different from $T_{24871+k,400}$ (k=*-3, -2, -1, 1, 2, 3*) because of the dramatic level change. As a result, PreSCRIMP cannot discover the motif pair unless either $T_{21842,400}$ or $T_{24871,400}$ is sampled.

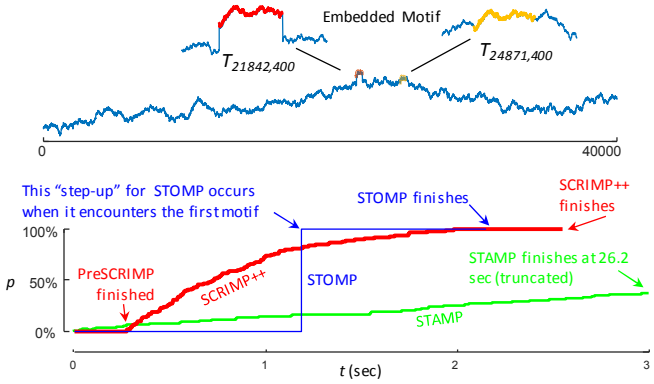

Fig. 20. *top*) A pathological random walk time series with a pair of embedded motifs. The level of the data dramatically changes just before and after the first motif pattern, which invalidates the CNP property. *bottom*) the observed probability for the top-1 motif discovered at each time instant. Note that the probability for STOMP is binary, and flips to 100% as soon as it encounters the first motif. That could happen arbitrarily late (i.e. to the far right) in the worse case.

However, as Fig. 20.*bottom* shows, the overall SCRIMP++ algorithm *still* converges much faster than STAMP [15] and STOMP [16] at the early stage. Here the result is averaged over 100 runs, and the value p represents the probability that the embedded motif pair is discovered at each time instant t. Although SCRIMP++ fails to discover the motif at the PreSCRIMP phase, p quickly increases as the algorithm turns into the SCRIMP phase thanks to its random computation ordering. In contrast, STOMP shows a 0% probability in locating the motifs until after 1.2 seconds (recall that STOMP is deterministic, and reports the same result over the 100 runs); STAMP shows a very low probability in finding the motifs even when SCRIMP++ finishes. This example demonstrates the robustness of SCRIMP++, even in the most pathological and contrived cases that defeat PreSCRIMP.

V. Conclusions

In many domains, including neuroscience [1][2], entomology [9], medicine and consumer-level energy conservation [8], etc., analysts routinely deal with datasets that are in the range of a few million data points long. For the first time, SCRIMP++ allows the possibility of real-time *interactive* discovery of motifs in such datasets, using off-the-shelf consumer desktops. We believe that this ability will allow novel discoveries to be made in the relevant domains, and even new types of analytics to be invented. We have made all code and data freely available in perpetuity to allow the community to confirm and extend our findings [18].

References

[1] Brown, A.E., et. al. A dictionary of behavioral motifs reveals clusters of genes affecting C. elegans locomotion. Proc. Natl. Acad. Sci. 110.2 (2013): 791-796

[2] Kolb, I., et. al. Evidence for long-timescale patterns of synaptic inputs in CA1 of awake behaving mice. Journal of Neuroscience 38.7(2018): 1821-1834

[3] Gu, Z., He, L., Chang, C., Sun, J., Chen, H., Huang C., Developing an efficient pattern discovery method for CPU utilizations of computers. International Journal of Parallel Programming. 45.4 (2017): 853-878.

[4] Li, Y., U, L.H., Yiu, M.L., Gong, Z. Quick-motif: An efficient and scalable framework for exact motif discovery. ICDE 2015: 579-590

[5] MASS, http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

[6] Mueen, A., Keogh, E., Zhu, Q., Cash, S., and Westover, M.B. Exact discovery of time series motifs. SDM 2009, vol. 9, pp. 473-484.

[7] Rakthanmanon, T., et. al. Mining trillions of time series subsequences under dynamic time warping. TKDD 7.3 (2013): 10

[8] REFIT: Smart Homes and Energy Demand Reduction. URL: www.refitsmarthomes.org/index.php/data. Accessed 01/21/2018.

[9] Szigeti, B., Deogade, A. and Webb, B. Searching for motifs in the behaviour of larval Drosophila melanogaster and Caenorhabditis elegans reveals continuity between behavioural states. Journal of the Royal Society Interface 12.113 (2015): 20150899.

[10] Torkamani, S. and Lohweg, V. Survey on time series motif discovery. Wiley Interdisciplinary Reviews: Data Min. Knowl. Discov. 7.2 (2017)

[11] Turkay, C., Kaya, E., Balcisoy, S., Hauser, H. Designing progressive and interactive analytics processes for high-dimensional data analysis. IEEE Trans. Vis. Comput. Graph. 23.1 (2017): 131-140.

[12] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E. Experimental comparison of representation methods and distance measures for time series data. Data Min. Knowl. Discov. 26.2 (2013): 275-309.

[13] Wei, L. SAX code for the N/n not equal an integer case (2006). URL www.cs.ucr.edu/~eamonn/SAX.htm

[14] Willett, D., George, J., Willett, N. Stelinski, L. and Lapointe, S. Machine learning for characterization of insect vector feeding. PLoS computational biology, 12.11 (2016): e1005158.

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

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

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

[18] Project Website: https://sites.google.com/site/scrimpplusplus/