Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining

Yan Zhu[1], Makoto Imamura[2], Daniel Nikovski[3], Eamonn Keogh[1]
[1]University of California, Riverside, yzhu015@ucr.edu, eamonn@cs.ucr.edu
[2]Tokai University, imamura@tsc.u-tokai.ac.jp
[3]Mitsubishi Electric Research Laboratories, nikovski@merl.com

***Abstract*— Since their introduction over a decade ago, time series motifs have become a fundamental tool for time series analytics, finding diverse uses in dozens of domains. In this work we introduce Time Series Chains, which are related to, but distinct from, time series motifs. Informally, time series chains are a temporally ordered set of subsequence patterns, such that each pattern is similar to the pattern that preceded it, but the first and last patterns are arbitrarily dissimilar. In the discrete space, this is similar to extracting the text chain "hit, hot, dot, dog" from a paragraph. The first and last words have nothing in common, yet they are connected by a chain of words with a small mutual difference. Time series chains can capture the evolution of systems, and help predict the future. As such, they potentially have implications for prognostics. In this work, we introduce a robust definition of time series chains, and a scalable algorithm that allows us to discover them in massive datasets.**

Keywords*— *Time Series, Motifs, Prognostics, Link Analysis

I. Introduction

Time series motifs are approximately repeating subsequences embedded in a longer time series. Since their formulation in 2002 [15] they have emerged as one of the most important primitives in time series data mining. Motif discovery has been used as a sub-routine in higher-level analytics, including classification, clustering, visualization [4], and rule-discovery [17]. Moreover, motif discovery has been applied to domains as diverse as severe weather prediction, robotics, medicine [20] and seismology [25].

In retrospect, it is easy to see why time series motifs are so useful. If a pattern is repeated (or *conserved*), there must be a latent system that occasionally produces the conserved behavior. For example, this system may be an overcaffeinated heart, sporadically introducing a motif pattern containing an extra beat (Atrial Premature Contraction [9]), or the system may be an earthquake fault, infrequently producing highly repeated seismograph traces because the local geology produces unique wave reflection/refractions [25]. Time series motifs are a commonly used technique to gain insight into such latent systems, in essence, they can be seen as "*generalizing the notion of a regulatory motif to operate robustly on non-genomic data*"[20].

In this work, we expand the notion of time series motifs to the new primitive of *time series chains* (or just *chains*). Time series chains may be informally considered motifs that evolve or drift in some direction over time. Fig. 1 illustrates the difference between time series motifs and time series chains (we defer formal definitions until Section II).

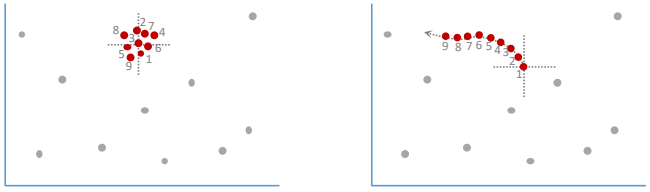

Fig. 1. Visualizing time series subsequences as points in high-dimensional space. *left*) A time series motif can be seen as a collection of points that approximate a platonic ideal, represented here as the crosshairs. *right*) In contrast, a time series chain may be seen as an evolving trail of points in the space. Here the crosshairs represent the first link in the chain, the *anchor*.

Both motifs and chains have the property that each subsequence is relatively close to its nearest neighbor. However, the motif set also has a relatively small diameter. In contrast, the set of points in a chain has a diameter that is much larger than the mean of each member's distance to its nearest neighbor. Moreover, the chain has the property of *directionality*. For example, in Fig. 1.*left*, if a tenth member was added to the motif set, its location will also be somewhere near the platonic ideal, but independent of the previous subsequences. In contrast, in Fig. 1.*right*, the location of the tenth member of the chain would be somewhere just left of item nine.

While we can clearly define *chains*, it may not be obvious that such constructs exist in the real-world. In fact, as we preview in Fig. 2, time series chains appear to be near ubiquitous in many domains, so long as the data trace is sufficiently long.

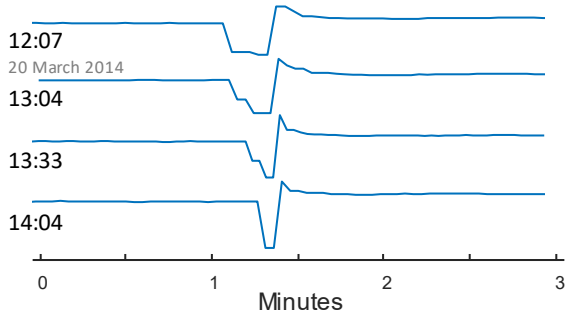

Fig. 2. A time series chain discovered in an electrical power demand dataset monitoring domestic freezer usage [14]. Note that through the early afternoon, the valley becomes narrower and the peak that follows it becomes sharper.

With a little introspection, it is clear that chains *should* exist. Consider the following systems:

- **Human Heart**: An overcaffeinated heart can sporadically produce a pattern containing an extra beat, but over time the caffeine leaves the blood stream, and the pattern fades [9].

- **Distillation Process**: A distillation column is a ubiquitous industrial tool used to separate a mixture into its component parts. Ideally, most telemetry monitoring a distillation column should reflect a *repeating* process, over production cycles. However, most large distillation columns are open to the atmosphere, and the patterns observed may drift as the seasons change. In addition, a slowly clogging feed pipe can throttle the feed rate and force the patterns to drift until they become unacceptable and force maintenance.
- **Aggregate Human Behavior**: Human behavior is often unpredictable for *individuals*, but more structured in *aggregate*. For example, online shopping behaviors often shows conserved motifs, but these motifs may drift over time in response to advertising campaigns or cultural shifts. This has been noted in recent studies. For example, [7] notes that their attempts to model consumer e-commerce visitation patterns "*suggests the existence of a slow rate of environmental change or exploration that would slowly undermine the model's accuracy*."
- **Machines:** In general, most mechanical and electrical systems such as cars, motors, elevators, air conditioners, etc. are subject to gradual deterioration over time. This deterioration can be manifested in shorter or longer duty cycles, increased vibration, or some other gradually changing pattern. In the field of prognostics, the degree of deterioration is often called the State of Health (SoH) of the system. SoH is rarely directly measurable, and its estimation typically involves advanced modeling and estimation algorithms. Because a time series chain defines an implicit curve in some high-dimensional space, as shown in Fig. 1, the natural coordinate along this curve can serve as a surrogate SoH measure. If high probability of failure can be associated reliably with a certain level of SoH, the discovered time series chain can be used successfully for prognostics and condition-based maintenance of machines.

As we will show in Section IV, once given the computational ability to find time series chains, we begin to find them everywhere, in datasets from ten seconds, to ten years in length.

The rest of this paper is organized as follows. In Section II we briefly review related work and background, then formally define time series chains. We introduce an ultra-fast algorithm to compute time series chains in Section III. Section IV shows the ubiquity of time series chains. Finally, in Section V we offer conclusions and directions for future work.

II. RELATED WORK AND BACKGROUND

Our review of related work is brief. To the best of our knowledge, there are simply no closely related ideas in the *time series* domain. However, there are very similar ideas in the *text* domain, even to the point of using similar language [24][1][22]. For example, Zhu and Oates discuss "*Finding Story **Chains** in Newswire Articles*" (analogous to our emphasis) [24]. Likewise, Bögel and Gertz argue for the need to go beyond finding repeated variants of news articles (like *motifs*), to allowing "*Temporal Linking of News Stories*" (like *chains*) [1]. Beyond the difference in data type considered, this work is much more supervised. The user typically selects a particular news article, and asks "*what leads up to this?*" or "*what happened next?*". In contrast, because we are often exploring domains for which we have limited intuitions, we want to tell the algorithms nothing (except the desired length of patterns to consider) and have the algorithm find the natural chains in the data (if any).

There is a huge body of work in finding *periodicity* in time series [8], however this work is orthogonal to the task-at-hand. A time series can have perfect periodicity, but no chains (i.e. a pure sine wave), and a time series can have chains, but no appreciable periodicity (it is easy to construct artificial examples, for example by embedding increasing damped sine waves in random walk).

The notion of chains invokes the familiar idea of *concept drift* in [3]; however, we are not starting with an explicit model to drift away from. Our starting point is a completely unannotated dataset.

Finally, time series chains are clearly related to time series motifs [15][23][25]. However, chains are neither a specialization nor a generalization of motifs. It is possible to have a rich set of motifs in a dataset, without having any chains. Time series motifs have a rich and growing literature, we refer the reader to [23][25] and the references therein.

A. Developing Intuition for Time Series Chains

To help the reader understand the task at hand, and our contributions to it, we begin by considering a similar problem in a domain that better lends itself to discussion. In particular, it will be helpful to sharpen our intuitions on *strings*, the discrete analogue to *time series*, and using the Hamming distance, the discrete analogue to the Euclidean distance.

A *word ladder* is a classic puzzle used to challenge children to build their vocabulary [11]. The challenge is as follows: given two related words, such as "cat" and "dog", find a path between the words that consists of legal English words that differ only in one letter. For example, this instance is solved by {cat, cot, dot, dog}. By definition, each word is exactly a Hamming distance of 1 from both its neighbors. Let us consider variants of this problem. Suppose our words are subwords of length m in a longer, unpunctuated string S, of length n:

thecatsleepinginthecotwasawokenbydothedogwh…

Further suppose that we are challenged to find the longest ladder (or *chain*) of words in this string. We are told only that the words are of length 3, and that each word is *at most* a Hamming distance of 1 from both its neighbors. The problem is still tenable by eye, at least for this short string. However, the problem becomes significantly harder if the words are no longer constrained to be English words:

uifdbutmffqjohjouifdpuxbtbxplfoczepuifephxi…

This string is actually just the previous string Cesar-shifted by one letter, but without the intuition of meaningful words, the problem becomes much harder for the human eye. Solving the problem with computers is also somewhat daunting. The obvious solution is depth-first-search, which only requires $O(n^2)$ space, but requires $O(n^n)$ time. If we constrain the subwords in a chain to have no overlap, the time complexity is slightly reduced to $O(n^{n/m})$.

Our consideration of strings allows further intuitive explanation of issues for the task at hand. Consider the following:

catauygfbatiuvheiucathoeircatiajesathfwecat…

Under the definition that each word is at most a Hamming distance of 1 from both its neighbors, this string has a chain of length six. However, this chain lacks *directionality*: the pattern is not drifting or evolving. Indeed, this "chain" might better be explained as multiple occurrences of a single prototypical pattern "cat", with some spelling errors. In the time series space, we already have a technique to find such patterns, *time series motifs* [15][23][25]. Thus, any definition we wish to formalize should guard against such pathological solutions.

Another important property that any definition of chains should have is *robustness*. Consider the following list of words that we will embed into a string {sad, had, ham, hag, rag}:

iwassadthatIhadahamsandwichwiththehaginrags…

Here we easily find the five-word chain. However, suppose we had a single letter misspelling in the string, for example:

iwassadthatIhadajamsandwichwiththehaginrags…

Because of this trivial single-letter difference, we can only find two chains of length two, something that might easily have happened by chance. This brittleness of chains has been understood for centuries. Alexander Pope noted in 1733 "*From Nature's chain whatever link you strike, tenth or ten thousandth, breaks the chain alike*". Thus, when designing the definition of chains for the time series space, we want to make sure that our definition is robust to one or two links being missing in an otherwise long chain. This is especially important in the time series domain where we often encounter noisy/missing data.

In summary, considering a simpler but related problem, we can see that when designing a formal definition for our task at hand, we must strive (at a minimum) to make it *efficiently computable*, *directional*, and *robust*. In the next section, we will introduce a definition of time series chains that satisfy these requirements.

Finally, this is a good place to introduce some nomenclature. We plan to support two types of time series chains (here we show their analogues in a string):

- **Unanchored**: In this case we are interested in finding the unconditionally longest chain in the string. For example, considering *S*, the first string we introduced, FindChain(*S*, *m*, default) would find the longest chain (with $m = 3$) of length 4: {cat, cot, dot, dog}.
- **Anchored**: In this case we want to start the chain with a particular subsequence. For example, FindChain(*S*, *m*, 20) would find the longest chain (with $m = 3$) starting with the subword at index 20, which is {cot, dot, dog}.

Note that if we have discovered all the anchored chains, the unanchored chain is simply the longest one among them.

B. Time Series Notation

Before we formally define time series chains, we need to review some related definitions (Definitions 1 to 3), and create some new ones (Definitions 4 to 11).

The data type of interest is *time series*:

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

A local region of 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 $1 \leq i \leq n-m+1$.

If we take a subsequence and compute its distance to *all* subsequences in the same time series, we get a *distance profile*:

Definition 3: A *distance profile* D_i of 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, $D_i = [d_{i,1}, d_{i,2}, \ldots, d_{i,n-m+1}]$, where $d_{i,j}$ $(1 \leq i, j \leq n-m+1)$ is the distance between $T_{i,m}$ and $T_{j,m}$.

This may *seem* like an expensive operation, but recent work has shown this can be achieved in just $O(n\log(n))$ time [13]. To concretely ground this, on a standard desktop (Intel i5-6330U CPU with 8GB Memory), it is possible to compute a distance profile for ten million data points in much less than 0.1 seconds.

We assume that the distance is measured by Euclidean distance between z-normalized subsequences [2][23][25].

Note that by definition, the i^{th} location of distance profile D_i is zero, and very close to zero just before and after this location. We avoid such "self" matches by ignoring an "exclusion zone" of length $m/4$ before and after the location of the query.

Recent work has introduced the *matrix profile* [23][25], a data structure that stores nearest neighbor information for every subsequence in a time series, and showed that it offers the solutions to many problems in time series data mining, including motif discovery and discord discovery. We propose to leverage these ideas. However, it is useful for us to "re-factor" the computation into two halves, independently considering the nearest neighbor to the *left*, and the nearest neighbor to the *right*. Note that the total amount of computation we need to do is the same. Fig. 3 previews the two data structures: *left matrix profile* and *right matrix profile*. We could create the original *matrix profile* [23][25] by simply taking the minimum of the two.

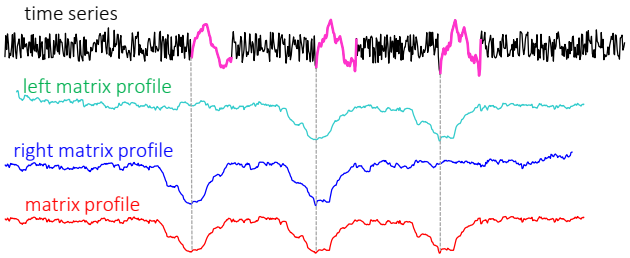

Fig. 3. The left matrix profile, right matrix profile and matrix profile of a toy time series. The deep valleys within the (left/right) matrix profiles indicate that the corresponding subsequence has close (left/right) nearest neighbors. The matrix profile shows general nearest neighbor information.

Before introducing the *left matrix profile* and *right matrix profile*, we begin by showing that we can divide a distance profile into a *left distance profile* and *a right distance profile*.

Definition 4: A *left distance profile* DL_i of time series T is a vector of the Euclidean distances between a given query subsequence $T_{i,m}$ and each subsequence that appears before $T_{i,m}$ in time series T. Formally, $DL_i = [d_{i,1}, d_{i,2}, \ldots, d_{i,i-m/4}]$.

Definition 5: A *right distance profile* DR_i of time series T is a vector of the Euclidean distances between a given query subsequence $T_{i,m}$ and each subsequence that appears after $T_{i,m}$ in time series T. Formally, $DR_i = [d_{i,\, i+m/4}, d_{i,\, i+m/4+1}, \ldots, d_{i,n-m+1}]$.

We can easily find the *left nearest neighbor* of a subsequence $T_{i,m}$ from the left distance profile, and the *right nearest neighbor* of $T_{i,m}$ from the right distance profile.

Definition 6: A *left nearest neighbor* of $T_{i,m}$, $LNN(T_{i,m})$ is a subsequence that appears before $T_{i,m}$ in time series T, and is most similar to $T_{i,m}$. Formally, $LNN(T_{i,m}) = T_{j,m}$ if $d_{i,j} = min(DL_i)$.

Definition 7: A *right nearest neighbor* of $T_{i,m}$, $RNN(T_{i,m})$ is a subsequence that appears after $T_{i,m}$ in time series T, and is most similar to $T_{i,m}$. Formally, $RNN(T_{i,m}) = T_{j,m}$ if $d_{i,j} = min(DR_i)$.

As shown in Fig. 3, we use a vector called *left matrix profile* to represent the z-normalized Euclidean distances between all subsequences and their left nearest neighbors:

Definition 8: A *left matrix profile* PL of time series T is a vector of the z-normalized Euclidean distance between each subsequence $T_{i,m}$ and its left nearest neighbor in time series T. Formally, $PL = [\min(DL_1), \min(DL_2), \ldots, \min(DL_{n-m+1})]$, where DL_i $(1 \leq i \leq n\text{-}m\text{+}1)$ is a left distance profile of time series T.

The i^{th} element in PL tells us the *distance* from subsequence $T_{i,m}$ to its left nearest neighbor in time series T. However, it does not tell *where* that left neighbor is located. This information is stored in a companion vector called the *left matrix profile index*.

Definition 9: A *left matrix profile index* IL of time series T is a vector of integers: $IL=[IL_1, IL_2, \ldots IL_{n-m+1}]$, where $IL_i=j$ if $LNN(T_{i,m}) = T_{j,m}$.

By storing the neighboring information this way, we can efficiently retrieve the left nearest neighbor of query $T_{i,m}$ by accessing the i^{th} element in the left matrix profile index.

Analogously, we define the *right matrix profile* (as shown in Fig. 3) and the *right matrix profile index* as follows:

Definition 10: A *right matrix profile* PR of time series T is a vector of the Euclidean distances between each subsequence $T_{i,m}$ and its right nearest neighbor in time series T. Formally, $PR = [\min(DR_1), \min(DR_2), \ldots, \min(DR_{n-m+1})]$, where DR_i $(1 \leq i \leq n\text{-}m\text{+}1)$ is a right distance profile of time series T.

Definition 11: A *right matrix profile index* IR of time series T is a vector of integers: $IR=[IR_1, IR_2, \ldots IR_{n-m+1}]$, where $IR_i=j$ if $RNN(T_{i,m}) = T_{j,m}$.

C. Formal Definitions of Time Series Chains

We are finally in the position to define time series chains. Before we do so, recall our guiding principle. We want something very like the definition of time series motifs [15][23][25], but with the additional property of *directionality*. For example, given a choice between the following:

{ ape → abe → ape → ape → abe → ape }

{ ape → apt → opt → oat → mat → man }

The latter is strongly preferred because the pattern is in some sense "evolving" or "drifting". We can now see this intuition in the *real-valued* space of interest. The definition below captures this spirit in the continuous case.

Definition 12: A time series chain of time series T is an ordered set of subsequences: $TSC=\{T_{C1,m}, T_{C2,m}, \ldots T_{Ck,m}\}$ $(C1 \leq C2 \leq \ldots \leq Ck)$, such that for any $1 \leq i \leq k\text{-}1$, we have $RNN(T_{Ci,m}) = T_{C(i+1),m}$, and $LNN(T_{C(i+1),m}) = T_{Ci,m}$. We denote k the *length* of the time series chain.

To help the reader better understand this definition, let us consider the following time series:

47, 32, 1, 22, 2, 58, 3, 36, 4, -5, 5, 40

Assume that the subsequence length is 1, and the distance between two subsequences is simply the absolute difference between them (to be clear, we are making these simple and pathological assumptions here just for the purposes of elucidation; we are actually targeting much longer subsequence lengths and using z-normalized Euclidean distance in our applications). According to Definition 9 and Definition 11, we can store the left and right nearest neighbor information into the left and right matrix profile indices, as shown in Fig. 4.

Index	1	2	3	4	5	6	7	8	9	10	11	12
Value	47	32	1	22	2	58	3	36	4	-5	5	40
IR	12	8	5	8	7	12	9	12	11	11	12	-
IL	-	1	2	2	3	1	5	2	7	3	9	8

Fig. 4. The left nearest neighbor index and right nearest neighbor index of the toy example.

Here the Index vector shows the location of every subsequence in the time series, *IR* is the right matrix profile index and *IL* is the left matrix profile index. For example, $IR[2]=8$ means the right nearest neighbor of 32 is 36; $IL[3]=2$ means the left nearest neighbor of 1 is 32.

To better visualize the left/right matrix profile index, in Fig. 5 we use arrows to link every subsequence in the time series with its left and right nearest neighbors.

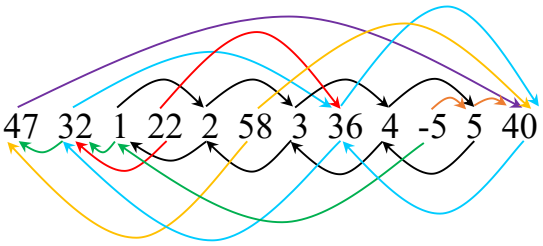

Fig. 5. Visualizing left matrix profile index and right matrix profile index: every arrow above the time series points from a number to its right nearest neighbor; every arrow below the time series points from a number to its left nearest neighbor.

We call an arrow pointing from a number to its right nearest neighbor (arrows shown *above* the time series) a *forward arrow* (i.e. $x \rightarrow y$ means $RNN(x)=y$), and an arrow pointing from a number to its left nearest neighbor (arrows shown *below* the time series) a *backward arrow* (i.e. $x \leftarrow y$ means $LNN(y)=x$). Definition 12 indicates that every pair of consecutive subsequences in a chain must be connected by both a forward

arrow and a backward arrow. The diligent reader may quickly discover the longest time series chain in our toy example:

47, 32, 1, 22, 2, 58, 3, 36, 4, -5, 5, 40 (*Raw data*)

1 ⇌ 2 ⇌ 3 ⇌ 4 ⇌ 5 (*Extracted chain*)

We can see that this chain shows a gradual increasing trend of the data. Note that in this one-dimensional example, the elements of the chain can only drift by increasing or decreasing. In the more general case, the elements can drift in arbitrarily complex ways. Our claim is that our definition is also capable of discovering complex drifting patterns in high-dimensional space. For example, the reader can easily verify that the two-dimensional chain in Fig. 1.*right,* a curvy evolving pattern, is captured by our definition. The definition also works for a sin-wave drifting pattern, a zigzag, spirals, etc. We defer real-world examples in much higher dimensional spaces to Section IV.

However, to be clear, we are *not* claiming that we can discover *all* kinds of drifting; we are only targeting chains with *directionality* (the last item should be very different from the first item, as suggested previously). Therefore, full closed circles (i.e. {1, 3, 4, 5, 1}) are not captured by our definition. However, if needed, we can still potentially capture such topologies if we consider combining multiple chains. For example, in {1, 3, 4, 5, 1}, our definition captures two chains: {1, 1} and {3, 4, 5}. The circle is a combination of the two.

Beyond satisfyingly the *directionality* requirement, here we provide a simple sanity check of the *robustness* of our definition by *removing* a link from the chain. Imagine that in Fig. 5, the number "3" is missing. In this case, $RNN(2)=4$, $LNN(4)=2$; we can still find the chain 1 ⇌ 2 ⇌ 4 ⇌ 5. We defer a more "stressed" and quantified robustness test to Section IV.E.

As suggested in Section II.A, we are especially interested in supporting two types of time series chains: anchored and unanchored chains. We formally define them as follows.

Definition 13: An *anchored* time series chain of time series T starting from subsequence $T_{j,m}$ is an ordered set of subsequences: $TSC_{j,m} = \{T_{C1,m}, T_{C2,m}, \ldots T_{Ck,m}\}$ $(C1 \leq C2 \leq \ldots \leq Ck, C1=j)$, such that for any $1 \leq i \leq k-1$, we have $RNN(T_{Ci,m})=T_{C(i+1),m}$, and $LNN(T_{C(i+1),m})=T_{Ci,m}$; for $T_{Ck,m}$, we have either $T_{Ck,m}$ is the last subsequence in T, *or* $LNN(RNN(T_{Ck,m})) \neq T_{Ck,m}$.

We can "grow" an anchored chain step-by-step as follows. Consider Fig. 5 as an example. If we start from 1, we find $RNN(1)=2$ and $LNN(2)=1$, so 2 can be added to the chain; since $RNN(2)=3$ and $LNN(3)=2$, 3 can also be added; this process continues until we reach 5. As $RNN(5)=40$ and $LNN(40) \neq 5$, the chain terminates, and finally we find the chain 1⇌2⇌3⇌4⇌5 as the longest chain starting from 1.

Note that our definition produces one and only one anchored time series chain starting from any user-supplied subsequence $T_{j,m}$ $(1 \leq j \leq n-m+1)$, as there is only one right (and also left) nearest neighbor for every subsequence in T. Based on this observation, we can find *all* the time series chains within T.

Definition 14: An all-chain set S_{TSC} of time series T is a set of all anchored time series chains within T that are not subsumed by another chain.

Here we are not simply finding all the anchored chains starting from all subsequences of T; S_{TSC} excludes those that are subsumed by another chain. For example, the all-chain set corresponding to Fig. 5 is S_{TSC} = {47, 32⇌36⇌40, 1⇌2⇌3⇌4⇌5, 22, 58, -5}. S_{TSC} does not contain the anchored chain 36⇌40, or 2⇌3⇌4⇌5, as they are both subsumed by longer chains.

Note that the all-chain set S_{TSC} has an important property: every subsequence of T appears exactly once in S_{TSC}. The all-chain set shows all possible evolving trends within the data.

We believe that of all the chains in S_{TSC}, the longest one should reflect the most general trend within the data. We call this chain the unanchored time series chain.

Definition 15: An *unanchored* time series chain of time series T is the longest time series chain within T.

Note that there can be more than one unanchored time series chain of time series T with the same maximum length. In case of such ties, we report the chain with minimum average distance between consecutive components. However, one might imagine other tie-breaking criteria, such as choosing the chain with smaller variance of consecutive pairwise distances.

One can imagine in some situations that the chain of interest may not be the longest one. In the next section, we provide an algorithm to compute the all-chain set, which we use to easily find *any* anchored or unanchored chain, from the set.

III. DISCOVERING TIME SERIES CHAINS

To compute the time series chains, according to Definition 12, we first need to find the left/right nearest neighbor of every subsequence in the time series. Such information can be found from two vectors: left matrix profile index and right matrix profile index (Definitions 9 and 11). The *LRSTOMP* algorithm is an (optimal) algorithm to efficiently compute these vectors.

A. LRSTOMP Algorithm

The recently introduced STOMP algorithm [25] can efficiently compute matrix profile and matrix profile index in $O(n^2)$ time and $O(n)$ space. Here we briefly review how STOMP [25] keeps track of the nearest neighbor of every subsequence: the algorithm computes distance profiles $D_1, D_2, \ldots, D_{n-m+1}$ (see Definition 3) in order. The matrix profile P is initialized as D_1 and the matrix profile index I is initialized as a vector of ones. As shown in Fig. 6, once the computation of D_i is completed, we compare every element of D_i with its corresponding element in P: if $d_{i,j} < P_j$, we set $P_j = d_{i,j}$ and $I_j = i$. In this way, the matrix profile P and matrix profile index I keep track of the nearest neighbors of every subsequence in the time series.

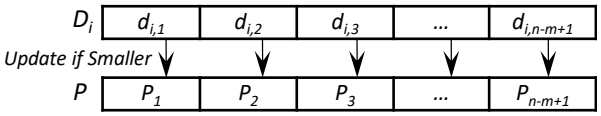

Fig. 6. STOMP keeps track of the general nearest neighbor of every subsequence in the time series

Instead of finding the general nearest neighbor information as in STOMP [25], to support chain discovery, we need to

separately find the left and right nearest neighbors of each subsequence in the time series.

Leveraging off the insights of STOMP [25], we call our algorithm LRSTOMP (Left-Right-STOMP). To initialize our four output vectors, we begin by setting both the left and right matrix profiles *PL* and *PR* as *Inf*s, and both the left and right matrix profile indices *IL* and *IR* as *zeros*. Then, using the technique in [25], we compute the distance profiles D_1, D_2, ..., D_{n-m+1} (see Definition 3) in order. Note that the i^{th} subsequence can only be the right nearest neighbor of the 1^{st} to the $(i-m/4)^{th}$ subsequence in the time series, and the left nearest neighbor of the $(i+m/4)^{th}$ to the last subsequence in the time series. Therefore, as shown in Fig. 7, after the i^{th} distance profile D_i is computed, we need to divide D_i into two halves. For $\forall\, j \in [1, i-m/4]$, if $d_{i,j} < PR_j$, we set $PR_j = d_{i,j}$ and $IR_j = i$. For $\forall\, j \in [i+m/4, n-m+1]$, if $d_{i,j} < PL_j$, we set $PL_j = d_{i,j}$ and $IL_j = i$.

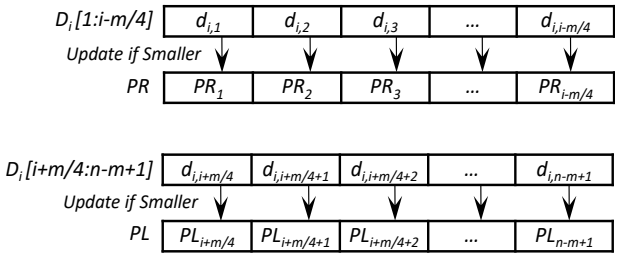

Fig. 7. LRSTOMP keeps track of both the left and right nearest neighbors of every subsequence in the time series.

After evaluating all of the distance profiles, we can obtain the final left and right matrix profiles (and matrix profile indices).

Note that switching the updating process from Fig. 6 to Fig. 7 does not affect the overall complexity of the algorithm. Therefore, the time complexity of LRSTOMP is $O(n^2)$ and the space complexity is $O(n)$, the same as STOMP [25].

B. Computing the Time Series Chains

Now we are in the position to compute the time series chains. We begin with the simpler variant, the algorithm to compute the anchored time series chains (ATSC) in TABLE I.

TABLE I. ATSC ALGORITHM

```
   Procedure ATSC(IL, IR, j)
   Input: The left matrix profile index IL and
   right matrix profile index IR generated by
   LRSTOMP(T, m), where T is the time series and
   m is the subsequence length; and j, location
   of the anchor subsequence.
   Output: Anchored Time series chain C, where
   C[i]=j means the i^th element of the chain is
   the j^th subsequence in the time series
1  C ← [j]  // initialization
2  while IR[j]≠0 and IL[IR[j]]==j do
3     j ← IR[j]
4     C ← [C, j]
5  end while
6  return C
```

The algorithm is straight-forward. We begin growing the chain from its user-specified anchor, the j^{th} subsequence. If the right nearest neighbor exists (if it does not exist, then *IR[j]*=0; this indicates that we have reached the end of the time series) and $LNN(RNN(T_{j,m}))=T_{j,m}$, then we set *j* as $RNN(T_{j,m})$ and add it to the back of the chain. The process iterates until nothing more can be added to the chain.

The time and space overhead of ATSC algorithm are both $O(n)$.

Given that we can efficiently compute the anchored time series chain starting from any subsequence, the all-chain set (ALLC) can also be computed. The (unanchored) time series chain is simply the longest chain in the all-chain set.

A simple approach to compute the all-chain set is enumerating all anchored chains starting from all subsequences, and removing those that are subsumed by longer chains. However, this brute-force approach would result in an undesirable $O(n^2)$ time complexity. Fortunately, as shown in TABLE II, by exploiting several properties of our definition of time series chains, we can reduce the time complexity of the ALLC algorithm to $O(n)$.

TABLE II. ALLC ALGORITHM

```
   Procedure ALLC(IL, IR)
   Input: The left matrix profile index IL and
   right matrix profile index IR generated by
   LRSTOMP(T, m), where T is the time series and m
   is the subsequence length.
   Output: The all-chain set S and the unanchored
   chain C
1  L ← ones, S ← ∅//initialization
2  for i : 1 to length(IR) do
3    if L[i]==1 do
4      j ← i, C ←[j]
5      while IR[j]≠0 and IL[IR[j]]==j do
6        j ←IR[j], L[j]←-1, L[i]←L[i]+1, C ←[C,j]
7      end while
8      S ← S ∪ C
9    end if
10 end for
11 C ← ATSC(IL, IR, Max_Index(L))
12 return S, C
```

The vector *L* in line 1 is a vector of length *n-m+1*, the same length as the four meta time series. We use *L[i]* to store the length of the anchored time series chain starting from *L[i]*, and initialize *L* with all ones (as the length of an anchored chain is at least 1). In lines 2 to 10, we iterate through all possible anchor points, and store in *L[i]* the length of the anchored chain starting from the i^{th} subsequence. We store all the chains found in *S*. In line 11, we find the unanchored time series chain corresponding to the maximum value in *L*.

Note that in lines 5-7, as we grow an anchored chain from the i^{th} subsequence, we set *L[j]* to *-1* for every subsequence *j* visited except the anchor subsequence. This helps us prune unnecessary computations, as there is only one anchored time series chain starting from any subsequence. Consider again the toy example in Fig. 5: when *i*=3, we discover the chain $1 \rightleftharpoons 2 \rightleftharpoons 3 \rightleftharpoons 4 \rightleftharpoons 5$. By marking out the length of the anchored chain starting from 2, 3, 4 and 5 as -1s, we can avoid spending time on growing a chain like $2 \rightleftharpoons 3 \rightleftharpoons 4 \rightleftharpoons 5$, which is subsumed by $1 \rightleftharpoons 2 \rightleftharpoons 3 \rightleftharpoons 4 \rightleftharpoons 5$. With this technique, every subsequence in the time series is visited exactly once; therefore, the time complexity of the algorithm is $O(n)$, which is inconsequential compared to

the $O(n^2)$ time (already demonstrated as ultra-fast in [25]) to compute the left/right matrix profiles and matrix profile indices. This $O(n)$-complexity algorithm is the optimal algorithm to compute the all-chain set under our definitions and assumptions, as we need to at least scan through the entire time series once.

Although we will mainly be showing the applications of the unanchored (or longest) time series chain, sometimes the chain of interest may not be the longest one. Based on domain knowledge, one may be interested in looking at the top-k chains, a chain starting from a specific location, a chain with less difference between the links, etc. All these tasks are trivial given the all-chain set *S*. Therefore, the ALLC algorithm can potentially help us discover any possible evolving trend within the time series. We reserve such considerations for future work.

IV. Empirical Evaluation

"*You reasoned it out beautifully, it is so long a chain, and yet every link rings true*." Sir Arthur Conan Doyle: Adventures of Sherlock Holmes, 1892.

We note in passing that all the experimental results in this paper are reproducible. To ensure this, we have created a website to archive all the datasets and code in perpetuity [26].

After an extensive literature search, we are convinced that there is no strawman algorithm to compare to. Moreover, unlike clustering or motif discovery, there is no formal metric to measure the quality of chains. In a sense, we are not the ideal group that should invent such a metric, as we could define one that tautologically rewards the properties *we* have defined. However, in Section IV.E, we provide a pseudo measure of quality as the gold standard. We measure the length of a chain in a data set, then ask how robust would our chain discovery algorithm be if we distorted the data in various ways. Clearly a chain discovery algorithm would engender little confidence if minor changes to the data could prevent the discovery of the (same basic) chain.

Before this robustness test, we provide four case studies in which we applied our algorithm to various datasets. These case studies will help the reader gain an appreciation for the utility of chain discovery. These datasets are designed to span the diverse types of data encountered in time series data mining, some are stationary, some have trends, some are smooth, some are noisy, the shortest is ten seconds long, the longest is ten years, etc.

While we can obtain the all-chain set with the ALLC algorithm, in this section, we are only showing the application of the unanchored time series chain. Unless otherwise stated, in the rest of this section, we use the term "time series chain" to represent unanchored time series chain in Definition 15, rather than Definition 12.

A. Case Study: Hemodynamics

In November 2016, we briefed Dr. John Michael Criley, Professor Emeritus at the David Geffen School of Medicine at UCLA, and Dr. Gregory Mason of UCLA Medical Center, a noted expert on cardiac hemodynamics, on the capabilities of time series chain discovery. They suggested more than a dozen possible uses for it in various clinical and research scenarios in medicine. Here we consider one example they are interested in.

Syncope is the loss of consciousness caused by a fall in blood pressure. The tilt-table test (see Fig. 8.*top.left*) is a simple, noninvasive, and informative test first described in 1986 as a diagnostic tool for patients with syncope of unknown origin [5].

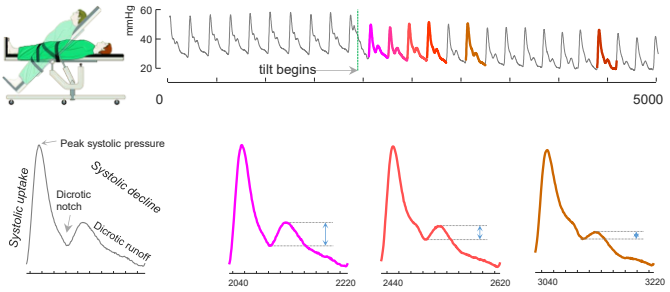

Fig. 8. *left-to-right*, *top-to-bottom*) A patient lying on a medical tilt table has his arterial blood pressure monitored. Nomenclature for a standard beat. The chain discovered in this dataset shows a decreasing height for the dicrotic notch.

Beyond diagnosing the condition, the test may reveal the cause, neurological disorder, metabolic disorder, mechanical heart disease, cardiac arrhythmias, etc. [12].

In brief, the clinician will want to contrast any evolving patterns in the patient's arterial blood pressure (ABP) that are a response to changes in positon induced by a tilt table, with evolving patterns that are not associated with changes of posture. As hinted at in Fig. 8, time series chains are an ideal way to find and summarize such patterns. Here we set m=200, as this is the typical length of an ABP signal (Fig. 8.*bottom.left*).

Fig. 8 shows just a snippet of the time series searched. We encourage the reader to see the full dataset/results at [26]. Nevertheless, even this snippet is visually compelling. It shows that as the table is tilted, the height of the dicrotic notch steadily decreases. Per Dr. Mason, the change in orientation "*dramatically increases central venous filling and subsequent left ventricular end-diastolic volume, for several heart beats. Left ventricular stroke volume and effective cardiac output increase transiently*, (likely due to) *relative hyperemia, which is well-described during recovery from transient vascular occlusion*".

As noted above, Fig. 8 only shows a small section of the data we searched. In addition to finding meaningful chains, a good algorithm should avoid finding spurious chains, even if there are dense motifs (recall the distinction visualized in Fig. 1). In Fig. 9 we show the prefix of the data we searched, but truncated out of Fig. 8, gratifyingly, the chain we discovered has no element here, even though there are clearly dense motifs [23].

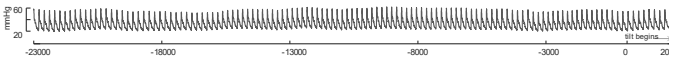

Fig. 9. The prefix of the ABP data shown in Fig. 8. There are no chain elements discovered in this region, although it is compressed of dense motifs.

B. Case Study: Penguin Behavior

In this case study, we decided to explore a dataset for which we have no expertise, to see if we could find time series chains, which we could then show to an expert for independent evaluation of meaning and significance (if any).

To this end, we consider telemetry collected from a Magellanic penguin (*Spheniscus magellanicus*). The dataset was collected by attaching a small multi-channel data-logging device to the bird. The full data consists of 1,048,575 data points recorded at 40 Hz (about 7.2 hours). While a suite of measurements was recorded, for simplicity we focus on the X-Axis acceleration (the direction of travel for a swimming bird). In Fig. 10 we show the snippet of the data in which we found a chain, with m=28. This is about 0.7 seconds, and the approximate period of the data.

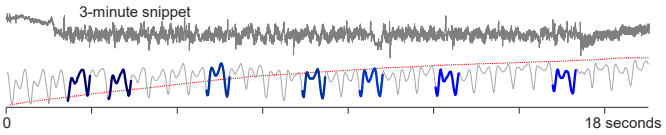

Fig. 10. *top*) A random three-minute snippet of X-Axis acceleration of a Magellanic penguin (from a total of 7.2 hours). *bottom*) An eighteen-second long section containing the time series chain. In the background, the red time series records the depth, starting at sea-level and leveling off at 6.1 meters.

In fact, this chain *does* have a simple interpretation. Adult Magellanic penguins regularly dive to depths of up to 50m to hunt prey, and may spend as long as fifteen minutes under water. One of our sensors measures pressure, which we showed in Fig. 10.*bottom* as a fine/red line. This shows that the chain begins just after the bird begins its dive, and ends as it reached its maximum depth of 6.1m. Magellanic penguins have typical body densities for a bird at sea-level, but just before diving they take a very deep breath that makes them exceptionally buoyant [16]. This positive buoyancy is difficult to overcome near the surface, but at depth, the compression of water pressure cancels it, giving them a comfortable neutral buoyancy [16][21]. In order to get down to their hunting ground below sea level it is clear that "(for penguins) *locomotory muscle workload, varies significantly at the beginning of dives*" [21]. The snippet of time series shown in Fig. 10 does not suggest much of a change in *stroke-rate*, however penguins are able to vary the thrust of their flapping by twisting their wings [21]. The chain we discovered shows this dramatic sprint downwards leveling off to a comfortable cruise. Fortunately, our data contains about a dozen major dives, allowing us to confirm our hypothesis about the meaning of this chain on more data.

Note that our chain does not include every stroke in the dive. Our data is undersampled (only 40Hz for a bird that can swim at 36kph) and this data is recorded in the wild, the bird may have changed directions to avoid flotsam or fellow penguins. However, this is a great strength of our algorithm: we do not need "perfect" data to find chains; we can find chains in real-world datasets. Also, from Fig. 10.*bottom* we can see that m=28 is longer than the actual period of the data; our algorithm is not sensitive to this and still discovered a meaningful chain.

C. Case Study: Human Gait

In the experiments in the previous section we could be sure of the validity of the discovered chains, because we had access to some ground truth. In this section and the next, we show examples of chains we discovered in datasets for which we do not have an obvious way to empirically verify. This demonstrates one use for chains, finding patterns that are interesting but speculative, and may warrant further investigation.

We first consider a snippet of a gait dataset recorded to test a hypothesis about biometric identification [6]. The dataset is shown in Fig. 11.*top*. We set m = 50 here, as this is the approximate length of a period of the data.

As hinted at in Fig. 11.*inset* (taken from the original paper), the authors of the study were interested in "*the instability of the mobile in terms of its orientation and position when it is put freely in the pocket*" [6]. Given the experimental setup, we suspected that the gait pattern might start out as being unpredictable as the phone jostled about in the user's pocket, eventually settling down as the phone settled into place. This is exactly what we see in Fig. 11.*top*. Note that the first few links are far apart and asymmetric, but the last few links are close together, and almost perfectly symmetric.

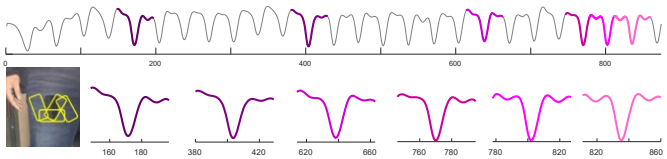

Fig. 11. *top*) A 30-second snippet of data from an accelerometer on a mobile phone. The phone was placed in the user's front pocket (*inset*). *bottom*) The extracted chain shows an evolution to a stable and symmetric gait

D. Case Study: Web Query Volume

In contrast to the smooth, stationary, oversampled accelerometer data considered in the last section, we next consider a dataset that is noisy, under-sampled and has a growing trend. We examined a decade-long *GoogleTrend* query volume for the keyword *Kohl's*, an American retail chain (data courtesy of [10]). As shown in Fig. 12, the time series features a significant "bump" around the end-of-years holidays, unsurprising for a store known as a destination for gift buyers. Here we set m = 76 (the approximate length of a "bump"). Note that m does *not* need to be precisely set. If we set m = 114 (50% longer), we can still obtain the same basic chain (though each link is 50% longer, see [26] for a visual comparison).

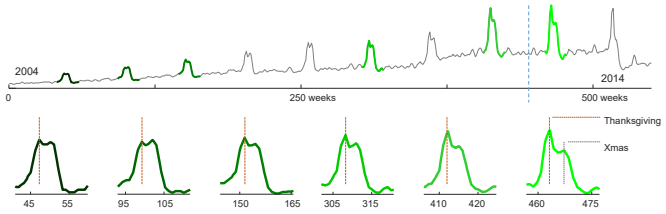

Fig. 12. *top*) Ten years of query volume for the keyword *Kohl's*. *bottom*) The z-normalized links of the time series chain discovered in the data hints at the growing importance of "*Cyber Monday*".

The discovered chain shows that over the decade, the bump transitions from a smooth bump covering the period between Thanksgiving and Xmas, to a more sharply focused bump centered on Thanksgiving. This seems to reflect the growing importance of *Cyber Monday*, a marketing term for the Monday after Thanksgiving. The phrase was created by marketing companies to persuade people to shop online. The term made its

debut on November 28th, 2005 in a press release entitled "*Cyber Monday Quickly Becoming One of the Biggest Online Shopping Days of the Year*" [19]. Note that this date coincides with the first glimpse of the sharping peak in our chain.

Here we seem to "miss" a few links in the chain. However, note that the data is noisy and coarsely sampled, and the "missed" bumps are too distorted to conform with the general evolving trend. This noisy example again illustrates the robustness of our technique. As before, we note that we do not need "perfect" data to find meaningful chains. Even if some links are badly distorted, the discovered chain will still be able to include all the other evolving patterns.

Furthermore, consider the potential of using chains to predict the future. Assume that we are now at mid-2012 (the location of the blue line in Fig. 12. *top*. We would like to forecast the shape S of the fist "bump" *after* the blue line, given the data *before* it.

In the data prior to mid-2012, we discovered a chain that consists of the first five links in Fig. 12. *bottom* (call them S_1, S_2, S_3, S_4, S_5). Our assumption is that the difference between S_4 and S_5 is the same as the difference between S_5 and S. We compare our prediction result with a popular strawman in the literature, *persistence* prediction (i.e. which assumes $S = S_5$) [18], in Fig. 13. Our simple, chain-based prediction method is more accurate (especially in the center part), as it captures the trend of the data.

Fig. 13. *left*) Our predicted shape (blue) is very similar to gound truth (red), with a Root Mean Squared Error (RMSE) of 0.17. *right*) Persistence prediction result (blue) is less similar to the ground truth (red), with a RMSE of 0.18.

E. Quantifying the Robustness of Chains

In the previous sections we showed the broad applicability of time series chains, and implicitly showed the robustness of our algorithm/definitions; given that it can find meaningful chains even in real-world "non-perfect" datasets. To further demonstrate this robustness, we need to provide a measure of the quality of time series chains that does not tautologically reward the properties we have *defined*, and can serve as a "gold standard" to compare the quality of chains before and after we have added some confounding factors.

To test the quality of our chain-discovery algorithm, we should consider two different scenarios: If the data include a long *intrinsic* chain, then a good algorithm should be able to discover (or "recover") a large portion of it. On the other hand, if the time series does *not* have any *intrinsic* evolving trend (for example, the data merely contains k repeated patterns), then we expect the length of the longest chain to be much shorter than k. We will test our algorithm in both scenarios.

Suppose we have a time series with an *intrinsic* chain of length k (that is to say, we know, possibly from external knowledge, that there should be exactly k evolving subsequences of length m in the time series, and we have a set L_{known}: $|L_{known}|=k\times m$ that shows the locations of all the data points within the embedded chain). Further suppose that, without knowing this, an algorithm discovers a time series chain of length $k_{discovered}$, and the locations of the $k_{discovered}\times m$ data points within the discovered chain is stored in the set $L_{discovered}$. Then we can define the *recall* of the chain as $R = \frac{|L_{discovered} \cap L_{known}|}{|L_{known}|}$ and the *precision* as $P = \frac{|L_{discovered} \cap L_{known}|}{|L_{discovered}|}$. For a robust chain, we expect $P \approx 1$. However, note that R does not necessarily need to be as large. Recall the example in Fig. 12; although the discovered chain only covers around 60% of the "bumps", it still reflects the general trend of the data.

Therefore, once L_{known} is given, R and P are excellent measures of quality for the discovered chain. We propose to exploit this idea by building synthetic time series for which we know true chains (both length and locations), and distorting the data to "stress-test" the chain discovery algorithm.

Fig. 14 shows an example of such a time series, with an embedded chain with k=5. Here the subsequences evolve gradually from a sine wave to a random-walk pattern, and in between the chain elements we inserted snippets of random noise.

Fig. 14. Synthetic time series embedded with a chain of five subsequences. The subsequences evolve from a sine-wave to a random-walk pattern.

We used 100 different random-walk patterns like the one in Fig. 14 to generate our benchmark time series. Each time series includes 20 subsequences of length 50 (k=20, m=50), evolving gradually from a sine wave to a random-walk pattern. Fig. 15.*top* shows how the average results of R and P vary, over the 100 runs as we increase the noise level.

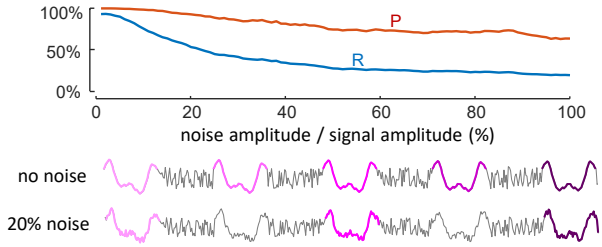

Fig. 15. *top*) *Recall* (R) and *Precision* (P) both decrease as the noise amplitude increases. *bottom*) A snippet of a "perfect" time series versus the same snippet with 20% noise added.

For a large amount of noise (1%~10% of the signal amplitude), we can successfully recover most of the embedded chain elements (more than 14 out of 20), with R>70% and P>95%. This demonstrates the robustness of our algorithm: though we missed a small number of embedded patterns, most of them are still recovered.

However, when the noise amplitude gets over 20%, R becomes smaller than 50%. This is because the noise level becomes large enough to hide the evolving characteristics within some part of the data. To see this, in Fig. 15.*bottom* we compared

a snippet from a "perfect" benchmark time series without noise to the same snippet with 20% noise. The evolving trend is originally clear in the "perfect" time series; when the noise amplitude increases to 20%, the second and fourth patterns are heavily distorted, so they can no longer be included in the chain. According to Fig. 15.*top*, though with 20% noise only about half of the embedded patterns (10 out of 20, with $R \approx 50\%$) are discovered, the precision P is still over 90%. Thus, the discovered chain can still reflect the general trend of the data. Moreover, note that in many cases we could "undo" much of the ill-effect of noise by simply smoothing the data, but that is orthogonal to the purpose of our demonstration.

We have demonstrated the that our algorithm is robust in the face of (a reasonable amount of) noise, with a synthetic dataset that contains an *intrinsic* chain. Conversely, we need to test if $k_{discovered}$ is small compared to k, when there is *no intrinsic* chain within the data, that is to say, are we robust to false positives?

To test this, as shown in Fig. 16, we constructed a synthetic time series with $k = 100$ repeated random-walk patterns.

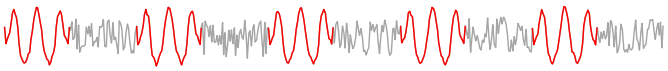

Fig. 16. A snippet of a synthetic time series with 100 repeated patterns.

As before, we added random noise to all the repeated patterns, so they look slightly different from each other. Unlike the data in Fig. 14, here the k patterns do *not* have an evolving trend. We constructed 100 such synthetic time series, and found that the average length of the discovered time series chain $k_{discovered}$ is 5.04, which is much smaller than $k = 100$. This result suggests that our algorithm is robust to discovering spurious chains, even in the face of frequent and dense motifs.

V. Conclusions And Future Work

We introduced *time series chains*, a new primitive for time series data mining. We have shown that chains can be efficiently and robustly discovered from noisy and complex datasets, to provide useful insights. In future work we plan to consider a more theoretical treatment of the properties of chains, and adapt/apply them to online problems, including prognostics.

Acknowledgment

We would like to acknowledge funding from MERL and from NSF IIS-1161997 II and NSF IIS-1510741. We especially want to thank Dr. John Michael Criley and Dr. Gregory Mason for their invaluable advice on the hemodynamics domain, and Dr. Matsubara for providing the *GoogleTrend* data.

References

[1] Bögel, T. and Gertz, M., 2015, June. Time will Tell: Temporal Linking of News Stories. In *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries,* pp. 195-204.

[2] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X. and Keogh, E. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *VLDB* 2008, 1542-52.

[3] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. and Bouchachia, A. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, *46,4* (2014): 1-37.

[4] Hao, M.C. et. al. Visual exploration of frequent patterns in multivariate time series. *Information Visualization*, *11.1* (2012): 71-83.

[5] Heldt, T., Oefinger, M.B., Hoshiyama, M. and Mark, R.G. Circulatory response to passive and active changes in posture. In *Computers in Cardiology* (2003):263-266. IEEE.

[6] Hoang, T., Choi, D. and Nguyen, T., On the instability of sensor orientation in gait verification on mobile phone. *In 12th IEEE International Joint Conference on e-Business and Telecommunications (ICETE), 4* (2015): 148-159.

[7] Krumme, C., Llorente, A., Cebrian, M., Pentland, A. and Moro, E. The predictability of consumer visitation patterns. *Scientific Reports* (1645). 2013.

[8] Li, Z., Han, J., Ding, B. and Kays, R. Mining periodic behaviors of object movements for animal and biological sustainability studies. *Data Mining and Knowledge Discovery*, (2012): 355-386.

[9] Lovallo, W.R., et, al. Blood pressure response to caffeine shows incomplete tolerance after short-term regular consumption. *Hypertension*, *43.4* (2004): 760-765.

[10] Matsubara, Y., Sakurai, Y. and Faloutsos, C. The web as a jungle: Non-linear dynamical systems for co-evolving online activities. In *Proc' of the 24th WWW* (pp. 721-731).

[11] McLoone J. URL retrieved September 6th 2016: blog.wolfram.com/2012/01/11/the-longest-word-ladder-puzzle-ever

[12] Moya, A. Tilt testing and neurally mediated syncope: too many protocols for one condition or specific protocols for different situations?. Eur Heart J. *30.18* (2009): 2174-2176.

[13] Mueen, A., Viswanathan, K., Gupta, C.K. and Keogh, E. The fastest similarity search algorithm for time series subsequences under Euclidean distance. URL retrieved Feb 2th 2017 www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[14] Murray, D. et, al. A data management platform for personalised real-time energy feedback. *EEDAL*, 2015

[15] Patel, P., Keogh, E., Lin, J. and Lonardi, S. Mining motifs in massive time series databases. In *Data Mining, Proceedings of the 2002 IEEE International Conference on.* 370-377.

[16] Ponganis, P.J., St Leger, J. and Scadeng, M. Penguin lungs and air sacs: implications for baroprotection, oxygen stores and buoyancy. *Journal of Experimental Biology*. (2015): 720-730.

[17] Shokoohi-Yekta, M. et. al. Discovery of meaningful rules in time series. In *Proc' of the 21th ACM SIGKDD* pp. 1085-1094.

[18] Silver, N. The signal and the noise: the art and science of prediction. Penguin UK, London. 2012.

[19] Smith J. "The Accidentally-on-Purpose History of Cyber Monday", URL retrieved February 5th 2017: www.esquire.com/news-politics/news/a23870/cyber-monday-online-shopping-4021548/

[20] Syed, Z., Stultz, C., Kellis, M., Indyk, P. and Guttag, J. Motif discovery in physiological datasets: a methodology for inferring predictive elements. *TKDD*, *4.1*(2010): 2.

[21] Williams, C.L., Sato, K., Shiomi, K. and Ponganis, P.J. Muscle energy stores and stroke rates of emperor penguins: implications for muscle metabolism and dive performance. *Physiological and Biochemical Zoology.85.2*(2011):120-133.

[22] Yan, R., Wan, X., Otterbacher, J., Kong, L., Li, X. and Zhang, Y. Evolutionary timeline summarization: a balanced optimization framework via iterative substitution. In *Proc' of the 34th ACM SIGIR* (2011): 745-754.

[23] 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*, pp. 1317-1322.

[24] Zhu, X. and Oates, T. Finding story chains in newswire articles. In Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on, pp. 93-100.

[25] 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. *ICDM 2016*, pp. 739-748.

[26] Supporting webpage: https://sites.google.com/site/timeserieschain/