Visualizing Variable-Length Time Series Motifs

Yuan Li[1] Jessica Lin[1] Tim Oates[2]
[1]George Mason University
[2]University of Maryland, Baltimore County
ylif@gmu.edu jessica@cs.gmu.edu oates@cs.umbc.edu

Abstract

The problem of time series motif discovery has received a lot of attention from researchers in the past decade. Most existing work on finding time series motifs require that the length of the motifs be known in advance. However, such information is not always available. In addition, motifs of different lengths may co-exist in a time series dataset. In this work, we develop a motif visualization system based on grammar induction. We demonstrate that grammar induction in time series can effectively identify repeated patterns without prior knowledge of their lengths. The motifs discovered by the visualization system are variable- lengths in two ways. Not only can the *inter-motif* subsequences have variable lengths, the *intra-motif* subsequences also are not restricted to have identical length—a unique property that is desirable, but has not been seen in the literature.

Keywords

Time Series, Motif Discovery, Grammar Induction, Visualization

1. Introduction

The task of frequent pattern mining is an important problem that has many applications. In addition to its own merit of summarizing and compressing data, it is also a pre-cursor to association rule mining [1]. Furthermore, in bioinformatics, it is well understood that overrepresented DNA sequences often have biological significance [6, 8, 10, 28, 32]. A substantial body of literature has been devoted to techniques to discover such patterns [1, 2], including sequential patterns on sequence data.

In previous work, we defined the related concept of "time series motifs" which are, informally, frequently repeated patterns. More specifically, a time series motif is a pattern that consists of two or more similar subsequences based on some distance threshold [18, 26]. Since then, a great deal of work has been proposed for the discovery of time series motifs [3, 4, 5, 18, 20, 21, 22, 23, 25, 26, 29, 30, 31]. Figure 1 shows an example of a time series motif in an insect behavior dataset [23]. This motif consists of four very similar subsequences of length 109.

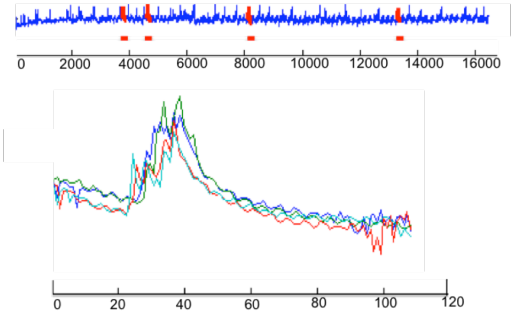

Figure 1. A motif of length 109 from an insect dataset. The top plot shows the entire dataset to present a context for the motif. The bars below the plot denote the locations of the motif instances. The original dataset is 16,384 in length.

Most existing work on time series motif discovery, however, suffers a limitation; that is, they require an input parameter, the motif length n, to be defined. This limits the search space for the algorithms; however, it also implies that the length of motifs (n) must be known in advance. In addition, frequent patterns of different lengths might co-exist within the same dataset. To find all significant patterns with unknown lengths, one would need to repeat the motif discovery algorithm several times—each time with a different window size. We also noticed that a well-cited motif discovery algorithm produces different sets of motifs even if the input length differs by just one (e.g. $n = 120$ vs. $n = 121$). This raises the question of whether such precision or *exact* motif discovery is necessary, or even desirable. In any case, it is better to have an algorithm that can automatically detect significant motifs of unknown, variable lengths, without exhaustively trying different subsequence lengths.

A few algorithms were proposed to discover motifs of variable lengths [21, 25, 30]; however, they either do so via post-processing, scale poorly, or quantize the whole data rather than considering overlapping subsequences, resulting in inaccurate and incomplete patterns found.

In the preliminary version of this work [16], we proposed an approach to discover variable-length motifs

by grammar induction [13, 14, 15, 24]. Grammars are compact generative representations of sets of objects. These objects are most often strings, as are generated by grammars in the Chomsky hierarchy, but can also be more complex structures like trees and graphs. In this work we are concerned exclusively with string grammars. Grammar induction (or grammatical inference, GI) is the process of learning a grammar given (at least) a sample of strings in the target grammar's language.

We argued that a grammar-based algorithm mitigates the aforementioned shortcomings of existing motif discovery algorithms, in addition to offering a unique advantage; that is, it reveals the underlying hierarchical structures and patterns of the whole data – something that cannot be achieved by existing motif discovery or sequential pattern discovery algorithms. Furthermore, a grammar-based method will allow a more natural mapping from data to rules [33], and can reveal hidden hierarchical structures in the data. There has also been increasing interest in grammar-based methods for feature extraction, classification and forecasting of time series [7, 11].

One novel contribution of this work is that we extended the notion of variable-length motifs. Intuitively, variable-length motifs is the set of motifs whose instances are of different lengths. As an example, in a dataset, there might be one motif A of length 100, and another motif B of length 250 (assuming no overlap). In this work, we allow subsequences for the same motif to have variable lengths as well, e.g. A might consist of a subsequence A_1 of length 109, and another subsequence A_2 of length 101, both of which are very similar to each other despite their different lengths.

One challenge we encountered is the evaluation of our methodology. This is partly due to the fact that no other work exists that achieve comparable goals, and partly due to the fact that it is difficult to evaluate the quality of *approximate* motifs. A common way to evaluate the approximate solution is to compare it with the exact solution. However, as mentioned earlier, it is debatable whether exact motif search is truly desirable; sometimes a little bit of error allowance might be more suitable for many real-world applications. As an example, for gait analysis or activity analysis, it is uncommon for an object, much less different objects, to repeat the *exact* same movements. In addition, since our algorithm allows both inter-motif and intra-motif subsequences to have different lengths, it is difficult to evaluate the patterns objectively, as we have deliberately removed the *exactness* from the solution.

We addressed the concerns above and implemented a variable-length motif visualization tool based on grammar induction. Unlike existing motif discovery algorithms, our grammar-based motif visualization system generates hierarchical rules from the data, and displays the patterns that form these rules. Clearly, the length of the rules is unrestricted, and so are the patterns (motifs) that correspond to the rules.

In summary, our work has the following novel contributions:

- Our grammar-based motif-discovery algorithm does not require the motif length to be known in advance. Our approach can find variable-length motifs simultaneously, with only one pass through the data. As we will demonstrate, not only can the *inter-motif* subsequences have variable lengths, the *intra-motif* subsequences also are not restricted to have identical length—a unique property that is desirable, but has not been seen in the literature. While the results produced by our algorithm are approximate solutions, it has been shown recently that in many applications, approximate solutions might be sufficient or even preferable due to efficiency [4].
- Our algorithm is both time- and space-efficient, and is suitable for streaming data.
- We developed a grammar visualization system that allows users to navigate the produced rules (i.e. the motifs), as well as the instances of the rules (i.e. the subsequences that form the motifs). Note that, unlike existing motif discovery algorithms, our notion of motifs are not defined based on some distance threshold. Rather, they are determined based on their symbolic *approximation*, and on the grammar rules.
- Our greedy approach is efficient; however, in some cases, we may desire better quality results at the cost of higher time complexity. We propose a search algorithm that finds better grammar, and allows users more flexibility in choosing between the efficiency and the effectiveness of the algorithm.

The rest of the paper is organized as follows. Section 2 discusses background and related work on time series motif discovery. Section 3 describes the greedy grammar induction algorithm that we adapted for our work. We describe our approach in Section 4, and demonstrate its utility with a case study on motif discovery by implementing a grammar visualization system in Section 5. We discuss the limitation of the greedy approach in Section 6, and propose a search algorithm to find better grammar. We conclude and discuss future work in Section 7.

2. Background and Related Work

In this section, we briefly discuss background and related work on motif discovery. We begin with definitions of time series:

Definition 1. *Time Series*: A time series $T = t_1,\ldots,t_m$ is an ordered set of m real-valued variables.

Since we are interested in finding local patterns, we consider time series subsequences as the basic unit:

Definition 2. *Subsequence:* Given a time series T of length m, a subsequence C of T is a subsection of length $n \leq m$ of contiguous position from p, that is, $C=t_p...t_{p+n-1}$ for $1 \leq p \leq m - n + 1$.

The subsequences can be extracted via the use of a sliding window:

Definition 3. *Sliding Window:* Given a time series T and a user-defined subsequence length n, all possible subsequences can be extracted by sliding a window of size n across T and considering each subsequence C_p, for $1 \leq p \leq m - n + 1$.

A time series motif I thus consists of a set of subsequences I_S. These subsequences, referred to as *intra-motif subsequences*, are similar to each other, but do not necessarily have identical lengths. We will explain what it means for the subsequences to be similar in a later section.

Our algorithm will find a set of motifs $\boldsymbol{M}$, each of which consists of its own set of *intra-motif subsequences* as described above. Subsequences from different motifs are called the *inter-motif subsequences*.

2.1 Related Work

In [18], we introduced the concept of time series motifs, and proposed a sub-quadratic algorithm based on hashing and matrix approximation to find exact motifs of a given length. Mueen et al. proposed the first tractable exact motif discovery algorithm, based on the notion of early abandonment [23]. Their algorithm is up to three orders of magnitude faster than the brute-force algorithm [23]. In some applications, it may be sufficient or even desirable to have a fast algorithm that can find *approximate* motifs [4, 5]. As an example, Chiu and Keogh proposed probabilistic motif discovery algorithm based on random projection [5]. Castro and Azevedo proposed a multiresolution motif discovery algorithm [4] that is both space and time efficient. Other approximate motifs algorithms exist [3, 5, 9, 20, 27, 30]; however, one common drawback of all these algorithms is that they require an input parameter for the motif length.

3. Greedy Grammar Induction

Given an input string, a common way to compress it or to induce grammar from it is by using a greedy approach: repeated patterns are merged, and replaced by a new symbol as soon as they are seen, thereby reducing the length of the original sequence and producing a hierarchical representation that summarizes the structure of the data. One example of such approach is *Sequitur*, a string compression algorithm that infers a context-free grammar from a sequence of discrete symbols [24]. Although simple in design, *Sequitur* has been shown to be competitive with state-of-the-art compression algorithms [24], maintaining its scalability even for large sequences. Moreover, *Sequitur* offers a unique advantage: it utilizes and identifies the hidden structure (recurring subsequences) in the input data sequence, requiring a relatively small memory footprint.

Sequitur works by maintaining two properties: digram uniqueness and rule utility [24]. The first property requires that no pair of consecutive symbols (terminals or non-terminals) appear more than once. When *Sequitur* reads a new symbol from the input sequence, the last two symbols of the sequence read so far—the new symbol and its predecessor symbol—form a digram [24]. A table that stores all existing digrams is maintained. If this new digram already exists in the digram table, i.e., it appears somewhere in the sequence already read, *Sequitur* uses a non-terminal to substitute these digrams, and, if such a rule does not exist, it forms a new grammar rule with the non-terminal on the left hand side. The second property, rule uniqueness, ensures that each grammar rule is used more than once, except for the top-level rule, since a grammar rule that occurs just once is not meaningful and should be removed. As an example, the input string S1: "12131213412" can be converted to the following grammar:

Table 1. Grammar generation process by *Sequitur* for the string "*12131213412*"

Grammar rule	Expanded Grammar rule
S1 -> BB4A	12131213412
A -> 12	12
B -> A13	1213

The top-level rule, $S1 \rightarrow BB4A$, denotes the input sequence seen so far. *Sequitur* is an online algorithm that generates the grammar incrementally as each symbol arrives. It is, therefore, ideal for streaming scenarios. It is both time- and space-efficient, requiring O(m) time to compress a sequence of size m, and a compressed sequence is of size O(m) in the worst case (i.e., no compression), and O(log$m$) in the best case.

This greedy grammar induction algorithm (and the many grammar induction algorithms in general) offers the following advantages: (1) it creates a compact summarization of data, from which hierarchical structures are identified; (2) recurring patterns are detected automatically, e.g., "1213" in the previous example; (3) these recurring patterns found can be of any length; and (4) it is suitable for streaming data since it constructs the grammars in an incremental fashion. These benefits suggest that grammar induction may be used to discover variable-length motifs.

4. Our Approach

Most grammar induction algorithms were originally designed for discrete data. However, time series are

real-valued data, requiring a pre-processing step to allow the application of a grammar-based algorithm.

In previous work, we introduced a time series symbolic representation called Symbolic Aggregate approXimation (SAX) [17, 19]. While there have been dozens of symbolic representations proposed for time series data, SAX has been shown to outperform existing methods. In addition, SAX has some unique, desirable properties such as dimensionality reduction, lower-bounding distance measures, and equiprobable symbols. For these reasons, we will utilize SAX to discretize our data.

All time series are z-normalized before discretization. SAX performs discretization by dividing the time series into w equal-sized segments. For each segment, their mean value is computed, and then mapped to a symbol according to a set of breakpoints that divide the distribution space into α equiprobable regions, where α is the alphabet size specified by the user. If the symbols were not equiprobable, some of the symbols would occur more frequently than others. As a result, we would inject a probabilistic bias in the process. The discretization steps are summarized in Figure 2.

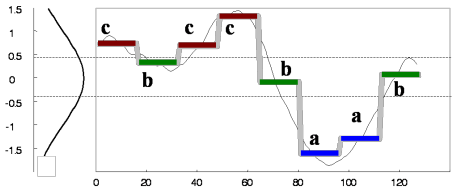

Figure 2. Example of SAX for a time series. The time series above is transformed to the string *cbccbaab*, and the dimensionality is reduced from 128 to 8.

There are two ways we may discretize a time series:

(a) Whole discretization: Convert each time series to a single SAX word. Figure 2 shows an example of whole discretization. The drawback with this approach is that it often does not capture enough local details in the data, which are essential for many pattern discovery tasks such as motif discovery.

(b) Subsequence discretization: Extract subsequences of length n from the time series, normalize the subsequences, and convert each subsequence into a SAX word. The result is a "bag" of SAX words generated from (all or selected) subsequences in the original data.

For subsequence discretization, we need to determine which subsequences to consider. It is generally a good idea to consider overlapping subsequences [18], since non-overlapping subsequences would result in too much loss in information. This can be achieved by using a sliding window of length n across the time series. For most motif discovery algorithms, this is also the length of the motif to be discovered. For our algorithm, however, n is just the initial window length; the algorithm grows the patterns automatically. We discretize each subsequence individually using SAX, and then concatenate them to form one single sequence. The following shows what a transformed sequence, S, might look like, in which consecutive, overlapping subsequences are delimited by spaces:

$$S = 113\ 113\ 123\ 122\ 134\ 113\ 113\ 113\ 123\ldots$$

Note each SAX word is equivalent to a terminal symbol. We already mentioned the drawback of whole discretization; that is, too much detail is lost. A natural question to ask is, why not discretize individual points, and convert a time series into a string of equal length? No doubt such an approach is more efficient than subsequence discretization, and allows more straightforward adaptation of existing sequential pattern mining algorithms. However, since time series are typically very noisy, particularly those that are generated at a fast rate (e.g. ECG), discretizing each time point into a symbol would give each time point equal weight, including the noise. In the contrary, the "aggregating" feature of SAX smooths out the subsequences and removes the noise.

4.1 Numerosity Reduction

One novel contribution of this work is that it allows intra-motif subsequences to have different lengths. We achieve this by employing *numerosity reduction* during the discretization process [18]. Since neighboring subsequences are likely to be similar to each other, their SAX words will likely be the same. If a string occurs many times consecutively, instead of recording every single string, we record only the first occurrence of it, but note the offset of this first occurrence. If the same string appears again after the appearance of some other string(s), then we can record it again. The string S thus becomes:

$$S1 = 113\ \sout{113}\ 123\ 122\ 134\ 113\ \sout{113\ 113}\ 123$$
$$= 113_1\ 123_3\ 122_4\ 134_5\ 113_6\ 123_9$$

The subscripts denote the offsets of the first occurrences of the strings. Doing so offers the benefits of eliminating trivial matches [18], and reducing storage space. As we will demonstrate later, it turns out that numerosity reduction is the key that makes variable-length motif discovery possible.

4.2 Grammar Induction on SAX Words

Once we transform the time series into a discrete sequence consisted of SAX words, we can induce grammar, e.g. by *Sequitur*, on the SAX word sequence. Each string delimited by ' ' represents one subsequence (or a series of subsequences mapped to the same string), and is treated as a terminal symbol, an atomic unit for patterns. One possible grammar rule that can be generated from the above string $S1$ is $A \rightarrow$ 113 123. The

patterns occur in *S1[1:3]* and *S1[6:9]*, respectively (note the difference in pattern lengths due to numerosity reduction). Since a grammar rule represents subsequences (patterns) that occur more than once, it can be regarded as a motif.

4.3 Mapping Rules to Patterns

Since we discretized the data, we now need to map the rules and strings back to the time series subsequences to identify the motifs. The number of rules generated can be large and, similar to association rules mining [1], not all rules are interesting or important. Several refinement steps can be performed on the grammar rules. The choice of the appropriate refinement depends on the pattern discovery task at hand. In general, we would want to rank the rules by their "interestingness." Several ranking criteria can be considered, such as the frequency of patterns represented by the rules, the lengths of the rules, the pattern variation, the amount of overlap between patterns, etc. In this work, we rank the rules by the rule lengths and their frequency of occurrences. We defer the study of other ranking criteria to future work.

4.4 Experimental Evaluation

Our experimental evaluation consists of two parts: (1) to demonstrate that it is indeed meaningful to find time series "grammar"; (2) to demonstrate that grammar induction in time series can help identify variable-length motifs. The experiments shown in this paper are subjective. The reason is that, to the best of our knowledge, there does not exist any other time series grammar induction algorithm, or variable-length motif discovery algorithm that we can meaningfully and fairly compare our technique to. As a result, our first experimental evaluation will focus on demonstrating the *potential* of using grammar induction for time series pattern discovery.

Figure 3 shows an ECG dataset, from which the grammar is induced. As the figure shows, the raw time series is 1500 in length. The initial window length for subsequences is 50 (note this is just the initial window that the patterns grow from). The SAX parameters used for this example are w = 3 and α = 2. The grammar for the ECG dataset is shown in Table 2. The column "Usage" records the number of occurrences for each rule/pattern.

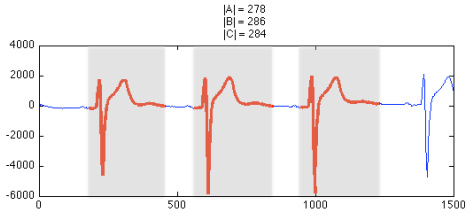

Figure 3. ECG time series. The patterns extracted from grammar rule R5 (below) are shown. The initial length is 50.

The instances of rule R5, which represent 3 individual heartbeats, are highlighted in Figure 3. The lengths of the subsequences are 288, 286, and 284, respectively, with the initial window length of 50. These three subsequences are all mapped to the same string "*221 211 212 112 122 112 122 121 221 211 212 112 122 121 221 211 221 211*." Notice the subsequences are of different lengths, and the lengths are not the same as the SAX string length. This is due to numerosity reduction – each SAX "word" can correspond to multiple (consecutive) subsequences, if their respective SAX strings are identical and contiguous. The subsequences can be seen as a "motif", which we will discuss more in the next section. Numerosity reduction makes motif discovery more robust, as we are matching patterns based on their shapes, even if they do not have the exact same lengths.

Table 2. Grammar found for the ECG dataset

Usage	Grammar Rule	Expanded
0	R0 -> R1 R2 121 R3 R3 R4 R5 R6 R5 R2 211 R6 122 R5 R7 R8 121 R7	221 211 221 211 221 121 122 112 122 112 122 121 221 211 212 112 122 221 211 212 112 122 221 211 212 112 122 121 221 211 212 112 122 121 221 211 221 211 221 211 212 112 122 121 221 211 212 112 122 121 221 211 212 112 122 121 221 211 212 112 122 112 122 121 221 211 212 112 122 121 221 211 212 112 122 221 211 212 112 122 112 122 121 221 211 212 112 122 121 221 211 221 211 221 211 212 112 122 112 122 121 221 211 212 112 121 221 211 212 112 122 112 122 121
20	R1 -> 221 211	221 211
5	R2 -> R1 221	221 211 221
2	R3 -> 122 112	122 112
4	R4 -> R9 R10	122 121 221 211 212 112 122
3	R5 -> R11 R4 121 R2 211	221 211 212 112 122 112 122 121 221 211 212 112 122 121 221 211 221 211
2	R6 -> R10 121 R8	221 211 212 112 122 121 221 211 212 112
2	R7 -> R11 R9	221 211 212 112 122 112 122 121
10	R8 -> R1 212 112	221 211 212 112
6	R9 -> 122 121	122 121
11	R10 -> R8 122	221 211 212 112 122
5	R11 -> R10 112	221 211 212 112 122 112

The rule R5 is composed of three other rules, R11, R4, and R2, along with two other short patterns "121" and "211." We plotted one instance from each of the three rules, and the result is shown in Figure 4. We can clearly see that the pattern for R5 is composed from the three sub-patterns for R11, R4, and R2. The grammar indeed reveals some hierarchical structure in the data.

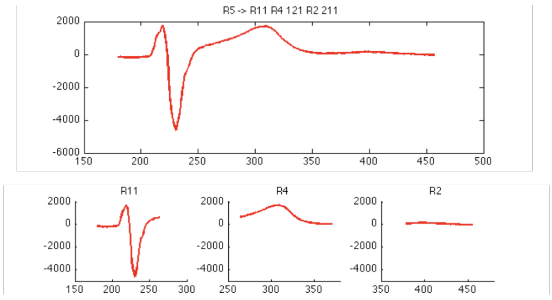

Figure 4. The decomposition of one of the motif instances.

We already demonstrate in Figure 3 that a motif can be found in the ECG dataset by examining the grammar rules generated. While only one motif is shown (which

are represented by 3 instances) in the figure, Figure 4 shows three shorter-length patterns (motifs) that make up the longer pattern. These three motifs are captured by rules R11, R4, and R2, respectively.

5. Visualization

Depending on the complexity of the dataset and the choices of SAX parameters, the number of rules generated by our grammar induction algorithm can be large. In addition, since the algorithm produces approximate results, it is desirable to have a mechanism that allows users to easily navigate through the patterns found. To best evaluate and validate the results, we implemented a grammar/motif visualization system based on the methodology proposed. Figure 5 shows a screen shot of the visualization tool.

The upper left panel shows the input time series, the "winding" dataset from The UCR Time Series Data Mining Archive [12]. The upper right corner is the "control panel", which allows the user to enter various options such as SAX parameters, the initial window size, and sliding window option. If the sliding window option is selected, then the time series will be discretized by subsequence discretization, otherwise whole discretization will be employed. Each button denotes a step in the algorithm: Load Data, Discretize, Generate Grammar (motifs). This allows the user to go back and change parameters without having to reload the dataset.

Once the data is loaded (and shown in the upper left panel), the time series is discretized based on the parameters and sliding window option. The SAX string (delimited by '*') is shown in the panel right below the input time series panel. Grammar can then be induced from the SAX string, and the production rules are shown in the lower left panel. As the user scrolls down the list of rules, the rule pointed to by the cursor will be highlighted, and the subsequences that form the rule will be plotted in the lower right panel. The corresponding subsequences will also be highlighted on the input time series. In this example, the length of the dataset is 2,500. The SAX parameters are $\alpha = 3$ and $w = 4$, the initial window size is set to 100, and the sliding window option is turned ON.

Each rule represents one motif, which consists of at least two subsequences (the rule utility policy governs that at least two matching subsequences are needed in order to form a rule). These intra-motif subsequences are similar in the sense that they all share the same symbolic approximation and grammar rule. While it is obvious that the length of each motif depends on its corresponding grammar rule (hence the variable-length motifs), the subsequences within each motif can also have different lengths due to the numerosity reduction feature.

Placing the cursor on Rule #12: *R12 → 2221 2321 R46 3321 3311* reveals the motif shown in the lower right panel in Figure 5. Following the rule is the number of occurrences for this pattern (3), the expanded rule (*2221 2321 2331 2321 3321 3311*)—which simply means that the subsequences for the motif all have this SAX representation—and the offsets for these subsequences on the *SAX string*. Their offsets on the original time series can be easily computed. The subsequences are approximately 140 in length, although one is slightly shorter than the other two. There are a total of 54 rules generated for this example. In the same run of the algorithm, another motif found is shown in Figure 6 (Rule #29). The subsequences for this motif have length of approximately 370. This and the previous example demonstrate the algorithm's ability to find variable-length motifs – motifs with varying lengths are discovered simultaneously.

Next we will demonstrate the non-sliding-window option (i.e. whole discretization). The screenshot is shown in Figure 7. For this option, the original time series of length 2,500 is discretized into one long SAX string, with $\alpha = 3$, and $w = 200$. While the non-sliding-window option usually finds fewer patterns (i.e. some patterns might be missed because they are broken up in two segments, or they might be shifted in their respective segments), they also result in fewer rules due to shorter, less complicated string. There are only 21 rules generated with the non-sliding-window option, compared to 54 rules with sliding windows. For this example, we are able to find a motif of length about 480.

Figures 8 and 9 show another example on a different dataset. The insect behavior dataset introduced in [23] has length of 18,667. The initial window size is again set to 100. The SAX parameters are $\alpha = 2$, and $w = 3$. Since the dataset is long and noisy, having a smaller alphabet size reduces the complexity of the resulting string, which in turn produces better results. There are a total of 60 rules. The first motif, shown in Figure 8, consists of two subsequences of approximately 750 in length. The second motif, shown in Figure 9, consists of two subsequences that are about 470 and 490, respectively, in length. This result is impressive, considering the initial window length is only 100, and the SAX approximation is very coarse.

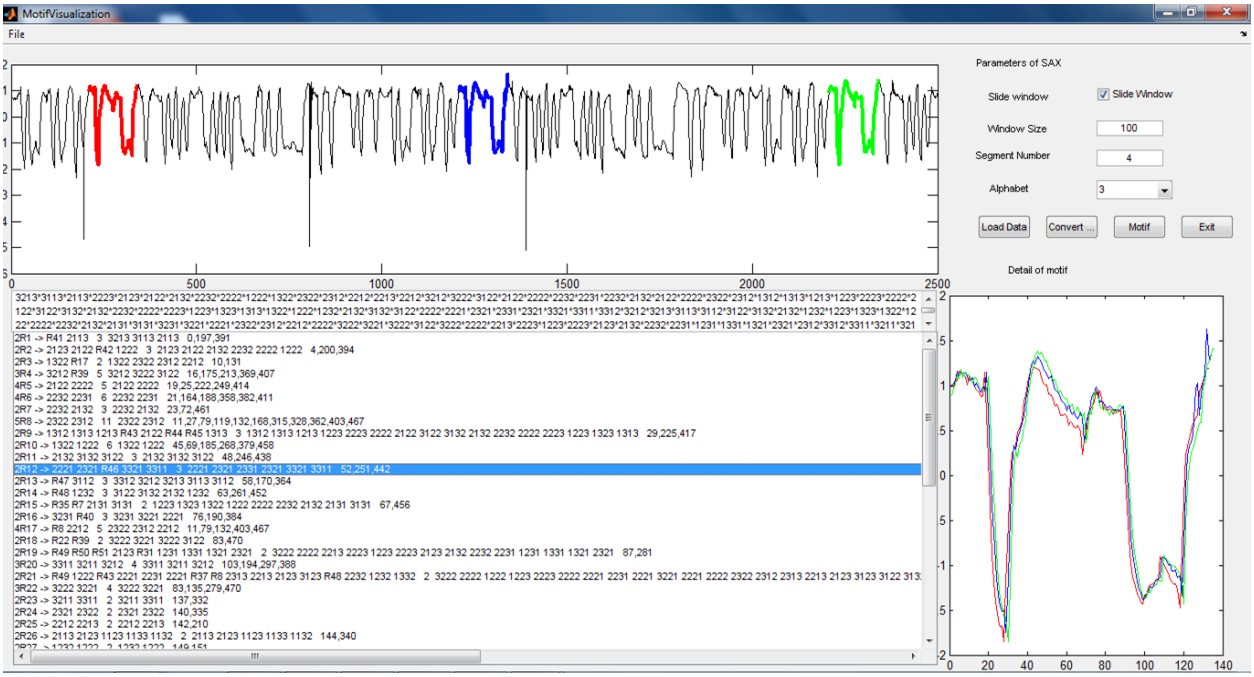

Figure 5. Screenshot of the grammar/motif visualization system on the *winding* dataset. The motif found is approximately 140 in length.

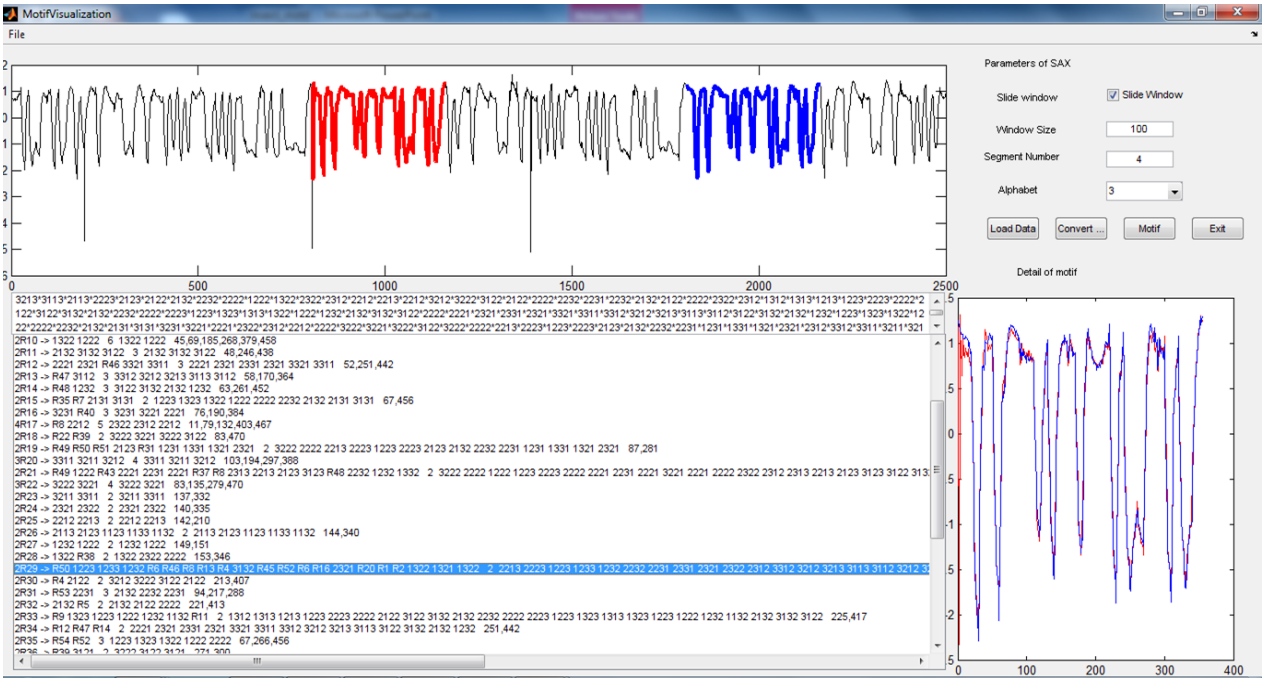

Figure 6. Screenshot of the grammar/motif visualization system on the *winding* dataset. The motif length is about 370.

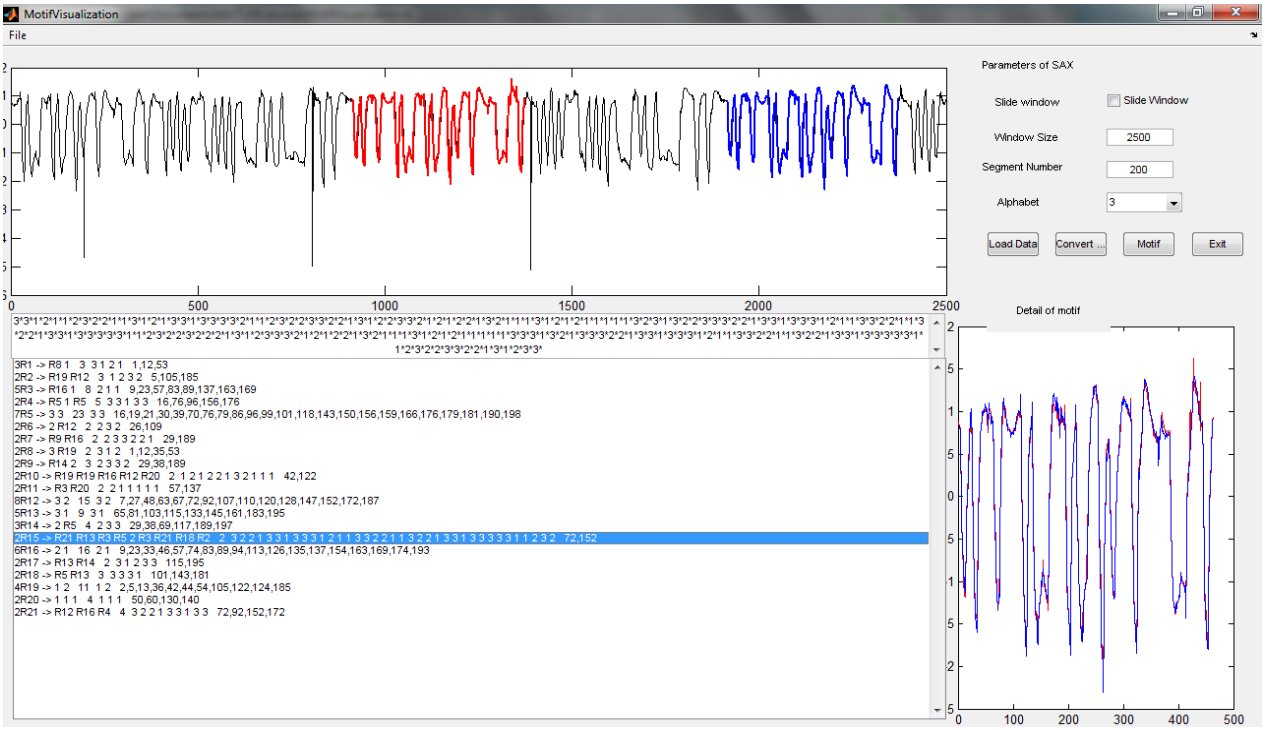

Figure 7. Screenshot of the grammar/motif visualization system on the *winding* dataset, without sliding window option. The motif length is about 470.

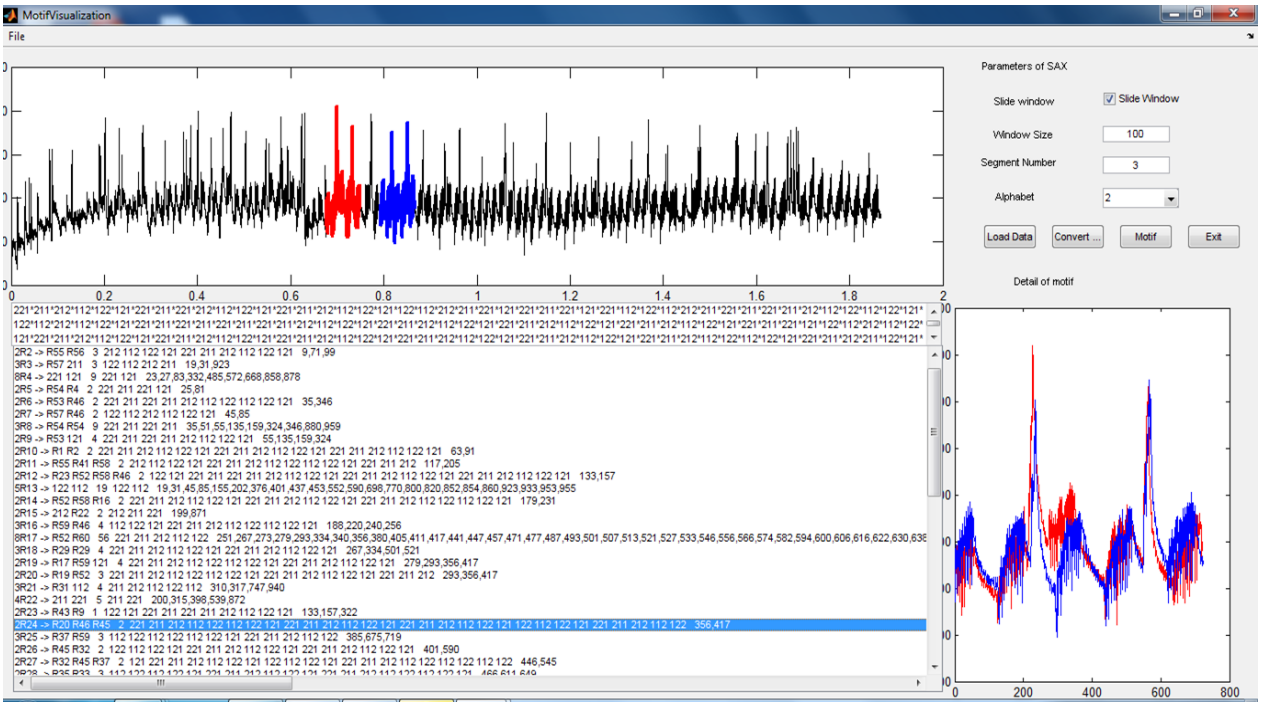

Figure 8. Screenshot of the grammar/motif visualization system on the *insect* dataset. The motif length is about 750

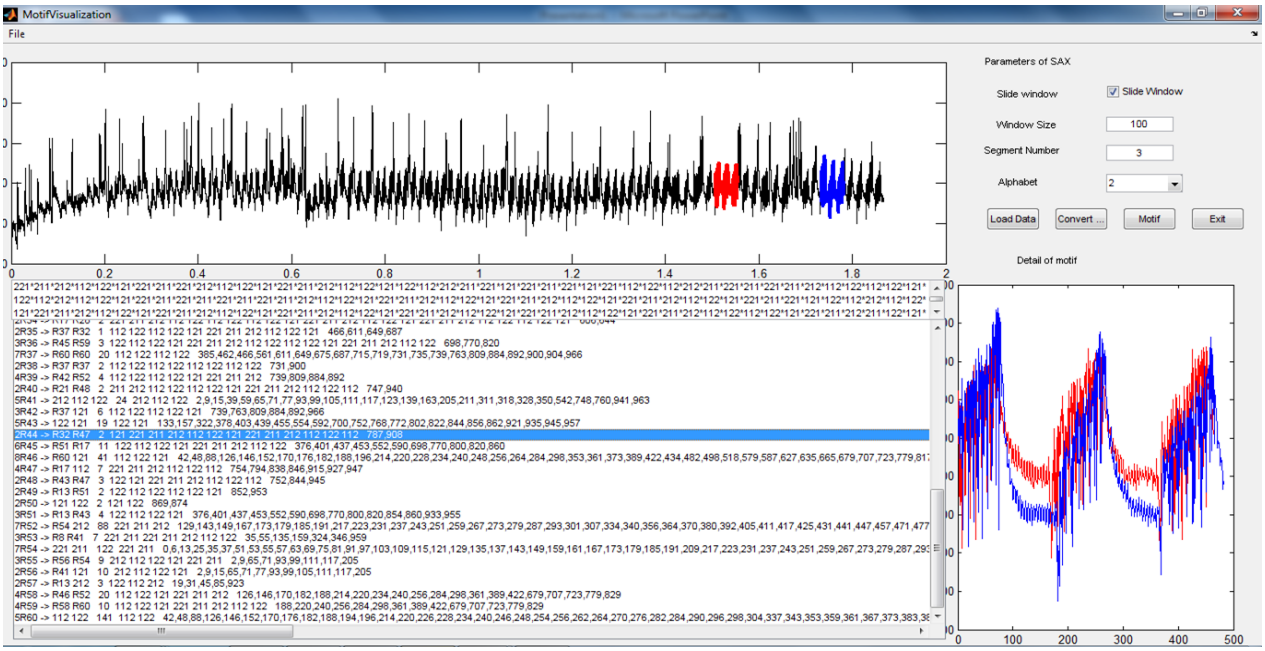

Figure 9. Screenshot of the grammar/motif visualization system on *insect* dataset. The motif length is between 470 and 49

Some of the rules might be redundant or uninteresting to users. While such rules may be eliminated by employing the chosen filtering criteria in the post-processing step as described in Section 4.3, the visualization capability allows the user to quickly examine and explore the rules. In this version, since the goal is to provide the user a complete picture of the rules found, no rule was eliminated in the visualization tool.

5.1 Parameter Setting

As shown in our experiments, the parameters do not need much tuning. For most cases, we chose the arbitrary window size of 100, $\alpha = 2$ or 3, and $w = 3$ or 4. We find that even though the SAX approximation may seem coarse, the motifs found are satisfactory. As mentioned, if the dataset is long and noisy, it may be better to choose a small alphabet size such as $\alpha = 2$, which reduces the complexity of the data. This might result in more erroneous matches; however, the visualization capability makes it very easy to visually prune off these matches.

Note that without numerosity reduction, the pattern growth would be limited. We are more likely to find longer patterns if numerosity reduction is used. On the other hand, with numerosity reduction, there is a higher chance for erroneous matches (due to the flexible lengths). Nevertheless, we find that in our experiments, having the numerosity reduction feature generally produces better results.

6. Finding Better Grammar

The greedy approach offers many desirable properties and is efficient for finding grammars, hierarchies, and regularities in data. Nevertheless, in some cases the inherent nature of the algorithm prevents it from meeting our expectations. Specifically, it is not surprising that the grammar inferred by *Sequitur* is not minimal because it is an on-line, greedy algorithm. In fact, it has been shown that finding the minimal grammar is an NP-hard problem [15].

The following example illustrates this limitation of *Sequitur*. Consider the input string *S2* = $(11112131131)^4$, which consists of four copies of *11112131131*. The grammar rules generated by *Sequitur* are shown in Table 3:

Table 3. Grammar generation process by Sequitur for the string $(11112131131)^4$.

Usage	Grammar Rules	Expanded
0	R0 -> R1 R1 R2 R2 R2 R3 3 1	1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1
12	R1 -> 1 1	1 1
3	R2 -> R3 R4 R1 1	2 1 3 1 1 3 1 1 1 1 1
4	R3 -> 2 1 R4	2 1 3 1 1
7	R4 -> 3 R1	3 1 1

Sequitur does not identify the fact that the string consists of 4 copies of the substring "*11112131131*", nor does it identify the longest repeating pattern: "*11112131131 11112131131*". This is because *Sequitur* generates a grammar rule as soon as a match is found. When *Sequitur* sees the last symbol '*1*' of the first copy of "*11112131131*", and the first symbol '*1*' of the next "*11112131131*", it determines that *11* is a match for the existing pattern *R1*, and replaces *11* with *R1*. Simply stated, *Sequitur* makes the decision on grammar rule inference using the partial knowledge it has so far, hence reaching a local optimum.

To ensure that merging is the best action *globally*, we would need to keep track of all possibilities: merge or

not merge with each matching diagram. The exhaustive approach is clearly infeasible, as the number of possible actions grows exponentially with the number of matching digrams. Our goal is thus to improve upon the greedy approach, without having to exhaustively consider all options. There are many ways to compare different grammars. For our purposes, we use the size of the top-level grammar rule as the measure for its quality. We argue that the size of the top-level rule of a grammar is a good indication of how well the sequence is compressed.

We extend the notion of digrams, and define a *trigram* to be a set of three symbols that appear consecutively in a sequence. Each trigram can be decomposed into two digrams. For instance, the trigram *abc* is composed of two digrams *ab* and *bc*. We call *ab* the left digram, *L,* and *bc* the right digram, *R*.

There are four different types of trigrams we may encounter:

- *Type I*: *L* already exists in the sequence processed so far; *R* does not.
- *Type II*: *L* does not exist in the sequence processed so far; *R* already exists.
- *Type III*: Both *L* and *R* exist in the sequence processed so far.
- *Type IV*: Neither *L* nor *R* exists in the sequence seen so far.

For both Type I and Type III trigrams, in which the left digram *L* already exist in the digram table, there are two possible courses of actions. For Type I trigrams, we can either replace *L* or keep going without replacing *L*. For Type III trigrams, we can either replace *L* or replace *R* (note that replacing *R* is equivalent to *not* replacing *L*). *Sequitur* always replaces *L* for both cases which, as we have already seen, can prevent longer patterns from being recognized in the future.

Having defined trigrams, we can envision a search tree, in which each node denotes the substring processed so far. Figure 10 shows the search tree for the string "*11112131131*". The leaf nodes (with bolded rectangles) store all the possible top-level grammar rules. A node is split when we encounter a Type I or a Type III trigram, and the branches are labeled with the new production rule generated by merging the left or right digram. This results in two new subtrees, each of which represents different grammar inference generated from the action described by the split.

For example, the root node "*11112*…" is split into 2 nodes. Its left child is "*AA213113*…", obtained after merging the digrams "*11*", replacing both occurrences of "*11*" with a new non-terminal "A", and scanning until it sees "*113*", which is a Type III trigram (both "*11*" and "*13*" exist already). Note that *Sequitur* would greedily parse "*113"* as (11)3, i.e. merging "*11*". However, in our search tree, we split the node again, with the left child representing the merging of "*11*", and the right child representing the merging of "*13*". The problem of finding the best grammar(s) thus becomes searching for the best path(s) to follow. *Sequitur* always follows the leftmost path (i.e. the circled path in the figure), which is often not the path that produces the best result. In this example, the path that produces the best grammar is highlighted.

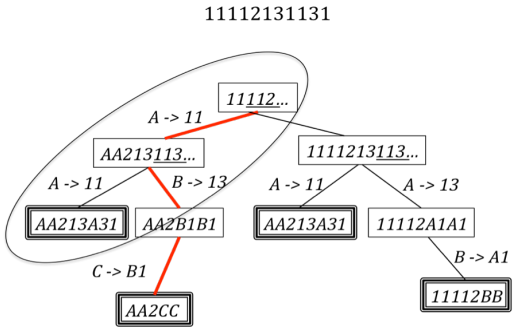

Figure 10. Grammar search tree for the string "11112131131"

Exhaustive search is clearly intractable, as there are approximately 2^d paths in the tree, where d = (# of Type I trigrams) + (# of Type III trigrams). As a result, we cannot actually build the entire tree. We propose a simple random search algorithm. At each node, we choose a child at random. In other words, whenever we read in a Type I or Type III trigram, we flip a coin and decide whether to merge or not to merge (for Type I), or to merge the left digram or the right digram (for Type III). With the exponential search space, this algorithm will likely require many iterations before we find one good path. We propose to always visit the leftmost path, which will lead us to the same grammar generated by *Sequitur*. We call this the *base grammar*. We then randomly search the rest of the tree for a grammar better than the base grammar. We divide the search space into k subtrees, where $k = 2^L$, and L is a user-defined parameter denoting the tree level where the random search starts. For example, if $L = 2$, then $k = 4$, and random search is performed in the subtrees rooted at the nodes on level 2 (assuming the root node is at level 0). For each subtree except for the first one (in which we follow the leftmost path), we repeat the search r times. The parameters L and r can be adjusted according to the computing resources available. Once all subtrees are searched, we can either stop the search and return the best grammar found (including the base grammar), or we can narrow the search among a few subtrees that return the highest counts of grammars better than the base grammar.

Table 4 shows the resulting grammar on the same string shown in Table 3, using random search. The parameter L is set to 3, and r is set to 50 (i.e., partition the search space into 8 subtrees, and run 50 random searches in each subtree except for the first one). The

best grammar was found after 66 iterations. Out of 350 random iterations, we found 17 grammars that are better than the base grammar.

Table 4. Grammar generated by our random search. The fact that the string consists of 4 copies of the substring "11112131131" is identified.

Usage	Grammar Rules	Expanded
0	R0 -> R1 R1	1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1
2	R1 -> R2 R2	1 1 1 1 2 1 3 1 1 3 1 1 1 1 1 2 1 3 1 1 3 1
4	R2 -> R3 R3 2 1 3 R4	1 1 1 1 2 1 3 1 1 3 1
12	R3 -> 1 1	1 1
4	R4 -> R3 3 1	1 1 3 1

We compare the grammars induced by *Sequitur* and random search algorithm, respectively, on longer sequences. ECG SAX is a SAX word sequence, converted from ECG data shown in Figure 3. The sliding window length is 100, and the SAX parameters are $a = 2$, $w = 3$. The second dataset is a DNA sequence. For both datasets, random search was run with 3200 iterations. Multiple grammars were found, in both cases, that are better than the ones from *Sequitur*. However, for each dataset, only the best (shortest) one is recorded, in terms of the length of the top-level rule. From the results, we can see that while random search *can* find better grammars than *Sequitur* with enough iterations, the improvement is marginal. It is unclear how close we are to finding the *best* grammar, since we did not examine all possible paths. We are not claiming that random search is the best search option; in fact, we believe that better approaches exist. Our experiments simply show that it is possible to improve the greedy grammar search by using a tree search algorithm. In fact, the tree-search-based method allows more flexibility in choosing between the efficiency of the algorithm and the quality of the results, whereas the traditional greedy approach produces fast but suboptimal result.

Table 6. Comparison between the grammars induced by *Sequitur* and random search

	# tokens	Sequitur first rule length	Random search first rule length
ECG SAX	130	20	17
DNA	149	57	51

7. Conclusion and Future Work

In this work, we propose a methodology to find approximate variable-length time series motif using a grammar-based compression algorithm. Our algorithm offers the advantage of discovering hierarchical structure, regularity and grammar from the data. The visualization tool further allows the user to navigate through and explore different motifs of variable lengths that co-exist in the dataset. Our results show that the grammar-based approach is able to find some important motifs and suggest that the new direction of using grammar-based algorithms for time series pattern discovery is worth exploring. We also proposed a search heuristic to improve the quality of induced grammar.

Many future directions are possible. We would like to analyze the time complexity for the random search algorithm. The time complexity can be controlled by limiting the number of iterations. However, with long sequences, we would possibly need an untenably large number of iterations in order to make some impact on the results. Though, in the worst case, the algorithm resorts to returning the same grammar as *Sequitur*. We would like to analyze and approximate the number of iterations needed and the fraction of all paths that result in better grammars than the base grammar. We can also allow different biases on the random search. For example, it may be possible to adjust the bias dynamically based on the quality of grammar found so far. Furthermore, we would like to explore other search heuristics. For the visualization tool, we would like to enhance the rule ranking and filtering feature. It is possible to prune off the false positives by calculating the distances between the motif instances.

8. References

1. R. Agrawal, T. Imielinski, and A. Swami. (1993). Mining Association Rules Between Sets of Items in Large Databases. In Proc. of the 1993 ACM SIGMOD Int'l Conference on Management of Data. Washington, D.C. May 26-28, pp. 207-216
2. R. Agrawal and Ramakrishnan Srikant. (1995). Mining Sequential Patterns. In Proc. of the 11th Int'l Conference on Data Engineering, Taipei, Taiwan, March.
3. P. Beaudoin, M. van de Panne, P. Poulin and S. Coros. (2008) Motion-Motif Graph*s*. In Proc. of the Symposium on Computer Animation.
4. Castro, N. and Azevedo, P. (2010). Multiresolution Motif Discovery in Time Series, in Proc. of the SIAM Int'l Conference on Data Mining. Columbus, Ohio, pp. 665-676.
5. Chiu, B. Keogh, E., & Lonardi, S. (2003). Probabilistic Discovery of Time Series Motifs. In the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Aug 24-27, 2003. Washington, DC. pp 493-498.
6. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. (1998). Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids: Cambridge University Press.
7. D. Eads, E. Rosten, D. Helmbold. (2009). Grammar-guided Feature Extraction for Location-Based Object Detection. British Machine Vision Conference. Queen Mary, University of London. London, UK. September 11.
8. A. Gionis and H. Mannila. (2003). Finding Recurrent Sources in Sequences. In proceedings of the 7th Int'l Conference on Research in

Computational Molecular Biology. Berlin, Germany. pp. 123-130

9. T. Guyet, C. Garbay and M. Dojat. (2007). Knowledge Construction From Time Series Data Using a Collaborative Exploration System. Journal of Biomedical Informatics 40(6): 672-687.
10. D. He. (2006). Using Suffix Tree to Discover Complex Repetitive Patterns in DNA Sequences, The 28th Annual Int'l Conference of the IEEE Engineering in Medicine and Biology Society. New York, NY. August 30 - September 3.
11. E. Keogh. Personal Communications.
12. E. Keogh. The UCR Time Series Data Mining Archive.
13. R. Ladner. (2003). Enhanced Sequitur for Finding Structure in Data. In Proceedings of the 2003 Data Compression Conference. March 25-27, Snowbird, UT. pp 425-.
14. P. Langley. (1995). Simplicity and Representation Change in Grammar Induction. Technical Report.
15. E. Lehman. (2002). Approximation Algorithms for Grammar-Based Data Compression, PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
16. Li, Y. and Lin, J. (2010). Approximate Variable-Length Time Series Motif Discovery Using Grammar Inference. In Proceedings of the Tenth international Workshop on Multimedia Data Mining. Washington, D.C., July 25 - 25.
17. J. Lin, E. Keogh, S. Lonardi, and B. Chiu. (2003). A Symbolic Representation of Time Series, with Implications for Streaming Algorithms, Workshop on Research Issues in Data Mining and Knowledge Discovery, the 8th ACM SIGMOD. San Diego, CA.
18. J. Lin, E. Keogh, P. Patel, and S. Lonardi. (2002). Finding Motifs in Time Series, the 2nd Workshop on Temporal Data Mining, the 8th ACM Int'l Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canada. pp. 53-68.
19. J. Lin, E. Keogh, W. Li, and S. Lonardi. (2007). Experiencing SAX: A Novel Symbolic Representation of Time Series. Data Mining and Knowledge Discovery Journal.
20. J. Meng, J.Yuan, M. Hans and Y. Wu. (2008). Mining Motifs from Human Motion, Proc. of EUROGRAPHICS.
21. D. Minnen, T. Starner, I. Essa, C. Isbell. (2006). Activity Discovery: Sparse Motifs from Multivariate Time Series. Snowbird Learning Workshop, Snowbird, Utah, April 4-7.
22. D. Minnen, C.L. Isbell, I. Essa, and T. Starner. (2007). Discovering Multivariate Motifs using Subsequence Density Estimation and Greedy Mixture Learning. In the 22nd Conf. on Artificial Intelligence. Vancouver, B.C., July 22-26.
23. A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover. (2009). Exact Discovery of Time Series Motifs. In proceedings of the 2009 SIAM International Conference on Data Mining. April 30-May 2. Sparks, NV.
24. C.G. Nevill-Manning and I.H. Witten. (1997). Identifying Hierarchical Structure in Sequences: A linear-time algorithm. Journal of Artificial Intelligence Research, 7, 67-82.
25. T. Oates. (2002). PERUSE: An Unsupervised Algorithm for Finding Recurring Patterns in Time Series. In proceedings of the International Conference on Data Mining. Maebashi City, Japan. Dec 9-12. pp. 330-337.
26. P. Patel, E. Keogh, J. Lin, and S. Lonardi. (2002). Mining Motifs in Massive Time Series Databases. In Proceedings of the 2002 IEEE International Conference on Data Mining. Washington, DC. pp. 370-377.
27. S. Rombo and G. Terracina. (2004). Discovering Representative Models in Large Time Series Databases, Proc. of the 6th International Conference on Flexible Query Answering Systems, pp. 84–97.
28. R. Staden. (1989). Methods for Discovering Novel Motifs in Nucleic Acid Sequences. Computer Applications in Biosciences. vol. 5. pp. 293-298.
29. Y. Tanaka and K. Uehara. (2004). Motif Discovery Algorithm from Motion Data. In proceedings of the 18th Annual Conference of the Japanese Society for Artificial Intelligence. Kanazawa, Japan. June 2-4.
30. Y. Tanaka, K. Iwamoto, and K. Uehara. (2005). Discovery of Time-Series Motif from Multi-Dimensional Data Based on MDL Principle. Mach. Learn. 58, 2-3, 269-300.
31. H. Tang and S.S. Liao. (2008). Discovering Original Motifs with Different Lengths From Time Series. Know.-Based Syst. 21, 7, 666-671.
32. M. Tompa and J. Buhler. (2001). Finding Motifs Using Random Projections. In proceedings of the 5th Int'l Conference on Computational Molecular Biology. Montreal, Canada. Apr 22-25. pp. 67-74
33. M.L. Wong and K.S. Leung. (2000). Data mining using grammar based genetic programming and applications. In: Genetic programming, vol. 3. The Netherlands: Kluwer Academic Publishers.