Finding the Intrinsic Patterns in a Collection of Time Series

Anke Schweier, Frank Höppner

Ostfalia University of Applied Sciences
Dept. of Computer Science, D-38302 Wolfenbüttel, Germany

Abstract. With most approaches to pattern discovery in time series the notion of a pattern is defined a priori and then an algorithm for the efficient discovery of patterns is proposed. But finding the *intrinsic patterns* in a collection of time series may require a search for the best pattern representation, too. For one dataset it may be important to consider absolute points in time, for other datasets only the shapes may be of interest. With some datasets reoccurring subseries match the pattern closely and with others only loosely. We propose an MDL-based approach to search not only for patterns, but for the *intrinsic pattern representation*. The preliminary results of this unsupervised method are promising, because in the examined (supervised) datasets the identified representations led to patterns that discriminate between classes.

1 Introduction

The ubiquity of sensor technologies (e.g. in mobile phones) and the affordability of storage capacities attract more and more companies to continuously record and store data. Prominent examples are 'open microphones' in new Android phones to continuously identify user commands or to create a play-list of all songs you (incidentally) came across today (www.shazam.com). Many daily activities (like driving a car) are potentially interesting (for the car manufacturer or insurance company) such that one may suspect that your car "likely has a black box spying on your already" [3]. Without necessarily sharing the visions behind these applications, the examples demonstrate that temporal data (such as time series) become increasingly popular and common.

To explore a collection of time series, it is helpful to summarise the series somehow, that is, to identify subseries that repeat often (within the same series or across different series). Such patterns can, however, be perceived in many different ways: using absolute time points ("driving to work at 6:30 in the morning"), shapes ("sharp increase followed by sudden drop"), etc. And to identify reoccurring patterns another important aspect is the accuracy of the matching step: do we have to carve out patterns exactly or only vaguely in order to find repeating occurrences? Most approaches from the literature define the type of patterns they are going to discover, but do not consider a search for the best pattern type. In this preliminary work, we investigate the possibility of identifying the *intrinsic patterns* in a collection of time series, that is not only the patterns but the conditions under which they expose best, with the help of the minimum description length (MDL) principle.

The paper is organised as follows: In the next section we review MDL and related approaches from the literature. Then, Section 3 gives an overview of the approach and the considered pattern representations. The algorithmic approach is covered in Section 4. Using different datasets from the UCR time series repository [6] the experimental evaluation is presented in Section 5. Finally, Section 6 concludes the paper.

2 Definitions and Related Work

A time series T of length m is an ordered sequence of real values $T = (x_i)_{i=1\dots m} \in \mathbb{R}^m$. In principle, any x_i may have arbitrary precision, but we assume that number are discretized to k different values (as suggested in [4]).

Minimum Description Length Principle. The description length $DL(T)$ of series T is the number of bits required to encode T. Depending on the type of encoding $DL(T)$ will vary. Assuming $k = 16$, a naive, direct calculation of $DL(T)$ requires 4 bits per value, amounting to $DL(T) = \log_2(k) \cdot m$. If we use different code lengths l_x to encode value x, we arrive at $DL(T) = \sum_{i=1}^{m} l_{x_i}$. We may obtain such a variable-length code from Huffman coding [5], which assigns shorter codes to more frequently occurring values, thereby minimising the overall code length.

Rather than a direct encoding of all values in T, a compact representation or approximation of T may help to further reduce the description length. Instead of the original time series T, a model M may be encoded using $DL(M)$ bits (depending on the type of model used). As the intention of the model is to approximate the original data, there is a loss of precision when considering model M instead of the original series T. To allow a lossless reconstruction of T, we have to encode the differences between model M and original series T, too. In total, the description of T via model M requires $DL(M)$ bits for encoding the model plus $DL(T|M)$ for a full reconstruction of T given the model:

$$DL(T, M) = DL(M) + DL(T|M)$$

If a model M captures the main characteristics of the series T well, using model M as an intermediate step may pay off in terms of the total description length, that is, we may observe $DL(T, M) < DL(T)$. Figure 1 illustrates such a situation. Encoding the series T of length 30 directly (4 bits per value) amounts to $30 \cdot 4 = 120$ bits. The Huffman code assigns codes of length 3 to the more frequent values and 4 to the less frequent values, amounting to 97 bits in total. An adaptive piecewise constant approximation (APCA) of T is shown in Figure 1 (red line). This model may be represented by a series of pairs denoting the point in time at which the segment starts and the value that holds within the segment:

$$M = ((1, 3), (17, 15), (25, 8)) \tag{1}$$

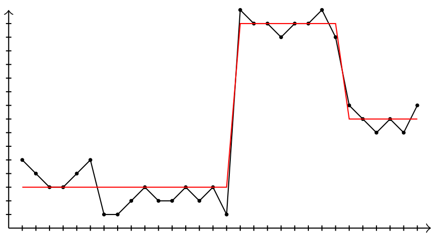

Fig. 1. An example time series.

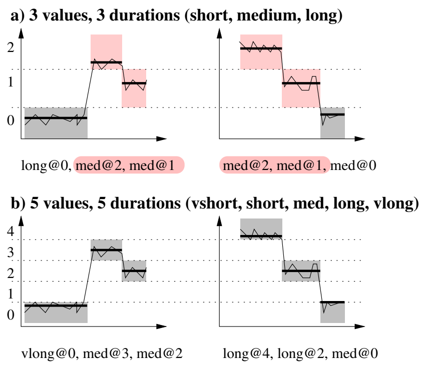

Fig. 2. Depending on the discretization, a pattern may or may not show up.

With a direct encoding (still assuming $k = 16$ different values and 24 points in time) we get (when ignoring the first time point as it is fixed to 1): $DL(M) = 3 \cdot \log_2(16) + 2 \cdot \log_2(24) \approx 22$ bits. To reconstruct the original series T, we still need the deviations from the model:

$$\Delta_M(T) = (2, 1, 0, 0, 1, 2, -2, -2, -1, 0, -1, -1, 0, -1, 0, -2, 1, 0, 0, -1, 0, 0, 1, -1, \ldots$$

In this example, the delta series consists of five values only $(-2, -1, 0, 1, 2)$ and may be (naively) encoded by 3 bits per value $(DL(T|M) = 90)$ or less when using Huffman coding $(DL(T|M) = 65)$. In either case, the total description length $DL(M)+DL(T|M)$ became smaller, because the model M captures T very well, such that the deviations can be encoded more efficiently. To save even more bits, we may choose a coarser granularity k_M for the model than the granularity k_T used for T and $\Delta(T)$.

To find the inherent structure of T, the MDL principle advocates to search for the best encoding [2]. In [4] a range of possible models is constructed[1] for a given time series T. The model with the minimal description length successfully identified the best-suited model for T.

Sequitur. To identify chunks of repeating segments we will use Sequitur [9], which is a string compression algorithm that constructs a context-free grammar from a text string and a compressed representation of the input string using non-terminal symbols of the grammar. For instance, the input sequence *aabaab* would be compressed to XX with two rules $X \to Yb$ and $Y \to aa$ (using capital letters for non-terminals). Sequitur has been used in [7] to derive patterns (grammar rules) from a symbolic approximation (SAX [8]) of time series. The greedy algorithm has several nice properties (e.g. a new non-terminal is introduced only if it can be used at least twice for the compression of the input sequence) and

[1] e.g. adaptive piecewise constant approximations (APCA), piecewise linear approximation (PLA), discrete Fourier decomposition (DFT), etc.

linear runtime complexity. As in [7], we will use Sequitur to compress segment series.

Time Series Patterns. Usually approaches to time series pattern discovery define some distance measure between subseries and apply a sliding window approach to compare subseries within the same or between different series (cf. [1,7] and ref. therein). Such approaches assume, for instance, that the position of a pattern in time is not relevant and/or that the pattern does not exceed the window length. It is also common to perform a z-score normalisation of time series, thereby losing the capability of focusing patterns on exact slopes or exact values. In this work, we want to avoid such assumptions and investigate if the MDL principle can reveal the circumstances under which patterns show up prominently.

3 Outline of the Idea

We assume a set $\mathcal{T}$ of time series is given, not necessarily all of the same length. We want to investigate, how the idea of finding a best representation of a single series by MDL from [4] can be successfully extended to the problem of finding a set of rules or patterns for a whole set of time series. While in [4] the *raw data* were the time series and the *models* were the APCA representations (amongst others), we start with APCA-transformed series, which take the role of *raw data* now. We use the term *segment series* for a given APCA model to emphasise that they become the raw data and to avoid confusion with the patterns that will serve as models hereafter.

A model is a condensed, lossy representation of the original segment series. Since we encode *all* segment series rather than just one, we hope to benefit from similar subsequences within the same and across different segment series. Once such re-occurring subsequences have been identified, we encode them as part of our model and refer to them rather than encoding them multiple times. The search for the best representation involves two aspects: (1) Similar to the different types of time series approximation (piecewise constant, piecewise linear, etc.) we can think of different segment representations that lead to different *types* of patterns (see below). (2) Secondly, for any kind of representation, we may consider two segments as being sufficiently similar (to match each other) if they become identical under some discretization.

Figure 3 shows two segment series (dashed blue line and dotted red line). We consider a number of possible representations for a given segment:

ATAV: The most direct representation of a segment series is $(t_i, x_i)_{i=1\ldots m}$ where t_i is the starting point in time of the i^{th} segment and x_i its value (ATAV: **a**bsolute **t**ime, **a**bsolute **v**alue) as used in (1). Then, two segment series share a common subsequence only if they are aligned in both dimensions simultaneously. The dotted red and dashed blue series in Fig. 3(top left) share the segment series drawn in black.

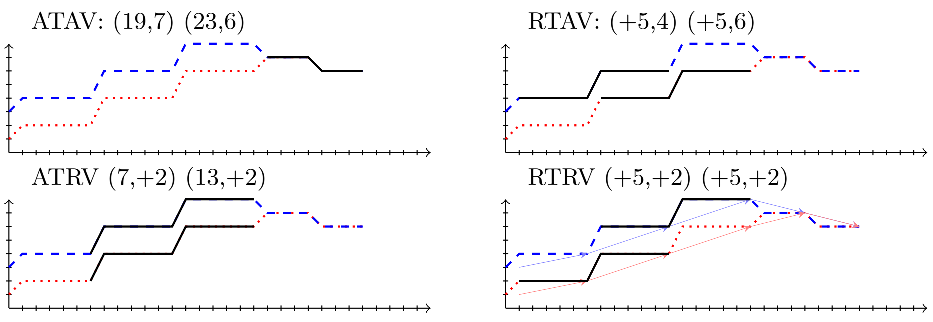

Fig. 3. Two segment series and (some) shared segments in various representations.

RTAV: Rather than encoding absolute time points, the tuples may store only relative time (duration), together with the absolute value (RTAV: relative time, absolute value). Identical subsequences may then appear at different positions in time but have identical values (top right).

ATRV: Similarly, relative values of the time series may be stored (keeping absolute time points). Identical subsequences then occur at the same point in time and change by the same amount in their value (lower left).

RTRV: Finally, both values may be encoded relatively to the previous segment. The segment itself is then interpreted as a vector (additionally drawn in the lower right image). Identical subsequences occur at different points in time and at different values, but keep the same shape.

We consider all pattern types as potentially useful. If the time series at hand are speed profiles recorded from car drivers (speed at time t, $t = 0$ at start of journey), we expect characteristic speed levels to reoccur in patterns (speed limits). To discover driving patterns, RTAV may thus be the best representation. Air pressure time series recorded at similar weather conditions (e.g. stormy) may occur at different levels of air pressure but usually exhibit similar slopes, so RTRV might be the best segment representation. When examining the effect of marketing effort on product orders over time (t=0 for start of sales promotion), we may observe characteristic lags between an increase in sales figures depending on the involved marketing channels, so ATRV may be a the appropriate representation. ATAV may be considered as the least interesting representation, as it corresponds to a 1:1 match of subseries. We do not consider it in this paper due to lack of space.

Regarding the matching of patterns, we consider two segments as *matching* if their discretized versions become identical. This is illustrated in Figure 2 for RTAV: Using a discretization into three values and three durations the sequence "*(medium length, value 2), (medium length, value 1)*" occurs in both examples (top row), but we observe no repetition at a finer granularity.

We want the MDL principle to identify the representation that characterizes a given dataset best. We will employ Sequitur to identify the patterns, so the Sequitur outcome (grammar and compressed sequence) corresponds to the model

Algorithm 1

Require: $\mathcal{T}$ set of time series, maximal temporal granularity k_T and value gr. k_V
Ensure: Best pattern representation (e.g. RTRV) and time/value discretization

```
 1: $best = \infty$
 2: $\mathcal{S} = \{S \mid T \in \mathcal{T},$ construct segment series S from an APCA representation of T$\}$
 3: for all segment representations $R$ (ATRV, RTAV, ...) do
 4:     Let $\mathcal{S}'$ be the set of segment series $\mathcal{S}$ in representation $R$
 5:     for all considered segment discretizations $(k_1, k_2) \in \mathbb{N}_{\leq k_T} \times \mathbb{N}_{\leq k_V}$ do
 6:         $\mathcal{D} = \{D \mid S \in \mathcal{S}',$ D is the $(k_1, k_2)$-discretized segment series $S\}$
 7:         merge consecutive segments in segment series of $D \in \mathcal{D}$
 8:         find Sequitur model $M$ (grammar and $c_{\text{seq}}$) from $c_{\text{raw}}$
 9:         if $DL(M) + DL(T|M) < best$ then
10:             $best = DL(M) + DL(T|M)$, store $R, k_1, k_2$
11:         end if
12:     end for
13: end for
14: return best representation (stored $R, k_1, k_2$)
```

and defines $DL(M)$. The difference between the Sequitur model and the set $\mathcal{T}$ determines $DL(\mathcal{T}|M)$.

4 Algorithmic Approach

We propose a simple approach (cf. Algorithm 1) to find the best patterns from a set of time series. A pattern is a subsequence of segments (encoded in alternative ways) at a given resolution (alternative discretizations of segments). The hypothesis is that the inherent properties of the series can be best exploited by the pattern representation that leads to a minimal description length for the whole set of series.

4.1 Preprocessing the series

As already mentioned, $\mathcal{T}$ denotes a set of time series $(x_1, \ldots, x_n)$ of varying lengths. Each time series is transformed into a piecewise constant approximation (using e.g. [4]), where the length of each segment may vary (line 2 of Algorithm 1). All further processing is done upon these APCA representations. As this approximation is performed for each time series individually, each series may come up with a different set of discrete values to approximate the original series. From the approximations, a segment series $S' = ((a_1, b_1), \ldots, (a_m, b_m))$ is constructed for each series S, where a_i denotes absolute or relative temporal information and b_i absolute or relative time series values, depending on the currently chosen segment representation (see page 4).

The next step (line 4) performs a discretization of the segments, which involves the discretization of both tuple values into k_1 and k_2 discretized values, resp., such that we deal with at most $k := k_1 \cdot k_2$ different discretized segments. For any choice of k_i we divide the range of values into k_i equally sized intervals. We refer to a discretized value of x or t by putting it into squared brackets $[x]$ or $[t]$. It may happen that two consecutive segments become similar after discretization such that they are better represented by a single segment (likely to happen if the APCA granularity is high but the current segment representation is coarse). If and how segments are merged depends on the chosen representation:

RTAV: Two consecutive segments $(\Delta t_1, v_1)$ and $(\Delta t_2, v_2)$ may refer to the same discretized value $v = [v_1] = [v_2]$ and are merged into one segment $(\Delta t_1 + \Delta t_2, v)$.

ATRV: Two consecutive segments $(t_1, \Delta v_1)$ and $(t_2, \Delta v_2)$ having the same discretized time point $t = [t_1] = [t_2]$ are merged into a segment $(t, \Delta v_1 + \Delta v_2)$.

RTRV: We interpret a segment $(\Delta t, \Delta v)$ as having a slope of $\frac{\Delta v}{\Delta t}$ for Δt time units; so we merge consecutive segments $(\Delta v_1, \Delta t_1)$ and $(\Delta v_2, \Delta t_2)$ to $(\Delta v_1 + \Delta v_2, \Delta t_1 + \Delta t_2)$ if both segments encode the same (discretized) slope.

Finally any segment series $S = ((a_1, b_1), \ldots, (a_m, b_m)) \in \mathcal{S}'$ has been transformed to a discretized series $D = (d_1, d_2, \ldots, d_{m'}) \in \mathcal{D}$ with $|\mathcal{D}| = k_1 \cdot k_2 =: k$, $m' \leq m$. For simplicity, we use numbers $1, 2, \ldots, k$ to refer to the available types of discretized segments. Two subseries with similar but different APCA representations may now, depending on the discretization parameters, appear identical after discretization and merging (cf. Figure 2).

4.2 Finding the model

Sequitur shall be used to identify re-occurring subsequences of segments. As the discretized segments are represented by numbers, patterns correspond to sequences in $\{1, \ldots, k\}$. Let us denote a *tuple concatenation operator* by $\bullet$, i.e., $(a, b, c) \bullet (d, e) = (a, b, c, d, e)$. With d_i denoting the i^{th} discretized series, the full set $\mathcal{D}$ of m series is encoded into a single sequence (over the alphabet $\{-m, \ldots, -1, 1, \ldots, k\}$):

$$c_{\text{raw}} := d_1 \bullet (-1) \bullet d_2 \bullet (-2) \bullet d_3 \bullet (-3) \cdots d_n$$

The negative numbers serve as separators between the encoded segment series. As Sequitur requires a symbol to occur at least twice before introducing a rule, these separators effectively prevent Sequitur from elaborating rules that connect symbols from the end of series d_i and the beginning of series d_{i+1}. Such rules would depend on the (arbitrary) order of series d_i and are therefore undesired.

The code c_{raw} represents the input to Sequitur. Sequitur delivers a grammar based on a set of new (non-terminal) symbols, which we encode also by numbers (starting at $k + 1$). A rule of the grammar thus reads like $A \rightarrow BC$ where A, B and C are numbers. A represents a non-terminal (thus $A > k$) and B (as well as C) may refer to a terminal symbol (discretized segment; $1 \leq B \leq k$) or another non-terminal symbol ($B > k$). Sequitur also delivers a compressed sequence c_{seq}

from which the original sequence c_{raw} can be reconstructed by replacing non-terminals with the resp. right-hand side of its rule. All separators $s < 0$ (e.g. (-1), (-2) from c_{raw} above) in c_{seq} may be removed completely or replaced by a single separator symbol (0) to preserve the separation of the series (but different symbols were only necessary to prevent Sequitur from deriving rules that link different series). Thus, if the grammar consists of r rules, the output is a sequence over the alphabet $\mathcal{A} = \{0, 1, \ldots, k + r\}$ ($\leq k$: discretized segments, $> k$: non-terminals).

For instance, we obtain $c_{\text{raw}} = (1, 3, 2, 4, -1, 1, 2, 4, 3, -2, 3, 1, 2, 4)$ from three discretized sequences $d_1 = (1, 3, 2, 4)$, $d_2 = (1, 2, 4, 3)$ and $d_3 = (3, 1, 2, 4)$. Sequitur may deliver two rules $5 \rightarrow 24$ and $6 \rightarrow 15$ (with new non-terminals 5 and 6), leading to $c_{\text{seq}} = (1, 3, 5, 0, 6, 3, 0, 3, 6)$. This procedure is carried out for all considered discretizations and all considered pattern representations in line 8 of Algorithm 1.

4.3 Calculating DL

The description length of the model consists of the Sequitur grammar and the compressed collection of sequences. We encode this as follows: (1) the number r of rules, (2) the right-hand side of all rules (no separation necessary because the right-hand side of a Sequitur rule has always 2 symbols), (3) the compressed sequence c_{seq}. We use a Huffman code to obtain minimal coding costs.

Secondly, we have to determine the difference between the original segment series $\mathcal{S}$ and the model. Let us ignore the merging step of line 7 for the moment. The Sequitur compression can be reversed to arrive at the original sequence c_{raw}, so we have to encode the differences between the discretized segments of c_{raw} and the original segments of $\mathcal{S}'$. We apply the same pointwise differencing as described for the case of time series in section 2: If the original segment series is $S = ((1, 4), (7, 2), (17, 15), (25, 8))$ and the time points and values are discretized to $\{1, 5, 10, 15, 20, 25, 30\}$ and $\{2, 6, 10, 14\}$, resp., we obtain

$$\begin{aligned} D &= ((1, 6), (5, 1), (15, 14), (25, 10)) \\ \text{and } \Delta_D(S) &= ((0, -2), (2, 1), (2, 1), (0, -2)). \end{aligned}$$

The better the discretization adopts to the segments occurring in $\mathcal{S}$, the shorter $DL(\mathcal{S}|M)$. Again, a Huffman code is used to encode $\Delta_D(S)$.

The merging of segments in line 7 is necessary to join segments that were considered different in their APCA representation but become identical under the currently applied segment discretization. Without such a merging step, we have as many original as discretized segments (1:1 relationship), but merging may reduce the number of discretized segments. Thus, from a single encoded segments in c_{raw} we may have to reconstruct multiple original segments in $\mathcal{S}'$. For example, if $S = ((1, 4), (7, 2), (15, 14), (17, 15), (25, 8))$ is discretized to $((1, 6), (5, 1), (15, 14), (15, 14), (25, 10))$ and subsequently merged to $((1, 6), (5, 1), (15, 14), (25, 10))$, we have to keep in mind which segments were merged in order to calculate $\Delta_D(S)$ correctly:

$$\begin{array}{rlllll} S = & (1,4) & (7,2) & (15,14) & (17,15) & (25,8) \\ D = & (1,6) & (5,1) & \multicolumn{2}{c}{(15,14)} & (25,10) \\ \Delta = & (0,-2) & (2,1) & (0,0) & (2,1) & (0,-2) \end{array}$$

There are multiple ways to encode how many segments of Δ belong to a single segment of D: either we include counts for the number of Δ-segments belonging to the next D-segment or we insert a special "glue" symbol (g) between Δ-segments belonging to a merged D-segment:

$$\begin{array}{rllll} \text{counts:} & 1(0,-2) & 1(2,1) & 2(0,0)(2,1) & 1(0,-2) \\ \text{glue symbol:} & (0,-2) & (2,1) & (0,0)g(2,1) & (0,-2) \end{array}$$

5 Experimental Evaluation

We examine three datasets to evaluate whether the identification of *intrinsic patterns* via MDL is viable: (1) a (one-dimensional) random walk dataset, (2) the CBF dataset consisting of short time series from three different classes (describing a cylinder, a bell and a funnel) and (3) the symbols dataset consisting of 6 different hand-drawn symbols on a touchscreen. The latter two datasets are taken from [6]. These datasets have been chosen for the first experiments because the random walk should not contain any particular patterns (by construction) while the other two datasets are known to contain patterns (cf. [6]).

By the term *configuration* we refer to both, the chosen pattern type (e.g. RTRV) and the granularity used for time and value discretization. Once Algorithm 1 has identified the best configuration, how do we know if this representation succeeded in capturing the patterns inherent in the time series collection? The CBF and symbols dataset have class labels, so we investigate if the discovered rules correlate with class labels. The algorithm is not aware of the class labels, but we expect intrinsic patterns to correspond to class-specific subseries. For rules of the grammar that apply to at least 5% of the series, we qualitatively examine their entropy. For each class, we report the rule with the lowest entropy.

Table 1 shows the results for all three datasets. The smallest total description lengths (per pattern type) are shown in column *total*. In all three datasets, the configuration with the smallest total length is of type RTRV. However, for the random walk and CBF dataset, only a few patterns (meeting the 5% usage threshold) were found for RTRV configurations. The minimal cost configuration does not seem to be a good indicator to identify the intrinsic pattern representation. We attribute this observation to the fact, that the RTRV representation benefits from the smoothness of the considered time series: consecutive segments are only a few time indices apart and (thanks to the smoothness) the deviation between consecutive values is also comparatively small. Many small differences (obtained from relative values) are encoded much more efficiently than absolute values that distribute more uniformly. This gives RTRV an advantage over RTAV and ATRV, because two values are encoded relatively rather than just one.

Table 1. Results for the three data sets.

a) random walk	best granularity		description length			reduced
	time	value	model	delta	total	size
RTAV	2	9	6191	69400	75591	40.6%
ATRV	4	2	3352	76603	79955	31.2%
RTRV	6	3	3352	69400	72752	85.4%

b) CBF	best granularity		description length			reduced	entropy of best rule		
	time	value	model	delta	total	size	cylinder	bell	funnel
RTAV	3	4	6594	40391	47525	37.5%	0.00	0.00	0.00
ATRV	3	2	3898	38063	41961	39.3%	0.49	1.53	0.00
RTRV	2	2	2554	35020	37574	91.1%	–	0.98	–

c) symbols	best granularity		description length			reduced	entropy of best rule				
	time	value	model	delta	total	size	1	2	3	4|5	6
RTAV	3	7	30k	379k	409k	23.8%	0.53	0.96	0.00	0.61	0.00
ATRV	13	2	23k	350k	374k	21.4%	0.95	2.00	0.00	0.11	0.00
RTRV	11	52	67k	259k	327k	45.9%	1.63	0.55	0.57	0.89	0.97

But this affects the encoding of the differences $\Delta_D(S)$ only (column *delta*). The description length of the (compressed) model involves only c_{raw}, which is just a sequence over an alphabet of terminal and non-terminal symbols. To evaluate how well a configuration supports the 'compressability' of c_{raw}, we report the size of the model as a fraction of the size of an uncompressed model (as if Sequitur delivered an empty grammar for c_{raw}) in column *reduced size*. A smaller percentage indicates a better compressability of the model and is thus considered to be a better indicator for the best pattern representation.

The 500 series in the random walk dataset (without class labels) consist of 250 values and start at $x_1 = 0$. Although there are no patterns imputed in the dataset there may nevertheless be incidental repetitions to be discovered by Sequitur. RTRV patterns are sequences of "increase by Δv_i within Δt_i time units". This representation is useful to approximate the up's and down's of the random walk locally, but due to the random nature of the dataset, a longer series of certain up's and down's is unlikely to repeat itself, so the representation is of limited use to identify reoccurring patterns (size remains 85.4% of uncompressed model, cf. Figure 1a). With ATRV we achieve the best model compression: The temporal granularity of 4 subdivides the time axis into 4 intervals and Δv takes only two values (increasing, decreasing). At this configuration, any random walk consists of a series of length 4 only, a particular series may be described as, e.g., 'values increase in the first and last quarter, but decrease in the second and third'. Only 2^4 different sequences exist, which is exploited by Sequitur. Thus, the best configuration takes a rather global perspective on the series, which is a reasonable result for random walk data.

The results for the CBF dataset (900 series, 3 classes) are shown in Table 1b. Only a few short rules are discovered by Sequitur for RTRV with little connection to the classes. The best model compression was achieved for the RTAV representation – and it is also the RTAV model which delivered patterns that best correspond to classes. Relative times are reasonable for CBF, because the imputed patterns are randomly displaced in time, as well as absolute values, because all CBF series jumps and linearly interpolates between two values only.

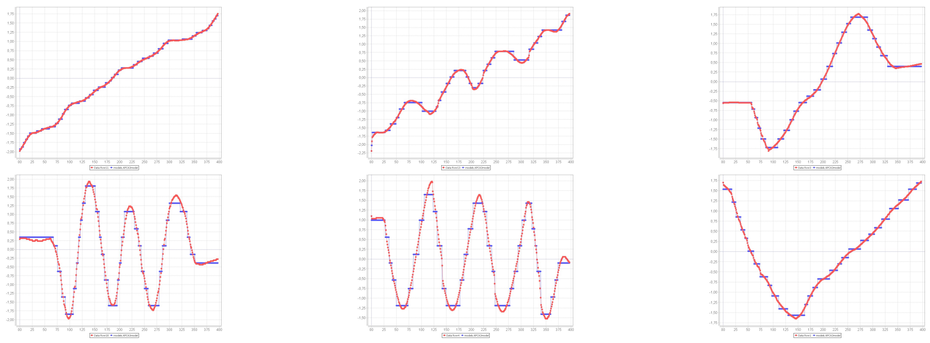

Fig. 4. Examples from the symbols dataset (note the similarity of classes 4 and 5).

However, the APCA approximation of the CBF series are quite short (only 3-5 segments remain per series), which limits the length of discoverable patterns. This is quite different for the symbols dataset (995 series, 6 classes, cf. Figure 4); results are shown in Table 1c. This time the optimal RTRV granularity is much higher and the discovered rules are much longer, non-terminals of the grammar represent sequences of up to 13 segments. The RTRV model compression is much more competitive compared to the CBF and random walk datasets, the best rule (in terms of entropy) for class #2 is of type RTRV. This is due to the fact that the time series consist of similar shapes that repeat across different classes and also within series of the same class. The highest model compression is achieved with ATRV (down to 21.4%). Series from multiple classes have long up/downward trends, which are also exploited by patterns of type RTAV and RTRV, but classes 1∪2, 3 and 6 can be easily distinguished if we know where these trends occur in time. Again, the best configuration provides those patterns that are most meaningful with respect to class labels.

6 Conclusions

Patterns may disguise themselves in time series in quite different ways. To identify similar subsequences, the shape of the subsequence may be important, the position in the time series, their absolute value, etc. No repetition will be exactly identical, but it is not a priori clear under which resolution patterns will show up. In this preliminary work we explored if the MDL principle can successfully be applied to identify the best pattern representation in terms of pattern types (absolute/relative values) and degree of similarity (discretization granularity). The preliminary results are promising: Despite the simplistic approach, the configuration that led to the highest model compression always delivered patterns that correspond best to class labels (which were unknown to the MDL approach).

References

1. T.-C. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, Feb. 2011.
2. P. D. Grunwald. *The Minimum Description Length Principle.* University Press Group Ltd, 2007.
3. K. Hill. Hate To Break It To You, But Your Car Likely Has A Black Box 'Spying' On You Already. http://onforb.es/I7BRLJ, 2012.
4. B. Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi, and E. Keogh. Discovering the Intrinsic Cardinality and Dimensionality of Time Series Using MDL. In *Proc. 11th Int. Conf. on Data Mining (ICDM)*, pages 1086–1091, 2011.
5. D. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept. 1952.
6. E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR Time Series Classification/Clustering Homepage, 2011.
7. Y. Li and J. Lin. Approximate variable-length time series motif discovery using grammar inference. *Proceedings of the Tenth International Workshop on Multimedia Data Mining - MDMKDD '10*, pages 1–9, 2010.
8. J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.
9. C. G. Nevill-Manning and I. W. Witten. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, Sept. 1997.