Knowledge-Based Systems 244 (2022) 108554

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

TA4L: Efficient temporal abstraction of multivariate time series

Natalia Mordvanyuk [a,*], Beatriz López [a], Albert Bifet [b,c]

[a] eXiT Research Group, University of Girona, Campus Montilivi, Building EPS4, 17071 Girona, Spain
[b] LTCI, Télécom Paris, IP Paris, France
[c] AI Institute, Univerity of Waikato, Hamilton, New Zealand

ARTICLE INFO

Article history:
Received 11 March 2021
Received in revised form 15 February 2022
Accepted 9 March 2022
Available online 16 March 2022

Keywords:
Multivariate time series
Temporal abstraction
Time interval sequences
Time interval related patterns

ABSTRACT

In this work, we introduce TA4L, a new efficient algorithm to transform multivariate time series into Lexicographical Symbolic Time Interval Sequences (LSTISs), that is, sequences ready to feed time-interval related pattern (TIRP) mining algorithms. The ultimate goal is to make explicit the embedded, ad-hoc pre-processes related to TIRP mining algorithms while offering an efficient solution for the required pre-processing. On the one hand, TA4L divides the signals into segments based on time duration (instead of the often-used practice based on the number of samples), which allows the construction of consistent time intervals. Concatenation of intervals is controlled by a maximum time gap constraint that reinforces the generated time intervals' consistency. Moreover, different ways to parallelise the algorithm are explored that are accompanied by efficient data structures to speed up the pre-processing cost. TA4L has been experimentally evaluated with synthetic and real datasets, and the results show that TA4L requires significantly less computation time than other state-of-the-art approaches, revealing that it is an effective algorithm.

1. Introduction

Around the 2000th year, while the research in sequential pattern mining (SPM) [1] was in full swing, [2] proposed a new SPM approach known as time-interval related patterns (TIRP's) mining or temporal arrangement's mining. TIRPs are richer in knowledge than patterns obtained with SPM. They take into account not only the events and their ordering (e.g. $\langle A, B, C \rangle$), but also the temporal relations between the events (i.e. $\langle A$ *occurs before* $B\rangle$, $\langle A$ *meets* $B\rangle$, $\langle A$ *overlaps* $B\rangle$, and so on) that discover from time-series data.

One of the major drawbacks of the TIRP mining algorithms is that they cannot process multivariate time series (neither time series) directly. They require a pre-processing phase to obtain sorted sequences of time-interval data. The pre-processing phase consists of several steps, usually carried out sequentially: 1-time series segmentation into intervals according to a given size, 2-discretisation of numeric values, 3-sequence construction and 4-sequence sorting (see Fig. 1). The first two steps comprise the Temporal Abstraction (TA) task, which involves the transformation of series of values indexed by time (i.e. ⟨0.2, 8:00⟩, ⟨0.2, 12:00⟩, ⟨0.1, 13:00⟩, ⟨0.2, 14:00⟩, ⟨0.8, 15:00⟩, ⟨0.7, 16:00⟩ into time intervals and the corresponding symbols that abstract the variable values along them (i.e. ⟨A, [8:00, 13:00]⟩, ⟨A, [13:00, 15:00]⟩, ⟨B, [15:00, 16:00]⟩). On the other hand, the third step concerns the concatenation of intervals with the same symbol (i.e. ⟨A, [8:00, 15:00]⟩ when ⟨A, [13:00, 15:00]⟩ follows ⟨A, [8:00, 13:00]⟩). In this example, all values below 0.5 are labelled with A and above 0.5 with B. Regarding the sequence sorting step, the sorting criteria [3] proved to be necessary to achieve a robust and simple representation of TIRPs. That means that sequences are sorted according to the starting time of intervals, ending time, and the discretised values called Lexicographical Symbolic Time Interval Sequences (LSTISs). Nevertheless, this pre-processing is usually overlooked in the literature, which focuses on providing details about the pattern-finding algorithms.

Considering the computational complexity involved in the whole pre-processing step of the TIRP mining algorithms, some authors started to investigate the pre-processing as a separate problem and develop a specific algorithm to generate LSTIS. In particular, [4] propose a method based on labelled data. The method presented in this work is unsupervised, as TIRP mining original goal was to find patterns from unlabelled data. The TA4L algorithm aims to accelerate the pre-processing time, merging all the pre-processing tasks to be executed together in a single, special purpose algorithm. As our algorithm performs a temporal abstraction and returns LSTISs, we decided to call it the name resulting from the union of the initials of the two concepts "TA4L".

Our method is partially inspired by SAX [5]. One of the main differences with SAX is that every interval is composed of a fixed

* Corresponding author.
E-mail addresses: natalia.mordvanyuk@udg.edu (N. Mordvanyuk), beatriz.lopez@udg.edu (B. López), abifet@waikato.ac.nz (A. Bifet).

https://doi.org/10.1016/j.knosys.2022.108554
0950-7051/

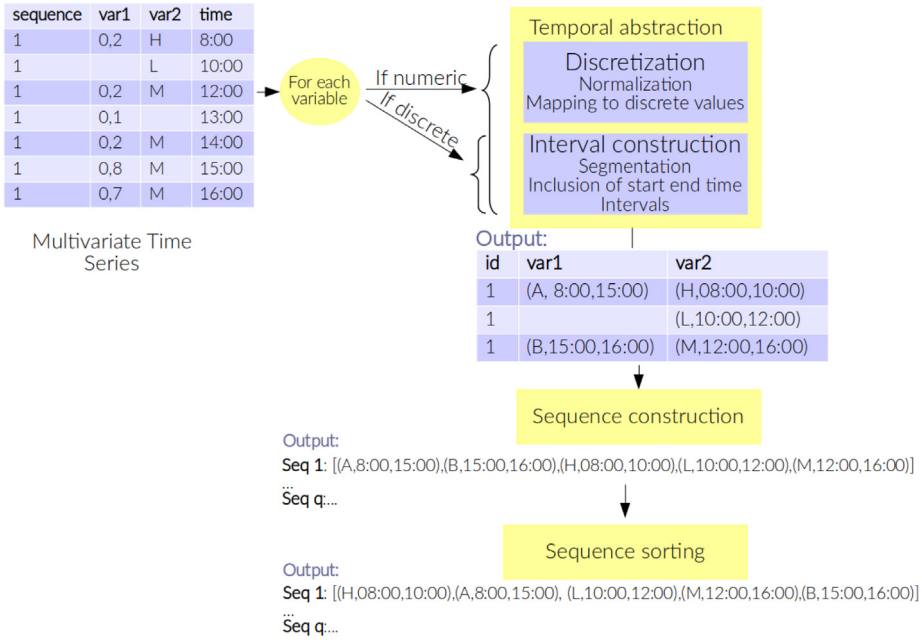


Fig. 1. Pre-processing step in the literature.

number of samples in SAX, while in TA4L, intervals are constructed based on a certain duration. This fact plays an important role when the dataset has missing values and could eventually lead to losing some TIRPs. Conversely, TA4L split input data into segments of equal time duration that later are concatenated. Such pre-processing achieves a more reliable time representation of the variables inside intervals and provides a solid basis for forthcoming TIRP mining.

The contributions of the present work are:

- The pre-processing phase for TIRP mining is explicitly described and formulated for unsupervised data.
- A new method called TA4L is provided, which generate sequences of symbolic time intervals sorted in lexicographical order. This involves a temporal abstraction approach that combines segmentation and discretisation of temporal data.
- The TA4L algorithm constructs intervals based on the time duration (instead of a fixed number of samples). In so doing, the limits of the time intervals are defined over the real values in the interval (not over the missing values resulting from the fixed-length).
- A maximum gap constraint is used to decide whether two consecutive points could be considered to be part of the same event (i.e. belonging to the same time-interval) or do not.
- Use of special data structure that allows performing intervals insertions into LSTISs efficiently.
- Different approaches to apply parallelism to the process.

To sum up, TA4L converts multivariate time series into LSTISs, and it is intended to be used as a preprocessing algorithm in the TIRPs mining field. The code of the TA4L algorithm will be publicly available from the Bitbucket repository (see Appendix A).

This paper is organised as follows. In Section 2, we provide some background and review related work on pre-processing methods for TIRP mining algorithms. In Section 3, we formulate the pre-processing problem. In Section 4, the novel TA4L algorithm is introduced. In Section 5, we give a detailed description of the datasets and the setup used in our experiments, and in Section 6, the results and a discussion about them are presented. Finally, Section 7 draws some conclusions from this work.

2. Related work

In this section, we first review how the preprocessing is performed in the TIRP mining algorithms; secondly, we go deepening on the discretisation methods employed in the TIRP mining field; finally, we motivate how the present work fits within the context of the existing literature.

2.1. The pre-processing in the TIRPs mining field

If we analyse the literature on TIRP mining, we will find that pre-processing is tightly related to the kind of data used to test the algorithms. When using synthetic datasets (as in [2,6,7]), data are usually generated with a synthetic data generator like [8], which have been modified to generate sorted symbolic time-intervals, and the pre-processing, in this case, is nonexistent.

On the other hand, when using real datasets as in [3,4,9,10], or on both synthetic and real datasets (as in the present work) [11–13], we can find different approaches to pre-processing to convert multivariate time series into LSTISs: internal [14–16] or external [11–13] to the TIRP mining algorithm. In general, the different steps followed in the literature are summarised in Fig. 1.

First, for each variable, a temporal abstraction is applied. Temporal abstraction is a conversion of a signal (e.g. blood pressure values) to an abstracted comprehensive to a human representation (e.g. "2 h of high blood pressure") [17]. When the variable is numeric, temporal abstraction involves both interval construction and discretisation steps. When the variable is already discrete, time intervals are constructed.

Regarding the interval construction step, data is segmented into time intervals, where each interval should include its corresponding start and end times. Intervals are constructed usually performing bottom-up, top-down or sliding-window approaches [18]. In the sliding window approach [19], a segment is grown until a specified error threshold is reached, where, in each window, a linear approximation is performed. In the top-down approach, time series are repeatedly split according to the best splitting point from all considered points until the desired number of intervals is obtained [20]. The bottom-up approach [10]

starts by segmenting the series with small segments and then iteratively concatenating adjacent segments. In the present work, a bottom-up approach is used.

During the discretisation step, first, the variable values are normalised. Afterwards, numeric values or the first derivative of values [4,21] are converted into a discrete representation.

TA steps (interval construction and discretisation) can be executed sequentially or at the same time with a mapping function [11,12]. The output of the TA task is a set of symbolic time intervals. Next, during the sequence construction step, all records belonging to the same sequence are grouped together. And, finally, during the sequence sorting step, sequences are ordered according to the TIRP's mining algorithm criteria: some TIRP mining algorithms require time-intervals to be sorted by their end time [6]. Others, by their start and end times [12]. Others sort them by start and end times and lexicographically [9] (as we do in the present work).

This is the process that, in general, is followed in the literature. But not always the starting point of the algorithm is the same. Examples of the full process, from multivariate time series to LSTIS, are [9–12,15]. Sometimes the starting point is interval-based data [13], and in other cases, the starting point is already ordered sequences of time-intervals [2,6].

2.2. Temporal abstraction in the TIRPs mining field

Knowledge-Based Temporal Abstraction (KBTA) [17] method is the most widely used discretisation method in the literature [9,10,12,13]. KBTA belongs to the group of supervised discretisation methods, where data is discretised following the knowledge of an expert in the corresponding field. This fact makes the output of these methods significant that lead to the successful data interpretation (e.g. [3,4,11]). The problem is when the expert knowledge is lacking, or when the discretisation is performed not for the interpretation of the time series, but rather for the performance of other tasks, possibly less intuitive for human experts, such as classification, clustering, and prediction. For example, these two works [3,4] are focused on this problem. Temporal discretisation for Classification (TD4C) [4] is another supervised discretisation method geared towards the enhancement of classification accuracy, which determines the cutoffs that will best discriminate among classes through the distribution of their states.

Alternative, non-supervised methods could be used, such as Equal Width discretisation (EWD) [22] or Symbolic Aggregate ApproXimation (SAX) [5]. EWD involves sorting the observed values of a continuous feature and dividing the range of observed values for the variable into k equally sized bins, where k is a parameter supplied by the user. SAX divides the original time-series into equally sized frames, computes the mean for each frame, and assigns a symbol to each calculated mean, based on the statistical table (of a normal continuous random variable) and on the size of the vocabulary. The size of the vocabulary and the number of frames are the parameters supplied by the user.

In [3] the KBTA, Equal Width discretisation (EWD), and Symbolic Aggregate ApproXimation (SAX) methods have been compared, and it was found that discretisation using SAX led to better accuracy on classification than using the EWD. While KBTA cut-off definitions, when available, were superior to both in terms of accuracy. TD4C was also compared to the EWD, KBTA, and SAX methods in [4]. In general, some configurations of the TD4C discretisation method and the KBTA method outperformed the other methods, but SAX was always very close to the TD4C methods and sometimes even outperformed some configurations of the TD4C.

2.3. How the present work fits within the context of existing literature?

Except for TD4C [4], in the literature on TIRP mining, the pre-processing from multivariate time series to LSTISs is usually mentioned in a brief paragraph of the experimental setup or algorithm section of the article. Unlike TD4C [4], TA4L is an unsupervised abstraction method that is transparent to discretisation, which means that the function to convert a numeric value to a discrete one is a parameter of the algorithm.

TA4L is provided with a duration constraint, designed to do not miss any TIRP without neglecting efficiency. In the TA4L, we use the maximum gap constraint to decide whether two consecutive points or intervals belong to the same interval or not. The maximum gap constraint is not a novelty. It has been widely used in sequential pattern mining [23,24] and also in TIRP mining approaches [6,25]. However, this is the first time it has been adapted in an algorithm to create LSTISs.

3. Problem statement

Given multivariate time series with missing values, where samples gather information about n variables in q situations or sequences, a minimum duration of an interval δ (e.g. in seconds), an alphabet Σ, and a maximum gap constraint *max_gap*, we are required to provide LSTISs, one per sequence $LSTIS_1, \ldots, LSTIS_q$, where variable values have been abstracted to symbols in the alphabet, annotated with time intervals bounded in $[s, e]$ (with $\delta \leq e - s \leq max\_gap$), and sorted according to time-interval boundaries and alphabet symbols.

3.1. Input parameters

Each sequence $sid \in [1, q]$ of the multivariate time series, has a total of n_{sid} samples. Each sample $Y(sid, tid)$ comprises the values of the n variables for a given sequence sid and time tid, that is, $Y(sid, tid) = \langle y_1(sid, tid), \ldots, y_n(sid, tid)\rangle$, where $y_i(sid, tid)$ is the value of the i variable of the sample. Moreover, for the purposes of this work, we denote $y_v(sid)$ to all of the values corresponding to variable v in a given sequence sid (i.e. signal).

The minimum duration of an interval can be expressed in any time scale (e.g. seconds, hours, days). The *max_gap* constraint should be consistent with the *max_gap* scale.

The alphabet Σ by default is defined in $[A, Z]$, but any other alternative or vocabulary is also possible. An important issue regarding the complexity of the problem to be tackled is its size $|\Sigma|$.

3.2. Output: LSTISs

Several notations related to LSTIS should be considered first.

Definition 3.1 (*Symbolic Time-interval*)**.** A symbolic time-interval I is a triple $I = \langle s, e, v.sym\rangle$, composed of a start time (s), an end time (e), and a symbol $v.sym$ where v is a variable name and $sym \in \Sigma$.

Time scales for the start and end times are application-dependent, as stated above. The same alphabet is used for all of the variables. Therefore we use the prefix v ($v.sym_j$) to different symbols corresponding to the different variables.

Eventually, different alphabets Σ_v could also be used, and thus $\Sigma = \bigcup_v \Sigma_v$).

In the definitions below, I.s, I.e, and I.sym are used when the start time, end time and symbol of an interval are referred to.

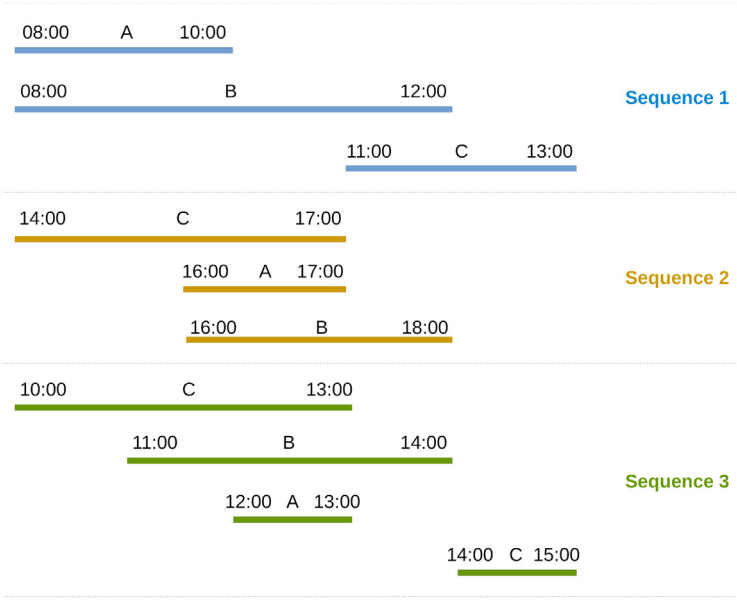


Fig. 2. Graphical representation of LSTIS.

Definition 3.2. A **symbolic time-interval sequence**, $IS = \langle I^1, I^2, \ldots, I^k \rangle$ represents a sequence of symbolic time-intervals I^i.

Given that there are n variables and n_{sid} samples per variable, a multivariate time series can be represented by a IS of length $k = (n * n_{sid})$.

To sort time-intervals, usually, relational operators are used. In the present work, rather than comparing two time intervals with an exact operator, we look for an approximate (epsilon) approach (following [3]). The epsilon approach allows for a certain variability in the temporary boundaries of events caused by noisy data [16]. The epsilon approach is implemented by means of the $=^\epsilon$ and $<^\epsilon$ operators.

Definition 3.3 (*Quasi-equal "$=^\epsilon$"*)**.** Two time-points t^i and t^j are quasi-equal, $t^i =^\epsilon t^j$, if $|t^i - t^j| \leq \epsilon$.

Definition 3.4 (*Precedes "$<^\epsilon$"*)**.** Time-point t^i precedes time-point t^j, $t^i <^\epsilon t^j$, if $t^j - t^i > \epsilon$.

Definition 3.5 (*Follows "$>^\epsilon$"*)**.** Time-point t^i follows time-point t^j, $t^i >^\epsilon t^j$, if $t^i - t^j > \epsilon$.

The relational operators $=^\epsilon$ and $<^\epsilon$ are used to sort time-interval events into a sequence.

Definition 3.6. A **lexicographical symbolic time-interval sequence** *LSTIS* is a symbolic time-interval sequence, IS $= \langle I^1, I^2, \ldots, I^k \rangle$, in which the elements are sorted in order of the start and end times using the relations $<^\epsilon$, $=^\epsilon$ and the symbols with a lexicographical order, that is: $\forall I^i, I^j \in IS (i < j)$:

$$((I^i_{.s} <^\epsilon I^j_{.s}) \vee$$
$$(I^i_{.s} =^\epsilon I^j_{.s} \wedge I^i_{.e} <^\epsilon I^j_{.e}) \vee$$
$$(I^i_{.s} =^\epsilon I^j_{.s} \wedge I^i_{.e} =^\epsilon I^j_{.e} \wedge I^i_{.sym} < I^j_{.sym}) \vee$$
$$(I^i_{.s} =^\epsilon I^j_{.s} \wedge I^i_{.e} =^\epsilon I^j_{.e} \wedge I^i_{.sym} = I^j_{.sym} \wedge |I^i_{.e} - I^i_{.s}| = max\_gap).$$

For example, ⟨⟨<8:00, 10:00, var1.A>, <8:00, 12:00, var1.B>, <11:00, 13:00, C>⟩, ⟨<14:00, 17:00, var1.C>, <16:00, 17:00, var1.A>, <16:00. 18:00, var1.B>⟩, ⟨<10:00, 13:00, var1.C>, <11:00, 14:00, var1.B>, <12:00, 13:00, var1.A>, <14;00, 15:00, var1.C>⟩⟩ is a LSTIS that contains three sequences, all of them from the same variable *var1*. The first sequence is composed of three symbolic time-intervals, ⟨<8:00, 10:00, var1.A>, <8:00, 12:00, var1.B>, <11:00, 13:00, var1.C>⟩. Fig. 2 shows a graphical representation of this example.

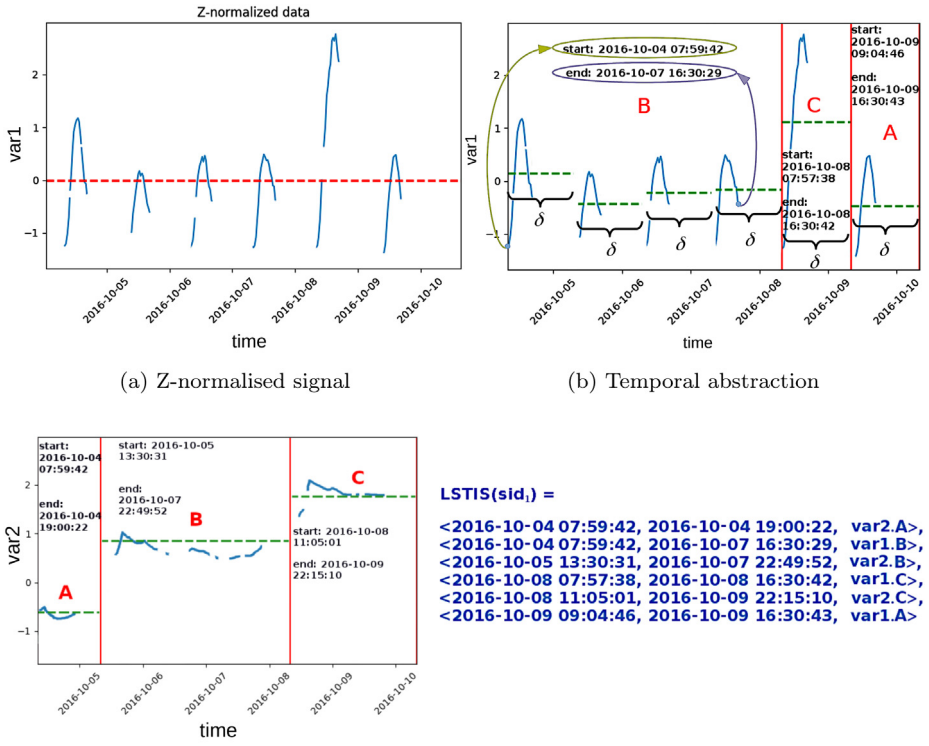
The requirement to have the LSTISs ordered implies a consumption in the algorithm. In this sense, other schemes using compositional tools (e.g. codawork, compositional distances) can be studied in the future, seeking a more compact representation of the LSTISs. However, this would impact the operation of the algorithms to be used next (downstream).

4. The TA4L algorithm

Algorithm 1: TA4L

input: MTS: multivariate time series
max_gap: the maximum gap constraint
dur: duration of the interval
output: LSTISs: a list of LSTISs for each sequence

LSTISs = {}
for *each* $MTS_{seq} \in MTS$ **do**
  $LSTIS_{seq}$ = {}
  for *each* $var \in MTS_{seq}$ **do**
    /* Temporal abstraction */
    if $MTS_{seq}(var)$ *is numeric* **then**
      $Z_{seq}(var)$ = Z-norm($MTS_{seq}(var)$)
      /* Framing, discretisation, Concatenation, and Sequence Generation (FDCS) */
      $LSTIS_{seq}$ = FDCS($Z_{seq}(var)$,max_gap,dur,$LSTIS_{seq}$)
    else
      /* Segmentation, Concatenation and Sequence Generation (SCS) */
      $LSTIS_{seq}$ = SCS($MTS_{seq}(var)$,max_gap,$LSTIS_{seq}$)
  $LSTIS_{seq}$ = sort($LSTIS_{seq}$) ; // if not using a sorted insertion
  LSTISs = append($LSTIS_{seq}$,*LSTISs*)



(a) Z-normalised signal (b) Temporal abstraction

(c) The resulting LSTISs

Fig. 3. TA4L over multivariate time series of two variables. The algorithm parameters used in this example have been: $\Sigma = 3$, $\delta = 24$ h, and discretisation = SAX.

The inputs of the TA4L algorithm include multivariate time series *MTS*, the maximum gap *max_gap* allowed between two consecutive time intervals, and the minimum duration of time intervals *dur*. The algorithm returns a list of *LSTISs*, where each position will be an LSTIS corresponding to a certain sequence.

The first step of the algorithm is to initialise the list of LSTISs. Then for each sequence, the algorithm constructs an $LSTIS_{seq}$ and adds it to LSTISs. If the variable is numeric first, a Z-normalisation of this variable is performed. Then each frame is discretised and concatenated with the next frame, based on the maximum gap and the symbol. If the variable is discrete, segmentation and concatenation are performed, also depending on the maximum gap and the symbol. And finally, in the case we decide not to use a sorted insertion to $LSTIS_{seq}$, the $LSTIS_{seq}$ must be sorted. The FDCS and SCS functions are detailed in Appendix B. The fundamentals of the proposal are technically in-deep introduced in the remaining of this section.

An example of TA4L is summarised in Fig. 3 for two variables, *var1* and *var2*. This example is used along the section to illustrate the details of the algorithm. Note that the first four bell curves of var1 in Fig. 3, from 2016-10-04 to 2016-10-07, are B, while the last bell curve at 2016-10-09 is A. This is because the breakpoints are determined by looking them up in a statistical table (look Section 4.2. Framing), and according to the statistical table, for vocabulary 3, the breakpoints are −0.4307273 and 0.4307273. This means that all values below −0.4307273 are labelled with A, all values between −0.4307273 and 0.4307273 are labelled with B, and all values above 0.4307273 are labelled with C. The means of the first four bells are 0.1562,−0.4162,−0.2381 and −0.1577, all of them between −0.4307273 and 0.4307273; therefore, all of them are labelled with B. The mean of the last bell is −0.4675, which is below −0.4307273; therefore, it is labelled with A.

4.1. Normalisation

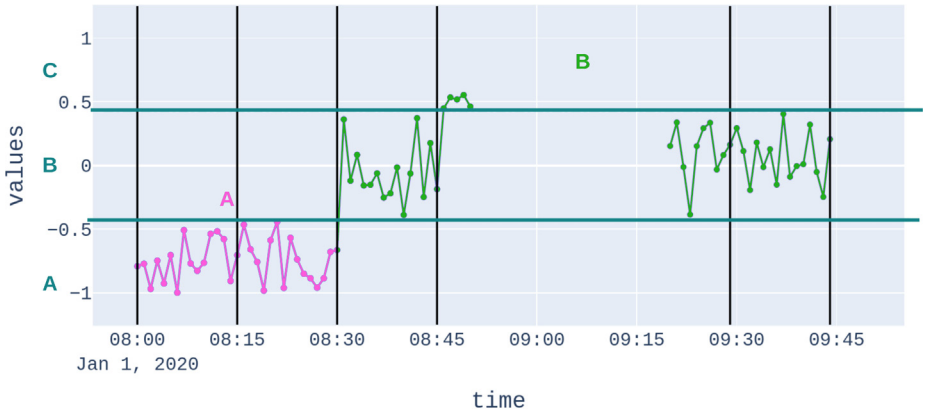
Given a sequence *sid* and the values of a variable v, $y_v(sid) = \langle y_v(sid, 1), \ldots, y_v(sid, n_{sid})\rangle$, the normalisation step applies the normalisation method using the Z-method, obtaining $y_v^Z(sid) = \langle y_v^Z(sid, 1), \ldots, y_v^Z(sid, n_{sid})\rangle$. Fig. 3(a) shows the normalisation of the variable *var1*. In the figure, it is possible to observe that there is several missing information for the variable.

4.2. Framing

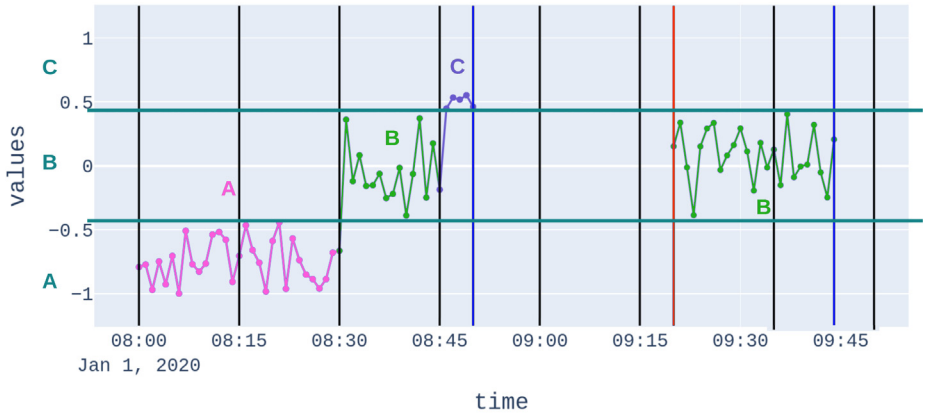
Instead of reducing the dimensionality of the input data based on the number of samples was is done in SAX, TA4L reduces it based on a duration δ, with time interval boundaries defined in known data (i.e. the start and endpoints of time intervals are on existing observations).

Given a normalised variable v in the scope of a sequence *sid*, $y_v^Z(sid) = \langle y_v^Z(sid, 1), \ldots, y_v^Z(sid, n_{sid})\rangle$, data is divided into time intervals of duration δ in the framing step, where the start time *I.s* and end time *I.e* is marked by the first and last point known inside the δ frame. For example, in Fig. 3(b), the start time of the real first time-interval is 2016-10-04 07:59:42, and the real end time is 2016-10-04 19:00:22 (not the 2016-10-05 07:59:42 marked by δ). Values from 2016-10-04 19:00:22 to 2016-10-05 07:59:41 are missing. The output of the fragmentation stage is $y_v^f(sid) = \langle y_v^f(sid, 1), \ldots, y_v^f(sid, m_{sid})\rangle$, where $y_v^f(sid, j) = \langle s_j, e_j, mean_j\rangle$, $|s_j - e_j| \leq \delta$, and $s_j < s_{j+1} \forall j$. Observe that $n_{sid} - m_{sid}$ is the data reduction achieved in this stage. Note that m_{sid} indicates the number of intervals after framing, while the 'f' is to indicate the result of framing. For example, y_v^f is the y_v after applying framing.

The fact that TA4L defines the time interval boundaries with known values greatly impacts discovering TIRP patterns in the future and presents some advantages from previous approaches as SAX [5]. For example, Fig. 4 illustrates the difference between



(a) SAX



(b) TA4L

Fig. 4. SAX and TA4L over a signal. The algorithm parameters used in this example have been: $\Sigma = 3$, $\delta = 15$ min.

applying SAX and TA4L to a signal. Note that the boundaries marked in black in Fig. 4(b), last δ (15 min). However, in the case of the presence of missing values, the final time of the interval is not the one marked by the delta (black line), but by the real final time (blue line), and in an analogous way, it is done with the start time (red line). In addition, the next limit (9:35) is calculated from the new real start time (9:20). SAX splits the original signal into five segments with the same amount of samples; its fourth segment contains unknown values, representing a 45′ time interval. In contrast to SAX, TA4L will characterise the fourth time interval with a 5′ duration. Consequently, a TIRP mining algorithm could eventually find the pattern "A before B before C" thanks to TA4L, but that will not be the case when using SAX.

4.3. Discretisation

Discretisation is the process of replacing the $mean_j$ value of the time intervals with $sym_j \in \Sigma$. To that end, TA4L uses by default the SAX approach [5], but other methods are also available, such as EWD [22] and KBTA [17]. Therefore, the output of the fragmentation stage is $y_v^d(sid) = \langle y_v^d(sid, 1), \ldots, y_v^d(sid, m_{sid})\rangle$, where $y_v^d(sid, j) = \langle s_j, e_j, v.sym_j\rangle$, $|s_j - e_j| \leq \delta$, $s_j < s_{j+1} \forall j$, and $sym_j \in \Sigma$.

If discretisation is performed following the SAX approach, it must first determine the *breakpoints*. Breakpoints are a sorted list of numbers $B = \beta_1, ..\beta_{\Sigma-1}$ such that the area under a *N(0,1)* Gaussian curve from β_i to $\beta_{i+1} = \frac{1}{|\Sigma|}$ (β_0 and β_Σ are defined as $-\infty$ and $+\infty$, respectively). These breakpoints are determined by looking them up in a statistical table computed from the data. The statistical table is used here to produce symbols with equiprobability. Secondly, the mapping from $mean_i$ to α_j, the j_{th} element of the alphabet (i.e., $\alpha_1 = A$ and $\alpha_2 = B$), is obtained as follows: $sym_i = \alpha_j$, iff $\beta_{j-1} \leq mean_i < \beta_j$. That means that all means that are below the smallest breakpoint are mapped to the symbol "A", all means greater than or equal to the smallest breakpoint and less than the second smallest breakpoint are mapped to the symbol "B" and so on.

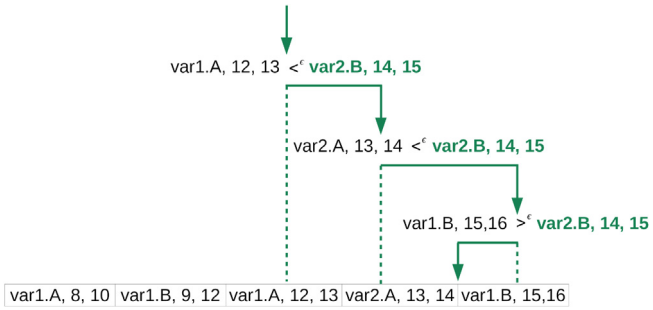
If the EWD discretisation approach is used, the range of variables is divided into $|\Sigma|$ breakpoints of equal size. Finally, when the KBTA approach is used, the breakpoints are acquired from a domain expert.

4.4. Segmentation

If the type of a v variable is discrete, temporal abstraction consists of a segmentation process (see Algorithm 1): symbolic time intervals are generated with the start time of the interval corresponding to the time a discrete value (symbol) appears, and the end time when there is a symbol change, or when there is a maximum distance with the start time of δ unit times. The output of the segmentation stage is equivalent to the discretisation step of the numerical variables, $y_v^d(sid) = \langle y_v^d(sid, 1), \ldots, y_v^d(sid, m_{sid})\rangle$, where $y_v^d(sid, j) = \langle s_j, e_j, v.sym_j\rangle$, $|s_j - e_j| \leq \delta$, $s_j < s_{j+1} \forall j$, and $sym_j \in \Sigma$. Although, in this case, the alphabet Σ is induced from the data, we keep the Σ notation for the sake of simplicity.

4.5. Sequence generation

Sequence generation is based on a data structure named *sorted list* with two main components: an LSTIS and the pointer to the



(a) Insert with an original binary search adapted to LSTISs

(b) Insert with an improved binary search adapted to LSTIs

Fig. 5. The difference between the original binary insert and an adapted binary insert that remembers the last inserted position applied to LSTISs.

last inserted element into the list. The pointer is initialised to 1 when the first interval of each variable is inserted. The remainder intervals are inserted following the binary search method [26] adapted for LSTISs. Moreover, at the moment of the time-interval insertion into the LSTIS, a concatenation process is applied. The purpose of merging the time series of the different variables of sequences is to obtain LSTISs for each sequence, which is the data format needed to perform TIRP mining.

4.5.1. Binary search for LSTISs

Binary search [26] works on sorted arrays, and it is theoretically the optimal and the fastest in practice search algorithm, which order is $O(log_2 N)$. The algorithm compares an element in the middle of the array with the value to be inserted. If the value to be inserted matches the element, its position in the array is returned; if it is less than the element, the search continues in the lower half of the array; if it is greater than the element, the search continues in the upper half of the array. By doing this, each iteration of the binary search narrows the search interval by half of the search interval of its previous iteration.

In the present work, an adapted version of the binary search was employed to search the position into LSTISs to perform the insertion of time intervals. The difference is that (i) it is applied to an LSTIS; and (ii) it takes advantage of the fact that data arrive in sorted order by time, and (iii) it uses the $=^{\epsilon}$ and $<^{\epsilon}$ relational operators.

First, a binary search is applied to each LSTIS instead of to an array of integers. Secondly, each variable time-interval is processed according to time. Therefore, the insertion takes advantage of that to start the search from the last inserted position, as shown in Fig. 5(b), instead of from position 0 as shown in Fig. 5(a). Finally, time intervals are compared with $>^{\epsilon}$ and $<^{\epsilon}$ epsilon-relational operators instead of $>$ and $<$ relational operators.

4.5.2. Concatenation: Intervals of maximum duration

If there are two or more consecutive intervals with the same symbol, these intervals are concatenated into a single interval, where the start time is the time corresponding to the start time of the first interval and the end time is the time corresponding to the end time of the last interval. The advantage of that is not only storing the memory but also providing a short, reliable and meaningful summary of information to the final user. For example, in the *var1* of Fig. 3(b), the first four intervals correspond to symbol B. Therefore, all of them are combined in a new time interval where the start time is the start time of the first time interval (2016-10-04 07:59:42), and the end time is the end time of the last time-interval (2016-10-07 16:30:29).

On the other hand, concatenation is controlled by the parameter *max_duration*, which aims to deal with particular situations of application domains. For example, for the following up of a person who suffers from diabetes, data analysed to evaluate her state should be within one day ($max\_gap = 24h$). Therefore, two consecutive time-intervals of the same sequence and symbol sorted by start time $I_i(sid) = \langle s_i, e_i, v.sym \rangle$ and $I_j(sid) = \langle s_j, e_j, v.sym \rangle$, are transformed into a single interval $I_k(sid) = \langle s_i, max(e_i, e_j), sym \rangle$, iff $(I_j.e - I_i.s) < max\_gap$ and $s_i \leq s_j$.

This has some advantages when discovering TIRPs. For example, in Fig. 6(b), we can see that from 8.50 to 9:20, there is no data. If $max\_gap = 30'$, both would not be concatenated into a single interval, and as a consequence, a TIRP mining algorithm could eventually find the pattern "a before b before c before c" thanks to the maximum gap constraint in the TA4L (see Fig. 6(b)), but that will not be the case when using SAX (see Fig. 6(a)).

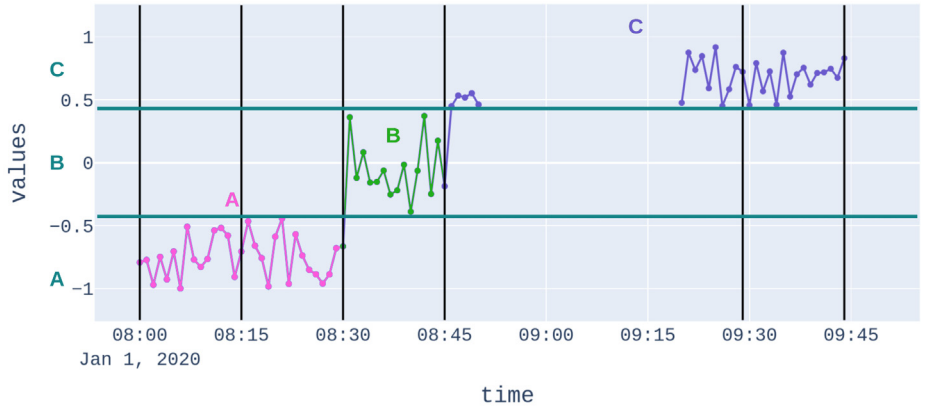
4.6. Data structures and parallel strategies

The basic procedure of TA4L presented in Fig. 3 considers the sorted list data structure where time intervals are inserted in a sorted manner with an adapted version of binary search as soon as they are obtained. In this context, sorted manner refers to finding the appropriate time interval in the LSTIS after which the input time interval is to be inserted (see Section 4.5.1). After such insertion, the existing LSTIS remain ordered following Definition 3.6. With this data structure, parallelism can be applied to sequences, i.e. the procedure of transforming time series to LSTISs is executed as an independent asynchronous thread for each sequence of the TA4L algorithm, as shown in the first image of Fig. 7. Concretely, from line 4 to line 11 of Algorithm 1 (that is, all the procedure of transforming time series of a sequence to LSTIS) is encapsulated within a function, where the input to the function is the time series of a sequence, and the output is the LSTIS the sequence. Then, the python multiprocessing Pool has been used for parallel execution of that function across multiple sequences, distributing the input data across processes (data parallelism). In this sense, everything that goes from line 4 to line 11 (e.g. framing, discretisation, binary search, etc.) need not be adapted to parallelisation because there is no communication between the processes, and its results are independent (i.e. LSTIS of sequence 1 is independent of sequence 2).

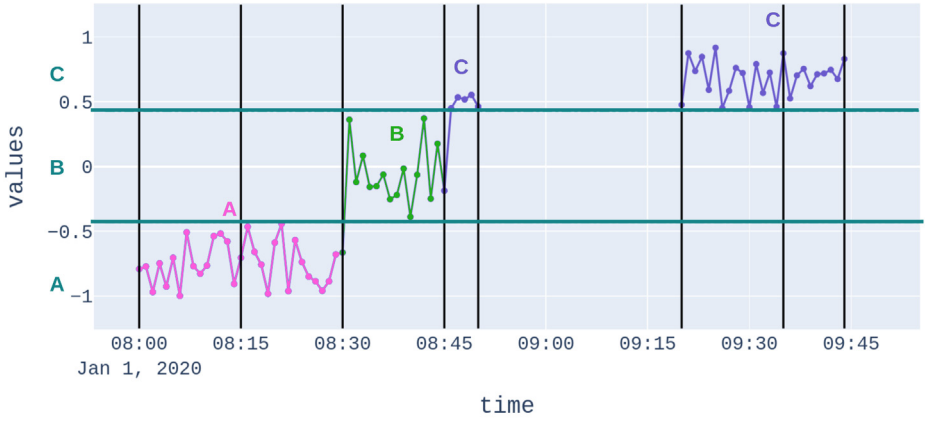
On the other hand, an alternative simple list data structure could be considered, and applying a sorting procedure in the end, opening the opportunity to other parallel strategies: parallelism can be applied apart from sequences (see the second image of Fig. 7) to the variables (see the third image of Fig. 7), i.e. the procedure of transforming a signal into LSTIS is executed as an independent asynchronous thread for each variable in a sequence. Table 1 summarises all the parallelism strategies. In the case of parallelism applied to the variables, from line 6 to line 10 of Algorithm 1 (that is all the procedure of transforming time series of a variable to LSTIS) is encapsulated within a function. The rest is analogous to the parallelisation by sequence.

5. Experimental setup

TA4L was implemented in Python 3.7.6. The different implementations of the TA4L shown in Table 1 have been implemented and labelled accordingly. Moreover, the state-of-the-art approach (Fig. 1) has been implemented according to the following configuration:



(a) SAX



(b) TA4L

Fig. 6. Concatenation example over a signal. The algorithm parameters used in this example have been: $\Sigma = 3$, $\delta = 15$ min, and $max\_gap = 30'$.

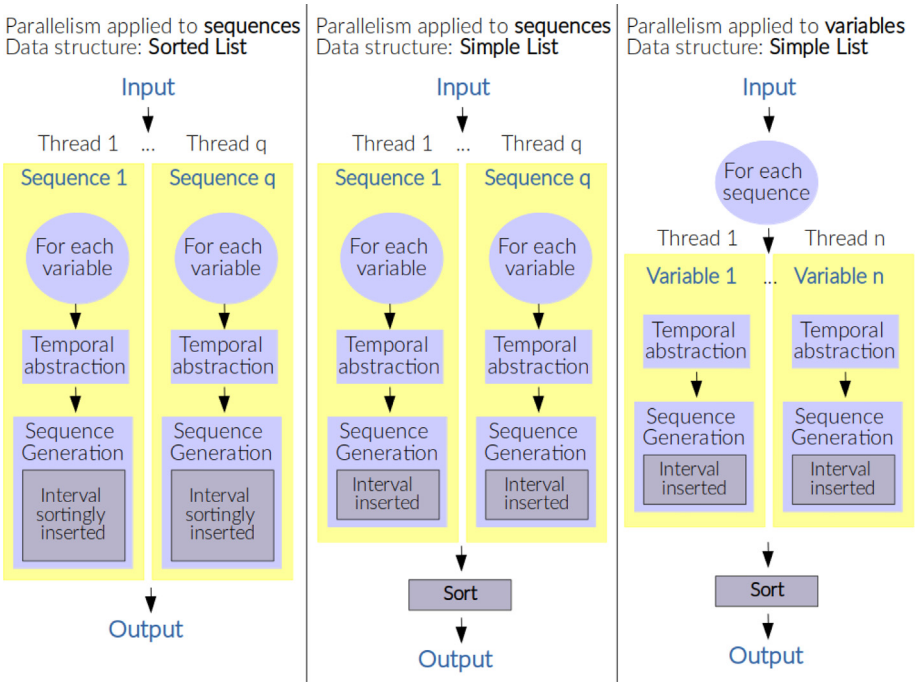


Fig. 7. Parallelism strategies of TA4L.

Table 1
Parallelism strategies of TA4L and corresponding data structures.

Name	Data structure	Parallelism strategy
TA4L_sl	Sorted list	Binary insert that remembers the last inserted element (no parallelism)
TA4L_sl_paral_seq	Sorted list	Parallelism applied to sequences (first image of the Fig. 7)
TA4L_se	Simple list	Sorts at the end (no paralellism).
TA4L_se_paral_seq	Simple list	Sorts at the end and parallelism applied to sequences (second image of the Fig. 7)
TA4L_se_paral_var	Simple list	Sorts at the end and parallelism applied to variables (third image of the Fig. 7)

- Z-normalisation
- Framing using a given amount of samples w
- Discretisation using the SAX approach
- Sequence generation with concatenation, without any gap constraint (i.e. $max\_gap = \infty$)

The state of the art approach is labelled as the Baseline in the experiments. Is it possible to observe that these configurations are close to the SAX approach; however, in a sensitivity analysis, we also test the discretisation approaches of EWD and KBTA, therefore covering a broad representation of the currently available best approaches.

All of the experiments were carried out on a virtual machine, Lubuntu 19.10, 64 bits, eight CPUs, and 12 GB of RAM. The experiments were conducted with both synthetic and real datasets under different scenarios, as described below.

5.1. Synthetic datasets

To better explore the efficiency of our algorithm, we generated numerous synthetic datasets in which the following factors were considered: (i) Percentage of missing values (PM); (ii) Number of variables (n); (iii) Number of sequences (q); (iv) Number of samples per sequence (n_{sid}). The Vocabulary size ($|\Sigma|$) and Interval size (δ) are the parameters of our algorithm.

The synthetic datasets have been created as follows. Given n, n_{sid}, and q, a dataset with this shape is created. A random part of columns in the dataset are decimal, and the rest are numeric. Then, each column from the dataset is populated with random samples from a uniform distribution over [0, 1), in the case of decimal columns; and from the "discrete uniform" distribution in the "half-open" interval [low, high) for integer columns. In the case of integer columns, the low value was set to 0, while a high value was chosen randomly from a range (0,1000000). Finally, in the same way, a filter to delete the desired percentage of missing values for each column was applied.

In addition, synthetic datasets with different autocorrelations have been generated, and additional experiments to test the behaviour of the algorithm in the function of the autocorrelation of time series was performed. In this regard, no significant changes have been detected regarding the algorithm performance. For more details, address Appendix C.

We experimented with values for PM of 1%, 25%, 50%, 75% and 99%; values for $|\Sigma|$ of 3, 10 and 20 (i.e. $\Sigma \in [A, C]$; $\Sigma \in [A, J]$; and $\Sigma \in [A, T]$); values for δ of 1%, 25% and 50% (this percentage is in respect to the total time); values for n of 3, 167, 334 and 501; values for q of 3, 167, 334 and 501; and values for n_{sid} of 3, 167, 334 and 501.

To analyse the complexity of the worst case, we generated datasets with numeric variables.

Table 2
Dataset description. If a dataset does not have discrete variables, $|\Sigma|$ is marked with "–"; otherwise, $|\Sigma|$ is the mean vocabulary size of the discrete variables.

Dataset	q	n	n_{sid}	$\lvert\Sigma\rvert$	PM	MTPS (s)
CBLS	44	2	18.136	–	2.82	540 742 254.545
$COVID_{esp}$	11	8	51.727	–	0.0	4 697 018.182
SI	38	3	134.526	4.0	0.0	347 068 800.0
MAV	11	2	116.818	30.5	0.0	4 444 193.636
$COVID_{all}$	188	14	332.761	188.0	0.918	25 056 000.0
HAR	30	562	343.3	6.0	0.0	876.288

5.2. Real datasets

The following public datasets were used for the experiments:

- *Childhood Blood Lead Surveillance (CBLS)* [27] dataset, which contains data on blood levels of lead in children from between 1995 and 2015, from several U.S. states and local health departments.
- *COVID-19 in Spain ($COVID_{esp}$)* [28] dataset, which contains data on daily increments split by age and gender of the COVID-19 pandemic in Spain.
- *Suicides in India (SI)* [29] dataset, which contains annual suicide records of all states of India with various parameters from between 2001 and 2012.
- *Mavlab (MAV) dataset* [30], which contains activities related to daily living, was collected using the MavLab testbed during March and April of 2003. This dataset captured an inhabitant's interactions with an intelligent home through sensors placed in different rooms.
- *COVID-19 by country — Daily update ($COVID_{all}$)* [31] dataset, which contains data on daily increments of the COVID-19 pandemy at the country level.
- *Human Activity Recognition (HAR)* dataset [32] built from the recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. This dataset can be found in the UCI Machine Learning Repository [33].
- *DEAP dataset (DEAP)* [34,35] which consists of information about 32 volunteers when watching 40 music videos. Each video is labelled with affective tags. For each video of each volunteer, there are 63 recordings seconds, including EEG (32 channels) and up to 8 complementary physiological signals (electrooculography, electromyography, galvanic skin response, respiration belt, plethysmography and temperature).

The datasets mentioned above are characterised in Table 2 in terms of the factors considered for the synthetic datasets, plus the mean total time per sequence (MTPS, in seconds).

5.3. Experimental scenarios

Four types of experimental configurations were defined:

1. *Performance in the function of the algorithm parameters.* This experimental configuration is directed to test the performance of the algorithm based on its parameters, such as $|\Sigma|$, δ and max_gap, and compare the results with the state of the art approaches (i.e. Baseline). The correctness of the outputs is analysed by using the Baseline approach with a perfect scenario (PM set to 0). In so doing, the number of time intervals found by the Baseline approach and the TA4L should be the same. On the other hand, when the PM increases, the number of intervals should be reduced in TA4L. Moreover, the efficiency of the data

structures proposed in TA4L is tested under the different parallelism strategies. This experiment has been applied to both synthetic and real datasets.

- To analyse the sensitivity to the size of δ, $|\Sigma|$ was set to the maximum value 20, and PM was set to 1. The hypothesis is: as the smaller is the δ, the larger the time and memory consumption.
- To analyse the sensitivity to the size of $|\Sigma|$, PM and δ was set to 1. The hypothesis is: as larger is $|\Sigma|$, more time and memory consumption will be required.
- To analyse the sensitivity to the length of *max_gap*, $|\Sigma|$ was set to 3. Since the effect of the *max_gap* depends on the number of nulls in the database, PM has been tested with 1%, 25%, 50%, 75% and 99%. The hypothesis is: as smaller is the *max_gap*, the larger the number of time intervals found, and consequently larger the memory and time consumption.

2. *Performance-based on the discretisation method.* To analyse the sensitivity to discretisation method, we compared the symbol assignation from SAX, EWD and KBTA methods in the Baseline and TA4L (TA4L_sl strategy) algorithms in the function of q, n_{sid} and $|\Sigma|$ parameters. Note that when we talk about applying SAX to TA4L, we are referring only to the task of assigning the symbol (as explained in Section 4.3). The hypothesis is: there should not be a significant difference between the runtimes of the discretisation methods (since all methods assign the symbol based on the range where the value falls), but regarding the memory consumption, SAX will be the method that consumes more memory, due to the storage of statistical tables, and the breakpoints determination. Although note that the KBTA will not go far from SAX either, as it also needs space to keep the rules set by the experts; and to simulate this, logical rules have been applied to each variable of datasets and following a common sense.
3. *Performance in the function of the dataset characteristics.* This other configuration aims to test the algorithm's performance based on the dataset characteristics, such as PM, q, n and n_{sid}. This experiment has been applied to synthetic datasets. In particular, the PM analysis aims to validate the approach followed in TA4L in which time intervals are fragmented according to known data.
   - To analyse the n **sensitivity**, $|\Sigma|$ was set to 20; q, PM and δ to 1. The discretisation method was SAX. The hypothesis is that: as more n, more time and memory will need the algorithm.
   - To analyse the n_{sid} **sensitivity**, $|\Sigma|$ was set to 20; q, PM and δ to 1. The discretisation method was SAX. The hypothesis is: as more n_{sid}, more time and memory will need the algorithm.
   - To analyse the q **sensitivity**, $|\Sigma|$ was set to 20, δ and PM to 1. The discretisation method was SAX. The hypothesis is: as larger is the q, more time and memory will need the algorithm.
4. *Performance in terms of classification.* This configuration aims to analyse how TA4L impacts classification effectiveness because, as mentioned in Section 4.2, TA4L generates sequences different from previous approaches (SAX). Since converting LSTISs to vectors of pattern characteristics to classify is a complex process, which also changes depending on the dataset, we decided to reuse the work we have in [36] prepared for the DEAP data. The idea is to apply the method described in Section 3.B of [36], using SAX and TA4L and then comparing the results. In this work, contrary to the work in [36], the minimum support of the vertTIRP algorithm was fixed to 0.7, and the Entropy threshold to 0.6 for all the labels. To cause the sequences found by the two algorithms to be different, 10 per cent of the data has been randomly deleted from the DEAP dataset. The classifiers tested are Ada Boost, Decision Tree, Gaussian Process, Naive Bayes, Nearest Neighbours, Neural Net, Random Forest and Support Vector Classifier. In this study, the parameters of the classifiers and the vertTIRP were not tuned, as improving the model's accuracy is beyond the scope of this study.

In order to be compared to the baseline approach, the epsilon parameter was set to 0.

6. Results

In this section, we estimate the cost of the TA4L algorithm, both analytically and experimentally. The results were evaluated in terms of computational runtime and memory usage.

6.1. Complexity

The complexity of the analysis has been carried out according to the different components of the algorithm. Fist, Z normalisation cost for one variable of a sequence is $n_{sid} - n_{sid} * PM$, which when simplified upwards (when $PM = 0$) becomes $O(n_{sid})$.

The cost of fragmentation and symbol assignation (which is only applicable to numeric variables) is complex. Although the algorithm loops through the values only once $O(n_{sid})$, every $100/\delta$ times, it averages the accumulated values and assigns the symbol value, where δ is represented as the percentage of time concerning 100 per cent of the total time. The symbol assignation cost is $O(|\Sigma|)$. Therefore the cost of fragmentation and symbol assignation is $O(n_{sid}+(100/\delta)*|\Sigma|)$. If the symbol of the previous I is the same as the symbol of the actual I, the end of the previous I will be updated with the actual end. This assignation has constant time complexity $O(1)$.

Z normalisation $O(n_{sid})$, fragmentation with symbol assignation $O(n_{sid} + (100/\delta) * |\Sigma|)$, and symbol insertion into the LSTIS with its' corresponding start and end times are repeated for every variable n and sequence q. Therefore, the cost of the algorithm for numeric variables is $q * (n * (n_{sid} + (100/\delta) * |\Sigma|) + insertionCost + sortingCostIfAny)$, and for discrete variables is $q * (n * (insertionCost) + sortingCostIfAny)$.

The final cost depends on the symbol insertion and sorting methods:

- Sorted insert. To find the position to insert the I, we have opted for an adapted version of binary search, that instead of searching from the beginning, searches from the last inserted position. The adapted binary search comes out at almost the same price as a normal binary search if the number of sequences is large, and the number of variables and the number of samples per sequence is small. And since this would be the worst-case scenario, we decided to count a typical binary search cost. Therefore let us consider the cost of an adapted version of binary search to be $O(log(n * n_{sid}))$, such as in a typical binary search. Here and throughout, $log(n * n_{sid}) = log_2(n * n_{sid})$ denotes the binary logarithm of $(n * n_{sid})$. Then, every time the algorithm inserts a value into position x, all the r remaining elements from the position x to the end of the list have to be moved 1 position. Therefore the cost of the sorted insert of one element is $O(r * log(n * n_{sid}))$. In the worst case, there will be $(n * n_{sid})$ insertions (i.e. $O((n * n_{sid}) * (rlog(n * n_{sid})))$), and r will be $(n * n_{sid})$

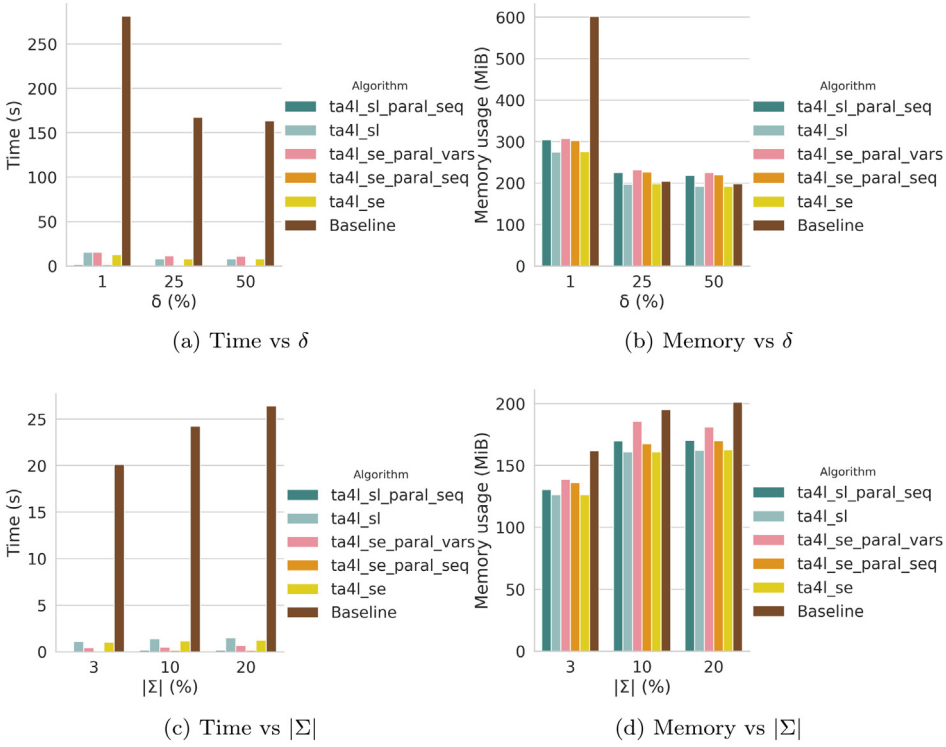


Fig. 8. Synthetic datasets. How varying input parameters δ and $|\Sigma|$ affects the performance of the algorithms.

(i.e. $O((n * n_{sid}) * (n * n_{sid}) log(n * n_{sid}))$ or simplified $O((n * n_{sid})^2 log(n * n_{sid}))$). So the cost of the algorithm in the case where all the variables are discrete is $O(q * n * ((n * n_{sid})^2 * log(n * n_{sid})))$. And the cost of the algorithm in the worst case, when all the variables are numeric (which corresponds to the final cost of the algorithm) is $O(q * (n * (n_{sid} + (100/\delta) * |\Sigma|) + (n * n_{sid})^2 * log(n * n_{sid})))$.

- Append elements normally and sort the list at the end. Append element at the end of the list, has constant time complexity i.e., $O(1)$. Since this operation will be executed n_{sid} times for each sequence and variable, the *insertionCost* in this case is $O(n_{sid})$. The complexity of sorting the list with mergesort at the end is $O((n * n_{sid}) log(n * n_{sid}))$, where $(n * n_{sid})$ is the total number of elements into a sequence of the LSTIS. So the cost of the algorithm in the case all the variables are discrete is $O(q * ((n * n_{sid}) + (n * n_{sid}) * log(n * n_{sid})))$. And the cost of the algorithm in the worst case, when all the variables are numeric is $O(q * (n * (n_{sid} + (100/\delta) * |\Sigma|) + ((n * n_{sid}) log(n * n_{sid}))))$.

6.2. Algorithm behaviour

In this section, we analyse experimentally the behaviour of algorithms based on input parameters, such as Σ, δ and *max_gap*. Fig. 8 shows the results obtained with the synthetic datasets, and Figs. 9 and 10 for the real ones.

We can observe that in the case of both synthetic and real datasets, time and memory increase with increasing vocabulary size and with decreasing interval size. In real datasets (Figs. 9 and 10) the increase in time and memory is most noticeable with large datasets, such as *HAR* and *COVID_all* datasets.

Multithread version by sequence (TA4L_sl_paral_seq and TA4L_se_paral_seq) consumes less time for both parameters (δ and $|\Sigma|$).

The parallel version by variables is the structure that consumes slightly more memory than the rest of the TA4L versions.

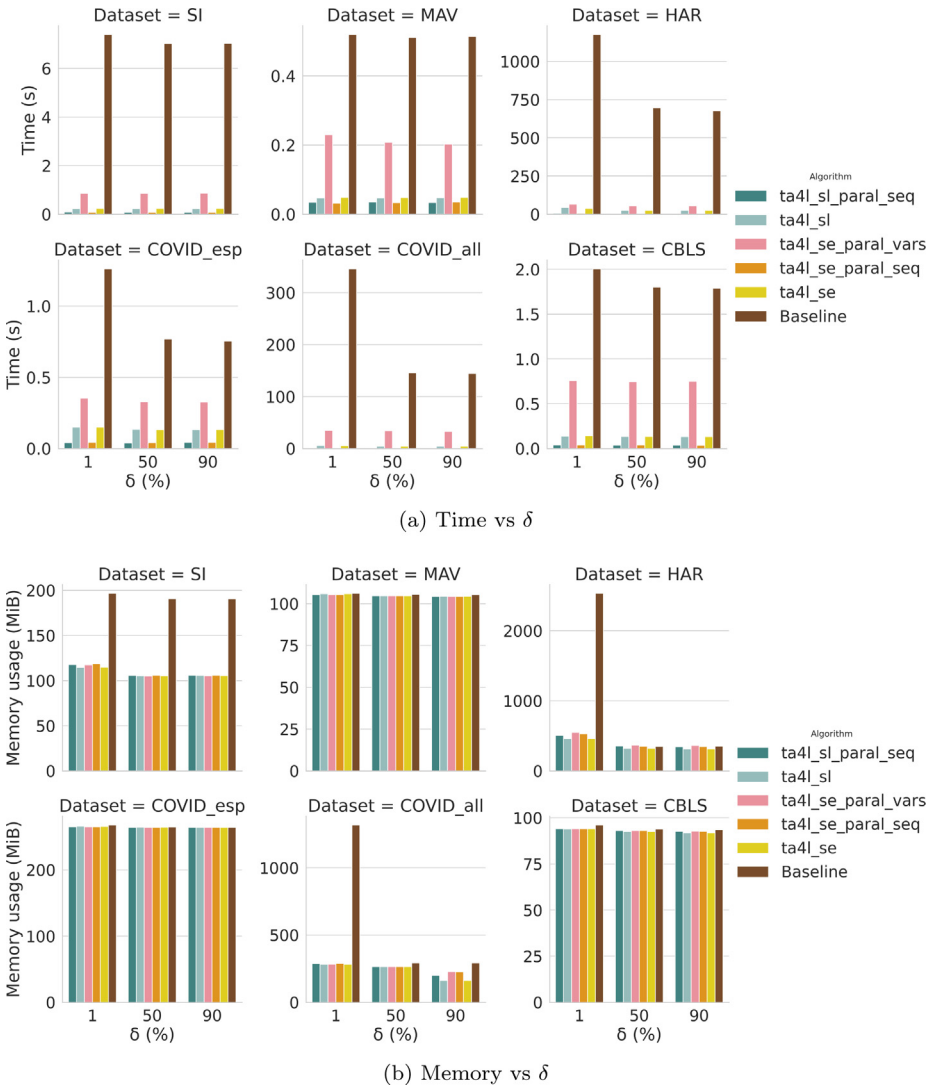
In general, in all real datasets, TA4L is significantly faster than Baseline for all configurations, and in terms of memory consumption, TA4L is more efficient than Baseline for relatively large datasets, such as HAR, *COVID_all* and IS datasets.

Regarding the *max_gap* constraint (see Fig. 11), on the one hand, we can appreciate that time and memory consumption are directly proportional to the number of the constructed time intervals (|I|). The more |I|, the greater time and memory consumption. For small values of *max_gap*, there are more |I| (1 624 261 |I| for *max_gap* 1 and missing 25%), and for big values of *max_gap* there are less |I| (1 341 094 |I| for *max_gap* 10 and missing 25%). For example, in the case of 25% of missing values, for *max_gap* = 1% the |I| is 1 624 261, the runtime is 61.7 s, and the memory usage is 552.7 MiB, while from the *max_gap* = 10% on the |I| is 1 341 094, the runtime is 58.5 s, and the memory usage is 510.3 MiB. On the other hand, this effect is less pronounced in cases of extreme missing values, that is, if the percentage of missing data is too small (1%) or too large (99%). For example, in the case of 1% of missing values, for *max_gap* = 1% the |I| is 1 546 397, the runtime is 62.5 s, and the memory usage is 543.5 MiB, while for the *max_gap* = 10% the |I| is 1531279, the runtime is 62.2 s, and the memory usage is 541.42 MiB.

In the case of *PM* values, on the one hand, we can appreciate that Baseline discovers fewer time intervals than the TA4L approach (see Fig. 12), due to the effect of missing values. And on the other hand, there is a decrease in time, memory and number of time intervals with an increase of PM values, confirming the hypothesis. And as for the number of Time intervals, we can check that when there are no missing values, the number of time intervals is the same for all versions, which indicates that the segmentation is done correctly.

6.3. Performance according to the discretisation method

This section presents the results of applying EWD, KBTA and SAX discretisation methods to the TA4L and Baseline approaches, based on the dataset characteristics such as n, q and n_{sid}.



(a) Time vs δ

(b) Memory vs δ

Fig. 9. Real datasets. How varying the δ affects the performance of the algorithms.

6.3.1. Synthetic datasets

In the case of synthetic datasets, in general for small values of q, n, and n_{sid} SAX and KBTA consume significantly less time and memory than the EWD approach, while for large values of q, n, and n_{sid} EWD consumes less memory and runtime than SAX (see Figs. 13–15). The baseline is significantly slower than TA4L (observe that time scales of plots in the figures are different in TA4L from Baseline, due to the big difference in the results achieved), and it also consumes more memory than the TA4L approach.

6.3.2. Real datasets

In the case of real datasets, TA4L generally consumes less time and memory than Baseline (observe the differences on the time scales of plots in the figures as happened with the synthetic datasets). Nevertheless, in TA4L implementations, the behaviour of the discretisation approach requires a deepening analysis.

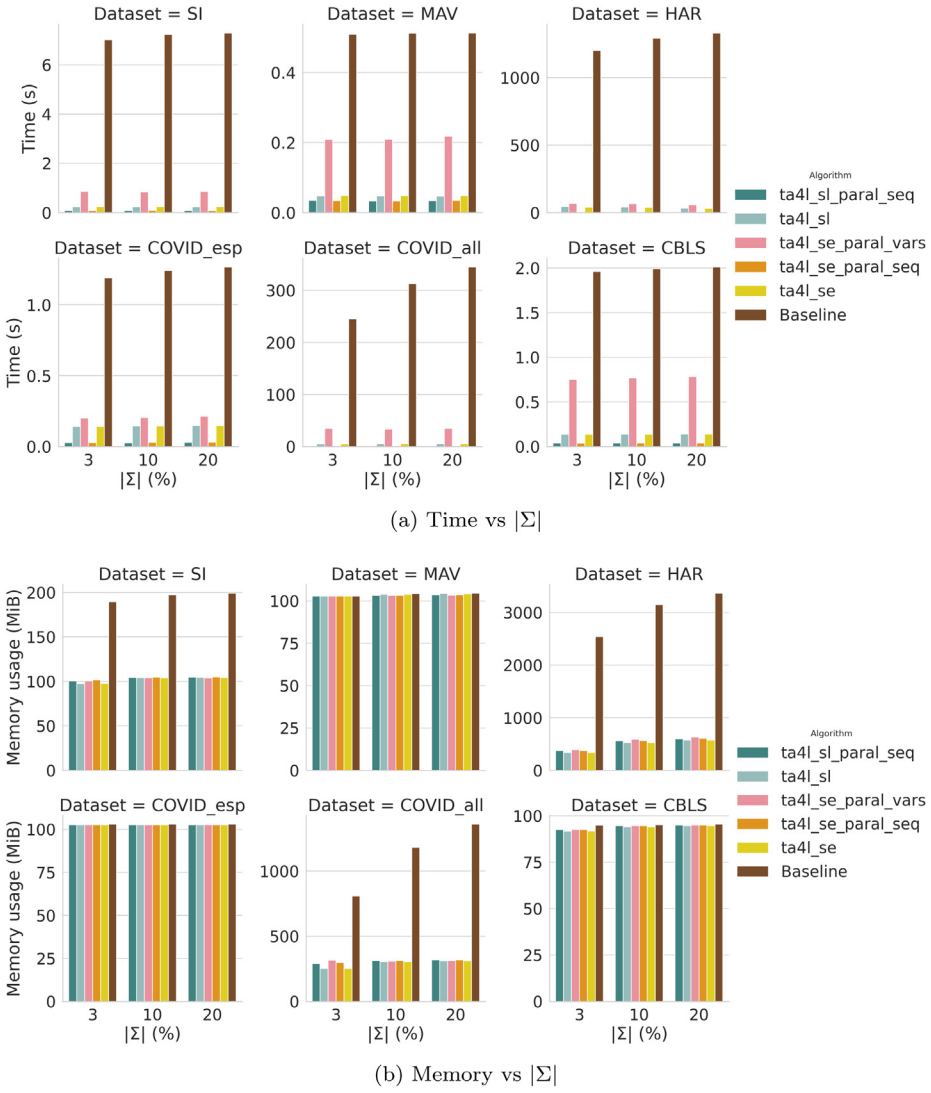
For example, with respect to memory consumption, in the case of the HAR dataset, for large q, n, and n_{sid} values (Figs. 16–18), Baseline_KBTA is the one that consumes significantly more memory than Baseline_EWD. Whereas in the case of the COVID_all dataset, for large values of q, n and n_{sid}, Baseline_SAX consumes significantly more memory than Baseline_EWD. In the case of large datasets (such as HAR and COVID_all datasets), for small values of q, n and n_{sid}, TA4L_SAX is the least memory consuming than TA4L_EWD and TA4L_KBTA methods.

Regarding the time consumption (Figs. 19–21), in the case of the COVID_all dataset, Baseline_EWD is significantly slower than Baseline_KBTA, especially for n_{sid} experiments. Regarding the sensitivity to discretisation in TA4L, in the case of the HAR dataset, in both n and n_{sid} experiments, SAX is significantly slower than the EWD and KBTA methods.

6.4. Performance according to the dataset characteristics

In this section, we analyse experimentally the cost of algorithms based on dataset characteristics, such as q, n and n_{sid}.

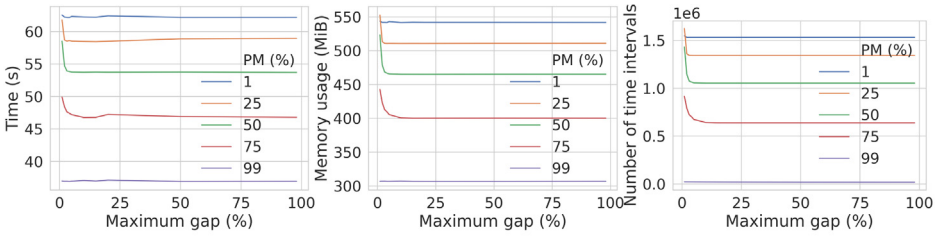
In the case of q (Figs. 22(d) and 22(c)), multithread version by sequence consumes less time for both data structures, which is around 3.8 s for q 501 (worst case). The state of the art approach is that consumes more time (732.19 s for q 501). However, the multithread version of TA4L by variables also presents a great runtime concerning other versions of TA4L, which is 35.22 s for q 500. Therefore, if we have many sequences and few variables, it is not useful to parallelise by variables. In the case of n (Figs. 22(b) and 22(a)) and n_{sid} (Figs. 23(b) and 23(a)) the two multithread versions by sequence also present the shortest runtime (0.134 s for n 501 and 1.39 s for PM 1), while the state of the art is



(a) Time vs $|\Sigma|$

(b) Memory vs $|\Sigma|$

Fig. 10. Real datasets. How varying the $|\Sigma|$ affects the performance of the algorithms.



(a) Time vs max_gap (b) Memory vs max_gap (c) |I| vs max_gap

Fig. 11. Synthetic datasets. How varying the *PM* and *max_gap* parameters affects the performance of the algorithms, where |I| is the total number of time intervals.

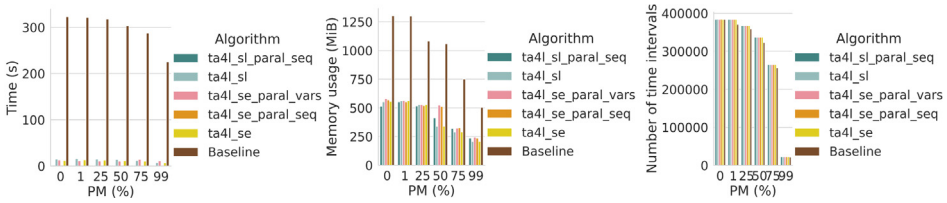


Fig. 12. How varying *PM* affects the performance of the algorithms, where |I| is the total number of time intervals.

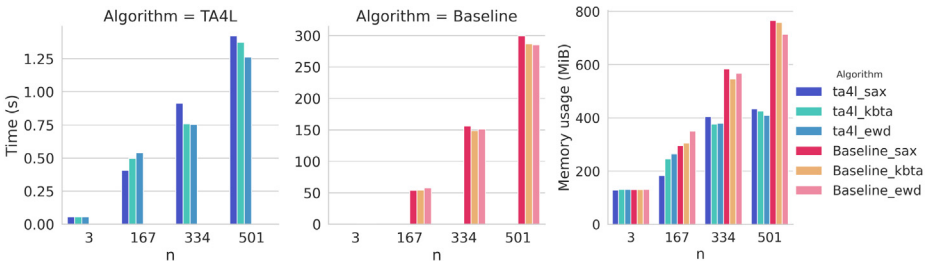


Fig. 13. Synthetic datasets. Sensitivity to discretisation methods while varying n.

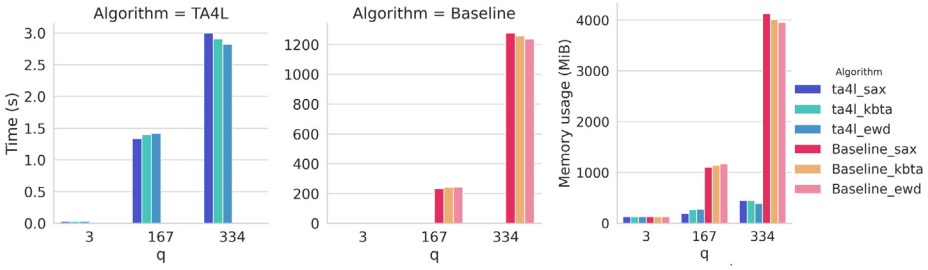


Fig. 14. Synthetic datasets. Sensitivity to discretisation methods while varying q.

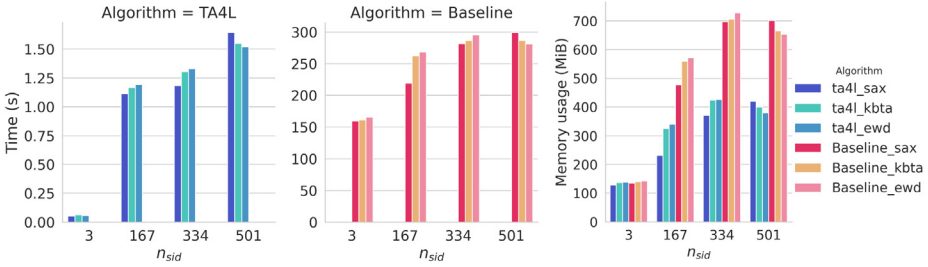


Fig. 15. Synthetic datasets. Sensitivity to discretisation methods while varying n_{sid}.

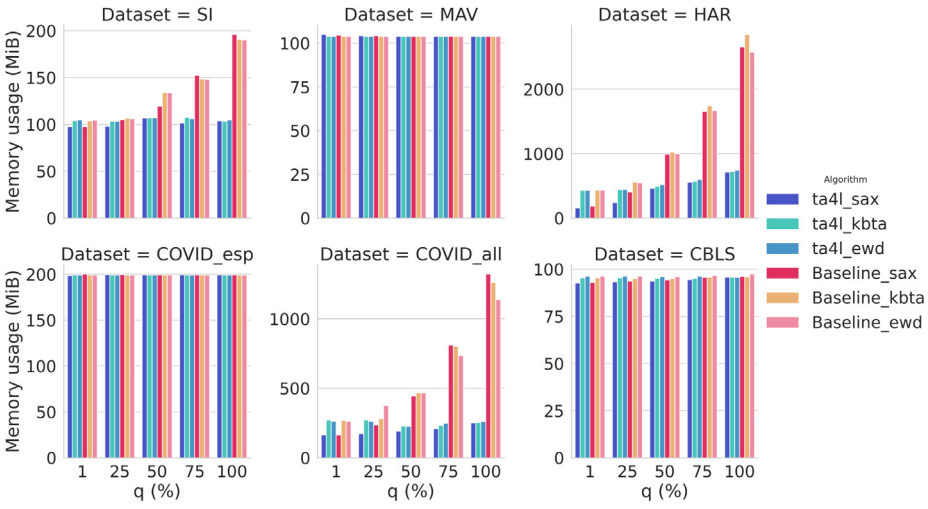


Fig. 16. Real datasets. Sensitivity to discretisation methods in terms of memory while varying q.

the longest one (23.75 s for n 501 and 321.33 for PM 1). And regarding memory consumption, the state of the art approach consumes more memory. In the TA4L versions, all algorithms consume approximately the same memory, except the parallel versions that consume more, especially the parallel version of TA4L by variables.

6.5. Performance in terms of classification

In this section, we analyse whether TA4L improves the effectiveness of the pre-processing in terms of classification compared to SAX.

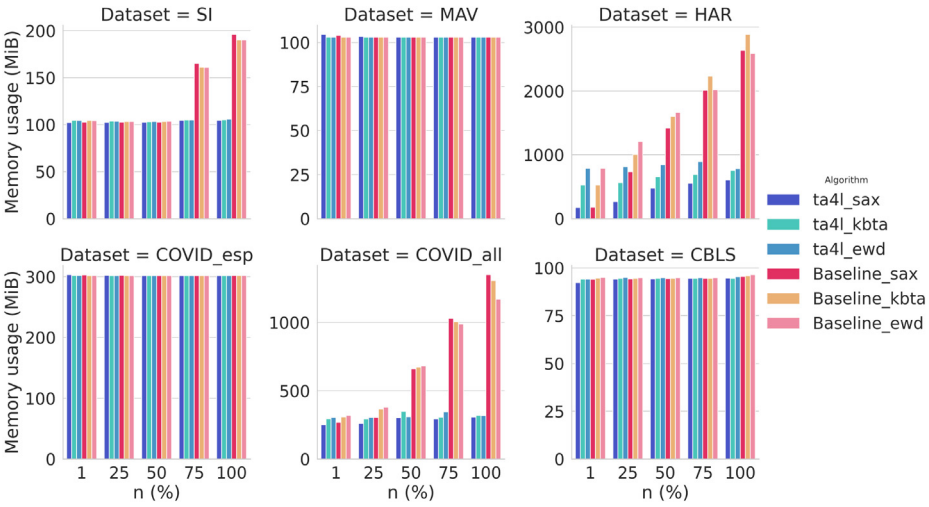


Fig. 17. Real datasets. Sensitivity to discretisation methods in terms of memory while varying n.

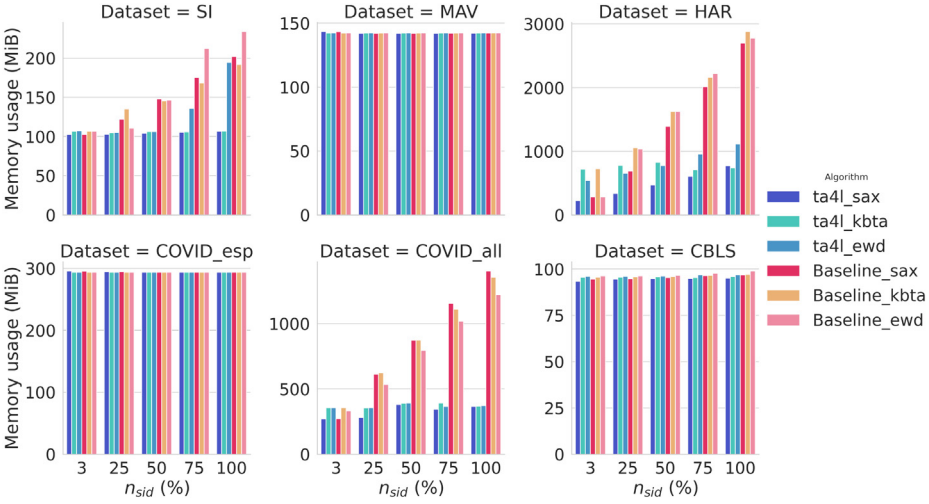


Fig. 18. Real datasets. Sensitivity to discretisation methods in terms of memory while varying n_{sid}.

Note that there is a graph for each class (see Fig. 24): arousal, dominance, liking, valence. The graph's Y-axis is the accuracy of the algorithms, and the X-axis of the graph is the different classification algorithms used to make the predictions. The blue bars are the Baseline, and the oranges are the TA4L.

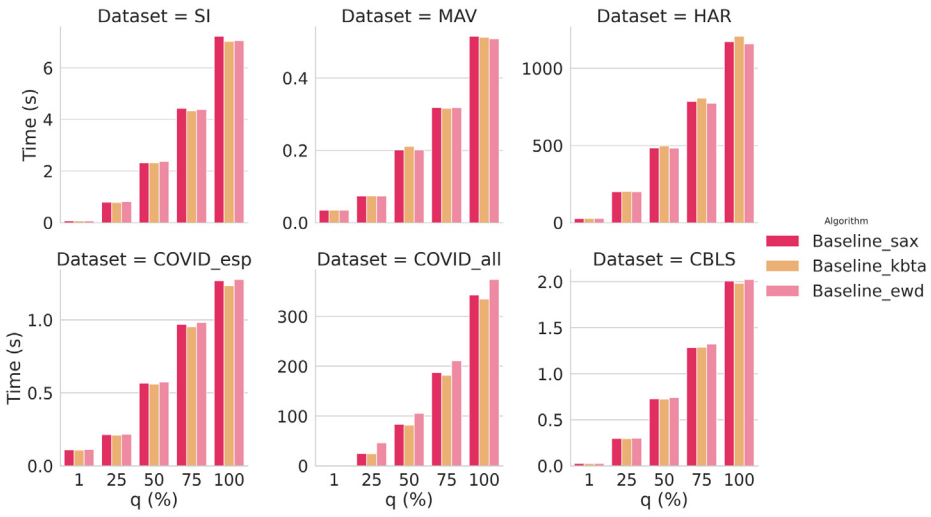
Because the sequences generated with TA4L vary from those generated with Baseline, the results also vary. The best combination for dominance, liking, and valence was TA4L + SVC, with 70.8182, 76.8479 and 68.1222 scores. And the best combination for the arousal was TA4L/Baseline + SVC, with a score of 61.63793. In general (that is, in 80 per cent of all experiments), better or equal results were obtained with TA4L than with Baseline.

6.6. Discussion

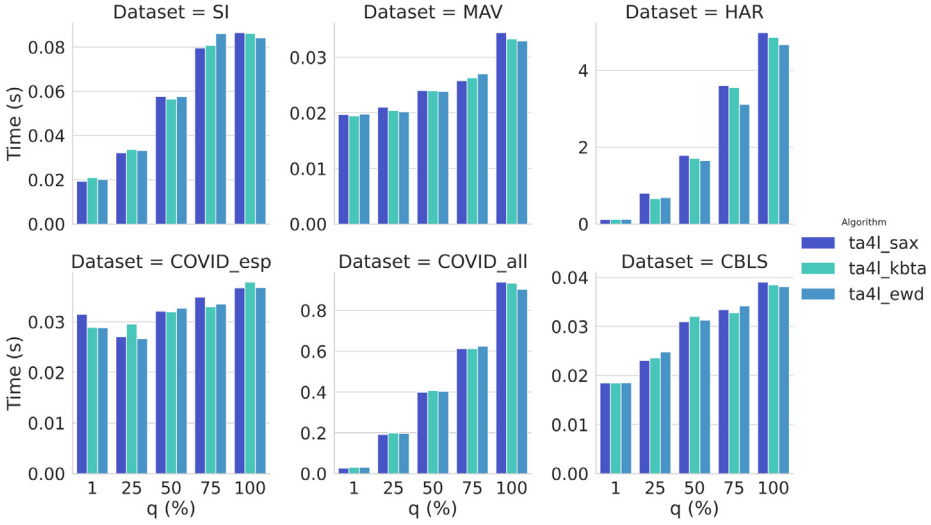
One of the first insights from the experimental analysis regarding the dataset characteristics is that it is observed that there is a decrease in time, memory and number of time intervals constructed with an increase in PM values, since as larger is *PM* values, fewer data per interval, and therefore, fewer time intervals constructed, memory usage and runtime. This also applies to the state-of-the-art algorithm, in which, first, each value is discretised and then concatenated. Another point to highlight regarding missing values is that in the presence of missing values, previous algorithms (SAX) generate the sequence AB for the same series (Fig. 4) generates the sequence AB, while TA4L ABCB. That means that the patterns found from this input by any TIRP mining algorithm will differ.

Regarding the algorithm parameters, as smaller is the δ, more intervals are generated, and therefore the algorithm performs more mean computations, symbol assignations and insertions, and therefore more time and memory consumption. Differently, the larger the delta, the more critical information is lost, and a small delta value will not remove noise enough. The choice of the delta should be established empirically and progressively. For example, suppose a posteriori is intended to do TIRP mining and classification. In that case, one could start with a small delta (i.e. 2% of total time) and increase it progressively until an improvement in accuracy is achieved.

And as larger is the $|\Sigma|$, more iterations performs the algorithm in the symbol assignation stage, and therefore more time and memory consumption. This behaviour is reflected in all versions of datasets and algorithms. Concerning *max_gap*, as smaller is the *max_gap* parameter, the algorithm performs fewer concatenations between the adjacent intervals, and therefore more intervals are generated, and consequently, more time and memory consumption is required. As expected, this effect is less noticeable



(a) Baseline: Time vs q



(b) TA4L: Time vs q

Fig. 19. Real datasets. Sensitivity to discretisation methods in terms of time while varying q.

if the dataset has a very small percentage of missing values (there are no gaps) or a very large number of missing values (low number of time intervals). In case of knowing the problem, the max_gap is the maximum time allowed between two consecutive events so that these two successive events can be considered one single event. The correct choice of the size of the sigma and the max_gap parameter without expert knowledge could be made in a similar way to the selection of the delta.

In terms of discretisation, in the case of synthetic datasets, for large q, n, and n_{sid} values EWD is the more memory efficient method, while for small values of q, n and n_{sid}, SAX is that consumes less time nor memory. In the case of real datasets, we were able to verify that, in general, there is no significant difference between the methods in terms of time and memory consumption. However, concerning memory consumption, in the case of large datasets (such as HAR or COVID_all datasets), for large q, n, and n_{sid} values, EWD is the more memory efficient method, while for small values of q, n and n_{sid}, SAX is the least memory consuming method.

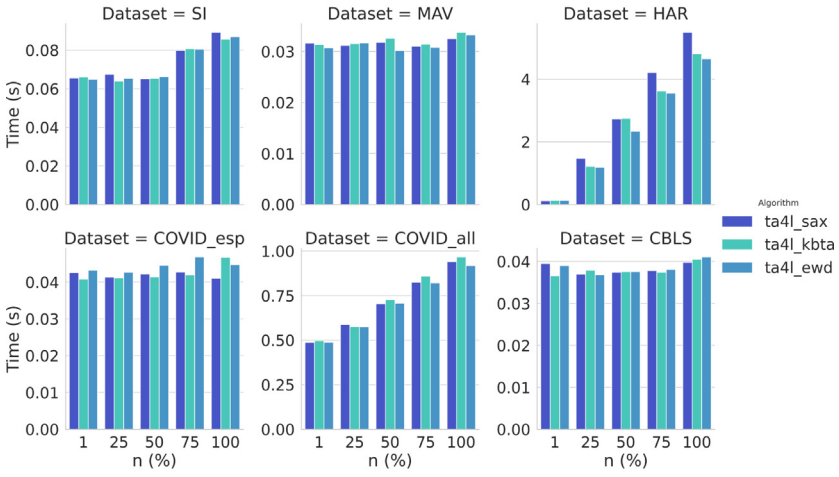
The cost is greater for numeric variables and a sorted-list like structure. Sorting all at once has a lot more design freedom than maintaining the ordering incrementally since an incremental update has to maintain a complete order at all times, which represents more time and memory consumption. And regarding multithreading, if we have a sudden processor and RAM, the best option is to parallelise the TA4L algorithm per sequence instead of per variable.

And in terms of classification, TA4L does not always have to be better than the Baseline in downstream tasks. TA4L is suitable for datasets where situations like the one described in Fig. 4 occur. A priori, it could be datasets with missing values, and where the duration of events plays an essential role in the prediction.

7. Conclusions

This paper presented the TA4L algorithm that converts multivariate time series into sequences known as LSTISs (symbolic time-intervals sorted by time and symbols). In the literature, it is not easy to find a specific pre-processing algorithm. Usually, pre-processing is explained in a subsection of a TIRP mining algorithm description. On the one hand, the TA4L algorithm aims to make the process explicitly and provide a solution based on a temporal abstraction and a sequence generation method to accelerate the pre-processing time, merging all the pre-processing tasks to be

(a) Baseline: Time vs n



(b) TA4L: Time vs n

Fig. 20. Real datasets. Sensitivity to discretisation methods in terms of time while varying n.

executed together in a single algorithm. On the other hand, in TA4L, instead of reducing the dimensionality (from time points to time-intervals) based on the number of samples (as is done in the state of the art algorithms, as SAX [5]), it is reduced based on a time duration that is adjusted to the real known values, guaranteeing a reliable posterior TIRP mining application with the LSTISs obtained. Moreover, the size of the time intervals is managed by a given *max_gap* constraint that enables controlling the maximum separation of two points to be considered part of the same event (i.e. symbol), a property of particular interest in different application domains.

Finally, TA4L considers a sorting list data structure together with several parallelism strategies (variable and sequence multi-threads) that make the whole pre-processing efficient.

The experimentation carried out in different synthetic and real datasets show that the TA4L cost varies in function of the data structure used and the type of the variables in the dataset (numeric or discrete). In general, TA4L provides a noticeable reduction in time and memory than the state of the art algorithm. Multithread version by variables is useless in the case the dataset has few variables and many sequences. On the contrary, the multithread version by sequence consumes less time for all the experiments.

An interesting direction for future work involves adapting the TA4L algorithm to an incremental version. However, an imminent future work includes classifying the TIRPs discovered from the intervals obtained with TA4L and those obtained from the Baseline and comparing the results. This experiment would allow us to power the results regarding the reliability of the intervals obtained from TA4L.

Declaration of competing interest

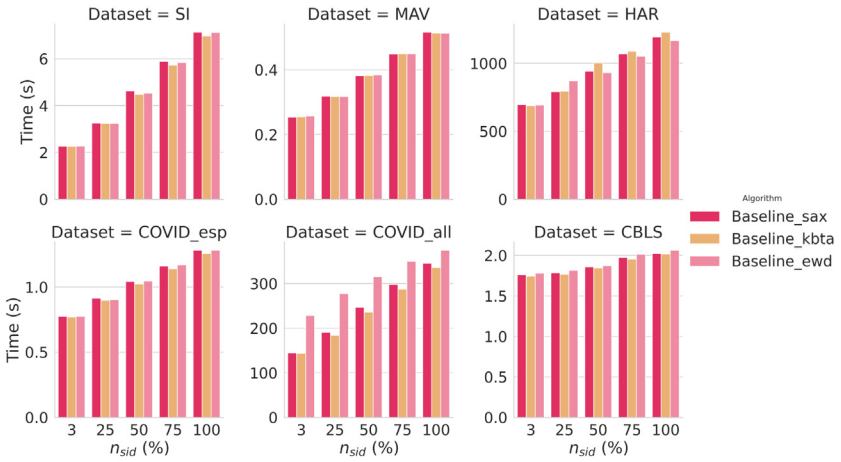
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

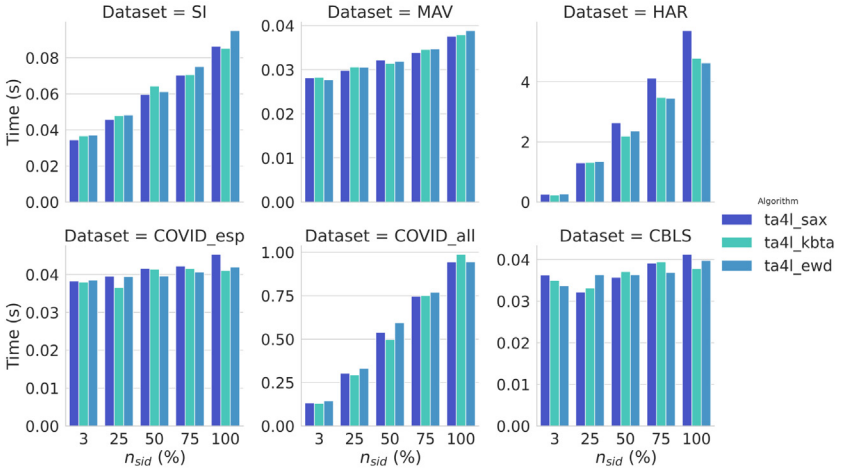
This project received joint funding from ERDF, the Spanish Ministry of the Economy, Industry and Competitiveness (MINECO) and the National Agency for Research, under grant no. RTC 2017-6071-1 (SERAS). The work was carried out with support from the Generalitat de Catalunya 2017 SGR 1551, a predoctoral grant from the University of Girona (grants for researchers in training/IFUdG2017) and a mobility grant (additional support for the mobility of UdG researchers/MOB2019).

Appendix A. Implementation of TA4L

Our implementation of TA4L is available from the Bitbucket repository at “git clone https://invited2bynatalia@bitbucket.org/

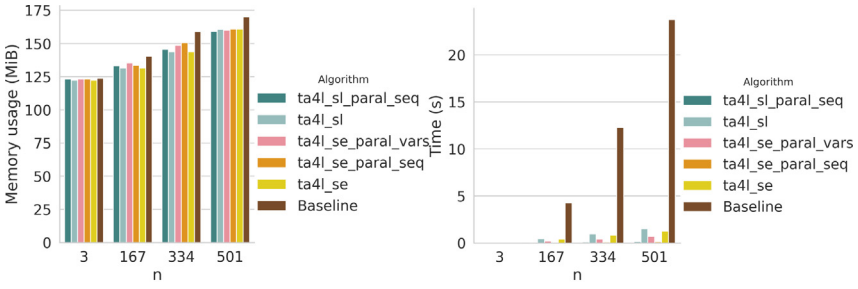


(a) Baseline: Time vs n_{sid}

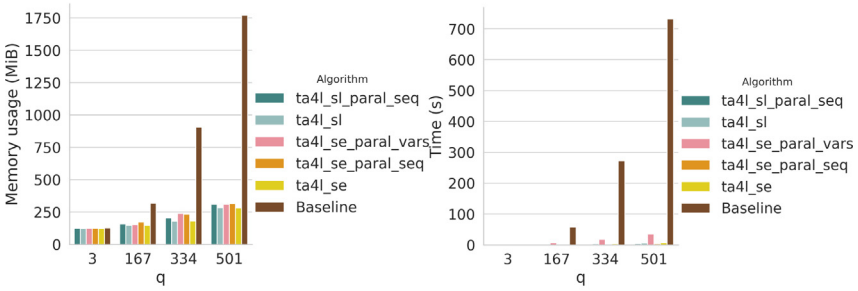


(b) TA4L: Time vs n_{sid}

Fig. 21. Real datasets. Sensitivity to discretisation methods in terms of time while varying n_{sid}.

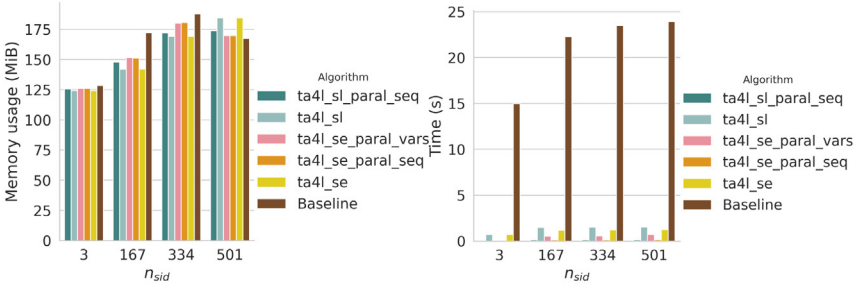


(a) Memory vs n (b) Time vs n



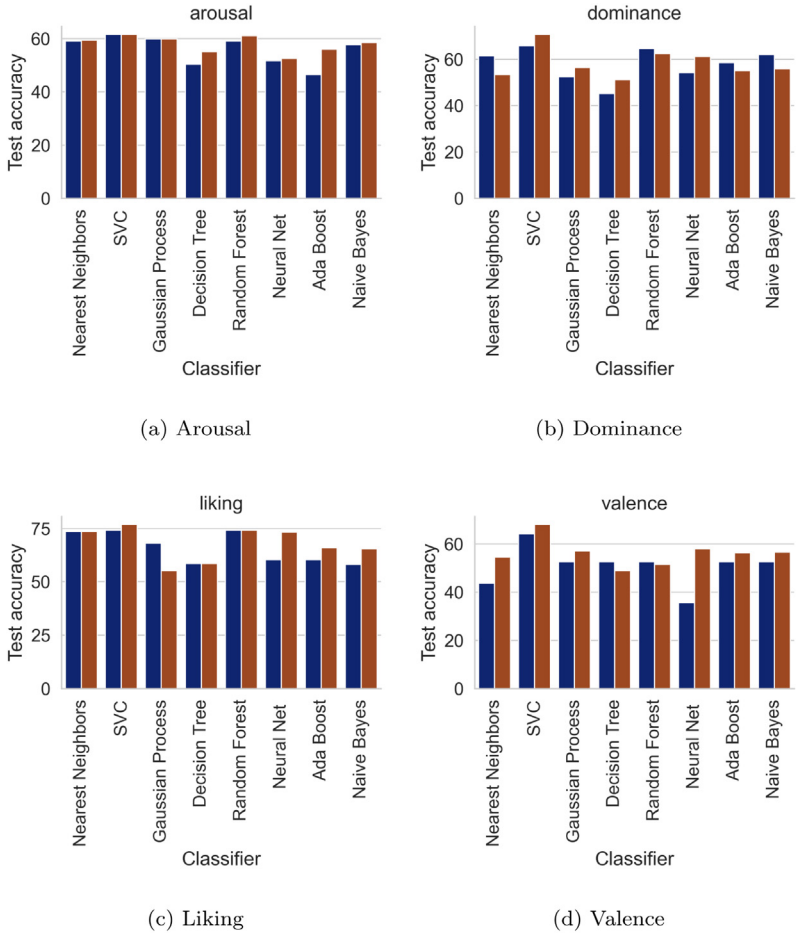
(c) Memory vs q (d) Time vs q

Fig. 22. Synthetic datasets. How varying n and q affects the performance of the algorithms.



(a) Memory vs n_{sid} (b) Time vs n_{sid}

Fig. 23. Synthetic datasets. How varying n_{sid} affects the performance of the algorithms.



(a) Arousal (b) Dominance

(c) Liking (d) Valence

Fig. 24. How TA4L impacts the performance of classification. In this chart, SVC refers to the Support Vector Classifier. The blue bars are the Baseline, and the oranges are the TA4L.

natalia_mordvanyuk/td4l.git". The following username and password can be used to log in:

username: invited2bynatalia
email: invited2bynatalia@gmail.com
password: 2wGW4hxgMVUtGkF..,

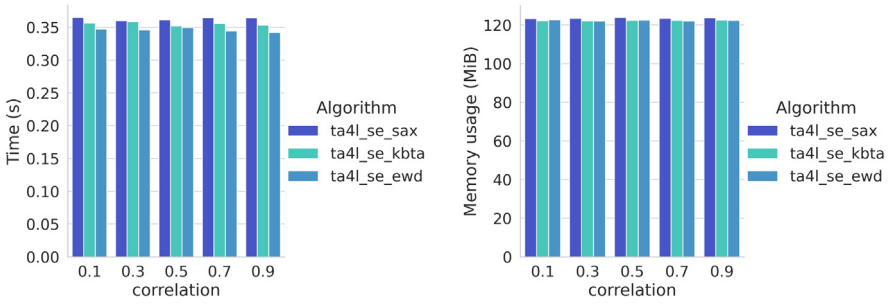
Appendix B. Auxiliar functions of TA4L

This section provides auxiliary functions used in the main TA4L algorithm. The first one is the FDC function.

The inputs of the FDC algorithm include time-series *TS*, the maximum gap *max_gap* allowed between two consecutive time intervals, the minimum duration of time intervals *duration*, and $LSTIS_{seq}$, that is the LSTIS for the sequence *seq*. The algorithm returns $LSTIS_{seq}$ updated with all new time intervals.

The first step of the algorithm consists of obtaining the first two TIs (the previous TI and the actual TI). If the previous and the actual TI have the same symbol, the end of the previous TI will be updated with the end of the actual TI unless the gap between them is greater than max_gap. The algorithm introduces the previous TI to the LSTIS in each iteration. The current TI is inserted when there are no more elements in the series.

The inputs of the SCS algorithm include time-series *TS*, the maximum gap *max_gap* allowed between two consecutive time



(a) Time vs δ (b) Memory vs δ

Fig. C.25. Synthetic datasets. How varying the auto-correlation of the time series affects the performance of the algorithm.

Algorithm 2: FDC

input: TS: multivariate time series
max_gap: the maximum gap constraint
duration: duration of the interval
$LSTIS_{seq}$: list of LSTIs
output: $LSTIS_{seq}$: list of LSTIs for the sequence 'seq'

```
i = 1
<TI_prev, i> = updateTI(TS, i, duration, NULL, max_gap)
<TI_curr, i> = updateTI(TS, i, duration, TI_prev,max_gap)
while i < len(TS) do
    while (TI_curr != NULL) and (TI_prev.sym == TI_curr.sym) do
        <TI_curr, i> = updateTI(TS, i, duration, TI_prev,max_gap)
    LSTIS_seq = insert(TI_prev, LSTIS_seq)
    if (i >= len(TS)) and TI_curr != NULL then
        /* last element                                    */
        if (TI_prev != TI_curr) then
            LSTIS_seq = insert(TI_curr,LSTIS_seq)
    TI_prev = TI_curr
return LSTIS_seq
```

Algorithm 3: SCS

input: TS: multivariate time series
max_gap: the maximum gap constraint
$LSTIS_{seq}$: list of LSTIs
output: $LSTIS_{seq}$: list of LSTIs for sequence 'seq'

```
1  i = 1
2  while i < (len(TS)-1) do
3      sym_cur = getValue(TS, i)
4      sym_next = getValue(TS, i+1)
5      start_TI = getTimestamp(TS, i)
6      gap = getTimestamp(TS, i+1) - getTimestamp(TS, i)
7      while (i < (len(TS)-1)) and (sym_cur == sym_next) and
        gap<max_gap do
8          i = i+1
9          sym_cur = getValue(TS, i)
10         sym_next = getValue(TS, i+1)
11         gap = getTimestamp(TS, i+1) - getTimestamp(TS, i)
12     end_TI = getTimestamp(TS, i)
13     TI_curr = TI(sym_cur, start_TI,end_TI)
14     LSTIS_seq = insert(TI_prev,LSTIS_seq)
15 if i > 1 and i < (len(TS) then
16     last_time = getTimestamp(TS, i)
17     gap = getTimestamp(TS,i)-getTimestamp(TS,i-1)
18     if (getValue(TS,i) == getValue(TS,i-1) and gap <=
        max_gap then
19         LSTIS_seq = changeEnd(TI_curr, last_time, LSTIS_seq)
20     else
21         TI_curr = TI(sym_cur, last_time, last_time)
22         LSTIS_seq = insert(TI_prev,LSTIS_seq)
23 return LSTIS_seq
```

intervals and the $LSTIS_{seq}$, that is the LSTIS for the sequence *seq*. The algorithm returns $LSTIS_{seq}$ updated with all new time intervals.

We can see that in the loop, in each iteration, a new TI is being added within LSTISs. To create the TI, you need the symbol, the start time, and the end time of the TI. The symbol and the initial time are set at the beginning of the loop. If the current symbol is equal to the previous symbol and the maximum gap is not exceeded, the counter is incremented to determine the final time. If any of these conditions are not met, the final time is set.

When we reach the end of the time series, it must be decided whether to change the final time of the last TI or whether to create a new TI and insert it into LSTISs. This decision is made in the lines 17–23.

Appendix C. TA4L on autocorrelated time series

This section includes the experiments related to the behaviour of the algorithm on autocorrelated time series. Fig. C.25 shows the consumption of time and memory as a function of the different degrees of time series autocorrelation. In this regard, no significant changes have been detected regarding the algorithm performance.

References

[1] P. Fournier-Viger, J.C.-W.L. School, R.U.K. University, Y.S. Koh, R. Thomas, A survey of sequential pattern mining, in: Data Science and Pattern Recognition, Vol. 1, 2017, pp. 54–77.
[2] P.-s. Kam, A.W.-c. Fu, Discovering temporal patterns for interval-based events, in: Y. Kambayashi, M. Mohania, A.M. Tjoa (Eds.), Data Warehousing and Knowledge Discovery, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 317–326, http://dx.doi.org/10.1007/3-540-44466-1_32.
[3] R. Moskovitch, Y. Shahar, Classification of multivariate time series via temporal abstraction and time intervals mining, Knowl. Inf. Syst. 45 (1) (2015) 35–74, http://dx.doi.org/10.1007/s10115-014-0784-5.
[4] R. Moskovitch, Y. Shahar, Classification-driven temporal discretization of multivariate time series, Data Min. Knowl. Discov. 29 (2015) 871–913, http://dx.doi.org/10.1007/s10618-014-0380-z.
[5] J. Lin, E. Keogh, S. Lonardi, B. Chiu, A symbolic representation of time series, with implications for streaming algorithms, in: Proceedings of the 8th ACM

SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003, pp. 2–11.
[6] E. Winarko, J.F. Roddick, ARMADA - an algorithm for discovering richer relative temporal association rules from interval-based data, Data Knowl. Eng. 63 (1) (2007) 76–90, http://dx.doi.org/10.1016/j.datak.2006.10.009.
[7] L. Hui, Y.C. Chen, J.T.Y. Weng, S.Y. Lee, Incremental mining of temporal patterns in interval-based database, Knowl. Inf. Syst. 46 (2) (2016) 423–448, http://dx.doi.org/10.1007/s10115-015-0828-5.
[8] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the Eleventh International Conference on Data Engineering, 1995, pp. 3–14, http://dx.doi.org/10.1109/ICDE.1995.380415.
[9] R. Moskovitch, Y. Shahar, Medical temporal-knowledge discovery via temporal abstraction, in: AMIA 2009 Symposium Proceedings, 2009, pp. 452–456, URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2815492/.
[10] R. Moskovitch, Y. Shahar, Fast time intervals mining using the transitivity of temporal relations, Knowl. Inf. Syst. 42 (1) (2013) 21–48, http://dx.doi.org/10.1007/s10115-013-0707-x.
[11] S.Y. Wu, Y.L. Chen, Mining nonambiguous temporal patterns for interval-based events, IEEE Trans. Knowl. Data Eng. 19 (6) (2007) 742–758, http://dx.doi.org/10.1109/TKDE.2007.190613.
[12] D. Patel, W. Hsu, M.L. Lee, Mining relationships among interval-based events for classification, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 393–404, http://dx.doi.org/10.1145/1376616.1376658.
[13] Y.C. Chen, W.C. Peng, S.Y. Lee, Mining temporal patterns in interval-based data, in: 2016 IEEE 32nd International Conference on Data Engineering, in: ICDE 2016, vol. 27, IEEE, 2016, pp. 1506–1507, http://dx.doi.org/10.1109/ICDE.2016.7498397.
[14] C.W. Yang, B.P. Jaysawal, J.W. Huang, Subsequence search considering duration and relations of events in time interval-based events sequences, in: Proceedings - 2017 International Conference on Data Science and Advanced Analytics, in: DSAA 2017, vol. 2018-Janua, 2018, pp. 293–302, http://dx.doi.org/10.1109/DSAA.2017.47.
[15] J.-W. Huang, B.P. Jaysawal, K.-Y. Chen, Y.-B. Wu, Mining frequent and top-k high utility time interval-based events with duration patterns, Knowl. Inf. Syst. 61 (3) (2019) 1331–1359, http://dx.doi.org/10.1007/s10115-019-01333-6.
[16] P. Papapetrou, G. Kollios, S. Sclaroff, D. Gunopulos, Mining frequent arrangements of temporal intervals, Knowl. Inf. Syst. 21 (2) (2009) 133–171, http://dx.doi.org/10.1007/s10115-009-0196-0.
[17] Y. Shahar, A framework for knowledge-based temporal abstraction, Artificial Intelligence 90 (1) (1997) 79–133, http://dx.doi.org/10.1016/S0004-3702(96)00025-2, URL https://www.sciencedirect.com/science/article/pii/S0004370296000252.
[18] R. Azulay, R. Moskovitch, D. Stopel, M. Verduijn, E. de Jonge, Y. Shahar, Temporal discretization of medical time series-a comparative study, in: Proceedings of IDAMAP2007: Intelligent Data Analysis in Biomedicine and Pharmacology, Vol. 43, 2007, pp. 48–58.
[19] E. Keogh, S. Chu, D. Hart, M. Pazzani, Segmenting time series: A survey and novel approach, in: Data Mining in Time Series Databases, World Scientific, 2004, pp. 1–21, http://dx.doi.org/10.1142/9789812565402_0001.
[20] F. Mörchen, A. Ultsch, Optimizing time series discretization for knowledge discovery, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2005, pp. 660–665, http://dx.doi.org/10.1145/1081870.1081953.
[21] S.S. Titarenko, V.N. Titarenko, G. Aivaliotis, J. Palczewski, Fast implementation of pattern mining algorithms with timestamp uncertainties and temporal constraints, J. Big Data 6 (1) (2019) 37, http://dx.doi.org/10.1186/s40537-019-0200-9.
[22] J. Dougherty, R. Kohavi, M. Sahami, Supervised and unsupervised discretization of continuous features, in: A. Prieditis, S. Russell (Eds.), Machine Learning Proceedings 1995, Morgan Kaufmann, San Francisco (CA), 1995, pp. 194–202, http://dx.doi.org/10.1016/B978-1-55860-377-6.50032-3, URL https://www.sciencedirect.com/science/article/pii/B9781558603776500323.
[23] C. Antunes, A.L. Oliveira, Generalization of pattern-growth methods for sequential pattern mining with gap constraints, in: International Workshop on Machine Learning and Data Mining in Pattern Recognition, Springer, 2003, pp. 239–251, http://dx.doi.org/10.1007/3-540-45065-3_21.
[24] C. Li, J. Wang, Efficiently mining closed subsequences with gap constraints, in: Proceedings of the 2008 SIAM International Conference on Data Mining, SIAM, 2008, pp. 313–322, http://dx.doi.org/10.1137/1.9781611972788.28.
[25] N. Mordvanyuk, B. López, A. Bifet, vertTIRP: Robust and efficient vertical frequent time interval-related pattern mining, Expert Syst. Appl. (2020) 114276, http://dx.doi.org/10.1016/j.eswa.2020.114276.
[26] P.-V. Khuong, P. Morin, Array layouts for comparison-based searching, J. Exp. Algorithmics 22 (2017) 1–39, http://dx.doi.org/10.1145/3053370.
[27] Centers for Disease Control and Prevention, Childhood blood lead surveillance, 2017, accessed: 10-10-2020. URL https://www.kaggle.com/cdc/childhood-blood-lead-surveillance.
[28] D. García, Kaggle, 2020, accessed: 10-10-2020. URL https://www.kaggle.com/danigarci1/covid19-in-spain.
[29] R. Ilangovan, Kaggle, 2017, accessed: 10-10-2020. URL https://www.kaggle.com/rajanand/suicides-in-india.
[30] J. Bugeja, Smart home datasets and a realtime internet-connected home, 2019, accessed: 10-10-2020. URL https://hyperionsec.wordpress.com/2019/06/03/smart-home-datasets-and-a-realtime-internet-connected-home/.
[31] J.C.S. Culebras, Kaggle, 2020, accessed: 10-10-2020. URL https://www.kaggle.com/jcsantiago/covid19-by-country-with-government-response?select=covid19_by_country.csv.
[32] D. Anguita, A. Ghio, L. Oneto, X. Parra, J.L. Reyes-Ortiz, A public domain dataset for human activity recognition using smartphones, in: Esann, 2013, pp. 437–442, URL https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2013-84.pdf.
[33] D. Dua, C. Graff, UCI machine learning repository, 2017, accessed: 10-10-2020. URL https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.
[34] S. Koelstra, C. Muhl, M. Soleymani, J. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, I. Patras, Deap: A database for emotion analysis using physiological signals, IEEE Trans. Affect. Comput. 3 (1) (2012) 18–31, http://dx.doi.org/10.1109/T-AFFC.2011.15.
[35] S. Koelstra, C. Muhl, M. Soleymani, J. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, I. Patras, DEAP dataset, 2012, accessed: 12-02-2022. URL https://www.eecs.qmul.ac.uk/mmv/datasets/deap/index.html.
[36] N. Mordvanyuk, J. Gauchola, B. López, Understanding affective behaviour from physiological signals: Feature learning versus pattern mining, in: 34th IEEE CBMS International Symposium on Computer-Based Medical Systems, 2021, pp. 438–443, http://dx.doi.org/10.1109/CBMS52027.2021.00049.