ClaSP - Time Series Segmentation

Patrick Schäfer[*]
Humboldt-Universität zu Berlin
patrick.schaefer@hu-berlin.de

Arik Ermshaus[*]
Humboldt-Universität zu Berlin
ermshaua@informatik.hu-berlin.de

Ulf Leser
Humboldt-Universität zu Berlin
leser@informatik.hu-berlin.de

ABSTRACT

The study of biological or physical processes often results in long sequences of temporally-ordered values, aka time series (TS). Changes in the observed processes, e.g. as a cause of natural events or internal state changes, result in changes of the measured values. Time series segmentation (TSS) tries to find such changes in TS to deduce changes in the underlying process. TSS is typically approached as an unsupervised learning problem aiming at the identification of segments distinguishable by some statistical property. We present ClaSP, a novel and highly accurate method for TSS. ClaSP hierarchically splits a TS into two parts, where each split point is determined by training a binary TS classifier for each possible split point and selecting the one with highest accuracy, i.e., the one that is best at identifying subsequences to be from either of the partitions. In our experimental evaluation using a benchmark of 98 datasets, we show that ClaSP outperforms the state-of-the-art in terms of accuracy and is also faster than the second best method. We highlight properties of ClaSP using several real-life time series.

CCS CONCEPTS

• **Computing methodologies → Unsupervised learning**; • **Mathematics of computing → Time series analysis**.

KEYWORDS

Unsupervised, Self-Supervised, Segmentation, Change Points

ACM Reference Format:
Patrick Schäfer, Arik Ermshaus, and Ulf Leser. 2021. ClaSP - Time Series Segmentation. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482240

1 INTRODUCTION

Recent years brought an explosion in applications for low-cost high resolution sensors, for instance in mobile devices, systems and manufacturing monitoring, or environmental and medical surveillance. These sensors produce large amounts of unlabelled temporally-ordered, real-valued sequences, also referred to as data series or *time series (TS)*. This leads to increasing interest in TS analytics, such as classification [4, 16], similarity search [28], or motif discovery [25]. An important field is unsupervised TS analysis, often used to gain an initial understanding of the data at hand. Research problems include motif discovery [25], discord discovery [22], and anomaly/outlier detection [5]. A particularly interesting problem which we study in this paper is time series segmentation (TSS). TSS aims at discovering regions of a TS that are (semantically) dissimilar to neighboring regions. TSS is an important technology, as it allows to infer properties of the underlying system by analysis of the measurements, because shifts from one segment to another are often caused by state changes in the process being monitored, such as a transition from one operational state to another or the onset of an unusual event. Change point detection (CPD) [30] is the task of finding such shifts in the underlying signal, and a segmentation is an ordered sequence of change points. A variety of CPD and TSS algorithms have been proposed, see [3] for a survey. Many examples of domain specific solutions exist for medical condition monitoring [6], climate change detection [15], or human activity analysis [7]. In contrast, only few domain-agnostic TSS algorithms exist, including FLOSS [17], Autoplait [24], and HOG-1D [35]. FLOSS is the-state-of-the-art for TSS and CPD. It annotates a TS with a bespoke *arc curve*, which spans from each offset in the TS to its 1-NN subsequence. The offsets with the least crossings of arc curves indicate the potential change points.

We present ClaSP (Classification Score Profile), a new and highly accurate algorithm for CPD and TSS. ClaSP approaches segmentation by adapting a TS classifier to identify regions of similar shape. The underlying assumption is that two adjacent segments of a TS are separated by a change point, separating the TS into two parts that are *self-similar* but dissimilar from the other. By first finding the strongest change point, additional change points increasingly refine the segmentation of the TS. The central idea of ClaSP is to iteratively determine change points by finding the point in the TS where the performance of a binary classifier, trained to separate subsequences either to belong to the left or the right part, is highest. This allows to use established methods from supervised TS analysis for solving an unsupervised TS problem. A potential drawback of this approach is the runtime necessary for training and evaluating a flood of classifiers for all hypothetical splits. However, we show that ClaSP, by relying on a k-NN classifier, achieves both highly accurate CPD and TSS results and is very fast, because most of the work can be factored out into a split point-independent preprocessing phase that is based on a fast pairwise distance matrix computation for overlapping subsequences, for which efficient algorithms exist [36]. Using this pairwise distance matrix, ClaSP then creates a Classification Score Profile (ClaSP) from which change points can easily be identified using a bespoke peak detection algorithm.

Figure 1 illustrates such a ClaSP together with the underlying TS. The dataset originates from [26] and was preprocessed by [17]. It shows an EEG of a human brain with tonic-clonic seizure activity

[*]The first two authors contributed equally.

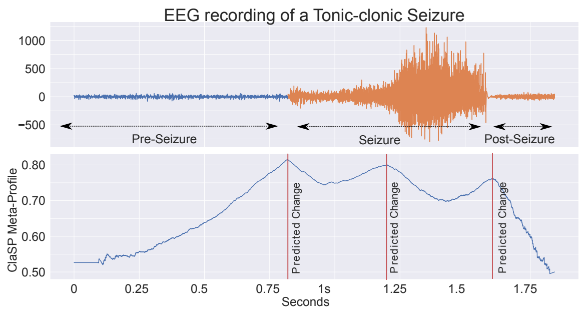


Figure 1: A scalp right central electrode recording of an EEG from a subject showing a tonic-clonic seizure [26]. It contains three segments that capture pre-seizure, seizure and post seizure EEG activity. The three local maxima in *ClaSP* (bottom) highlight potential change points based on changes in the underlying shapes of the divided segments, representing the three activities.

after about 1 minute of recordings. The three local maxima in ClaSP (bottom) mark the change points between dissimilar segments. The first segment captures the pre-seizure, the two following the seizure (made of two distinct phases), and the last segment corresponds to post-seizure activities. The global maximum is between the first and second segment (first red vertical line) and matches the start of the seizure. The specific contributions of this paper are:

(1) We present ClaSP, a novel method for CPD and TSS based on annotating a TS with a bespoke classification score profile using concepts of self-supervision [31].
(2) We provide an efficient implementation of ClaSP which achieves a runtime as fast as its competitor methods.
(3) ClaSP has just a single hyper-parameter, which is the length of subsequences used as features for the self-supervised classification. We describe methods to automatically determine this hyper-parameter for a given TS.
(4) We performed a series of experiments on CPD and TSS using a benchmark of 98 datasets [17] and compare results to five state-of-the-art competitors, namely Autoplait, FLOSS, BinSeg-L_2, BOCD, and one simple baseline, namely Window-L_2 [30]. ClaSP exhibits the lowest overall segmentation error and achieves best segmentations in 80 out of the 98 cases. Its runtime is lower than that of its 2nd best competitor FLOSS.
(5) An additional advantage of ClaSP is that the classification score profile allows for intuitive visualizations, making the segmentation decisions easily interpretable for humans.

The remainder of this paper is organized as follows: Section 2 presents background and definitions. Section 3 introduces related work. Section 4 presents the ClaSP method. Section 5 presents our experimental evaluation and Section 6 concludes the paper.

2 BACKGROUND AND DEFINITIONS

In this section, we formally introduce the concepts of time series (TS), subsequences, windows, and the change point detection (CPD) and segmentation (TSS) problem.

Definition 1. *A* time series *T is a sequence of $n \in \mathbb{N}$ real values, $T = (t_1, \ldots, t_n), t_i \in \mathbb{R}$. The values are also called data points.*

Definition 2. *A TS is* stationary *if any of its values are independent of the time they were observed.*

Intuitively speaking, a stationary TS exhibits constant behavior, leading to time-independent mean, variance, and auto-correlation. There exists no trend or seasonality.

Definition 3. *Given a TS T, a* subsequence *$T_{s,e}$ of T with start offset s and end offset e consists of the contiguous values of T from position e to position s, i.e., $T_{s,e} = (t_s, \ldots, t_e)$ with $1 \leq s \leq e \leq n$. The length of $T_{s,e}$ is $|T_{s,e}| = e - s + 1$.*

We sometimes call subsequences *windows* and their length *width*.

Definition 4. *A* segmentation *of a TS T into $C + 1$ segments is an ordered sequence of change points (or splits) $t_{i_1,\ldots,t_{i_C}}$ with $1 < i_1 < \cdots < i_C < n$.*

Definition 5. *The problem of time series segmentation (TSS) is to find a meaningful segmentation of a given TS T under the assumption that T was generated by a process with discrete states. A segmentation is considered meaningful when the change points between two subsequent segments correspond to state changes in the underlying process.*

Our definition follows [17] by making the underlying assumption that a physical process has discrete states which lead to changes in the measured values. This is opposed to problems such as trend detection [30]. ClaSP, the method for TSS we propose in this paper, is based on *time series classification* (TSC), though it solves the unsupervised TSS problem. TSC is the task of predicting a class label for a TS from a predefined set of labels [4]. A TS classifier is a function learned from a set of labeled TS (the training data). It takes an unlabeled TS as input, and outputs a label. We apply TSC to TSS based on the idea of *self-supervision* [31]. Self-supervised learning is a form of unsupervised learning in which the data generates the supervision (labels). In representation learning [23], for example, self-supervision refers to learning representations of the data without annotated labels, purely based on the data.

3 RELATED WORK

Truong et al. [30] present a review of selected domain-specific methods for CPD and TSS. They compare methods regarding their cost function, search method, and additional constrains, e.g., whether the number of change points is given beforehand. They discern three main classes: (a) Likelihood-based methods, (b) Kernel-based methods, and (c) Graph-based methods. Likelihood-based methods split TS into consecutive windows and compare the probability distributions of windows [21]. If these differ significantly, a change point is introduced. Kernel-based methods also split the TS into windows and then use a kernel-based statistical test to assess the homogeneity between subsequent windows [18]. Graph-based methods first infer a graph by mapping observations (i.e. windows or sets of TS) to nodes and connecting nodes by edges if their pairwise similarity exceeds a predefined threshold. Next, a bespoke graph statistic is applied to split the graph into sub-graphs leading to change points in the TS [8]. So-far domain-agnostic segmentation solutions have seen less popularity than domain-specific methods. The three most prominent of these algorithms are FLOSS [17], Autoplait [24], and HOG-1D [35]. FLOSS [17] uses the proximity of a window to the most similar other window to create a bespoke arc curve, which is a vector that contains for each index i the number of *arcs* that cross over i. Local minima of this number indicate boundaries (change points) of self-similar regions. FLOSS also published a TSS/CPD benchmark, which we use in our experiments (see Section 5). Autoplait [24] determines segments of a TS using the Minimum Description Length (MDL) principle to recursively test if a region should best be modelled by a single Hidden Markov Model (HMM) or two HMMs. A recent survey compared CP methods [32]. The best methods in their evaluation were Binary Segmentation (BinSeg) [29] and Bayesian Online Changepoint Detection (BOCD) [1]. We compare ClaSP to these 5 competitors regarding accuracy and runtimes in Section 5. In contrast to previous approaches, ClaSP uses self-supervision to apply classification techniques to the problem of TSS. A similar technique is used in [20], but in this case applied for detecting differences between two data sets. We use this rationale for comparing potential binary splits of a TS. Note that our idea is very different from supervised TSS, where a model is trained on TS with annotated segments [10]; in unsupervised TSS, no such annotation is required or given.

4 CLASP - CLASSIFICATION SCORE PROFILE

Classification Score Profile (ClaSP) is a novel method for TS segmentation based on self-supervision. Segments may vary in length and are divided by change points (CP) – short: splits. We first give an intuitive overview of our method, before we explain its individual steps in detail in Sections 4.1 to 4.5, using CPD as running motivation. We then describe the extensions necessary for approaching the general segmentation problem in Section 4.6. The workflow is shown graphically in Figure 2. Pseudo code is shown in Algorithm 1.

Given a TS T with length $|T| = n$, ClaSP first computes a classification score profile. To this end, we first partition T into overlapping windows of a fixed length w (Algorithm 1, line 4). Next, hypothetical splits are generated for increasing offsets $i \in [w+1, \ldots, n-w-1]$, and certain characteristics are extracted for each window as features for the later used classifier (line 5). We transform each such split into a binary classification problem $Y = \{0, 1\}$ by attaching label 0 (1) to all windows to the left (right) of the split point. A binary k-Nearest-Neighbour classifier (k-NN) is trained on these features and evaluated in a cross-evaluation setting (lines 6-9). We interpret the cross-validation score of the classifier as a measure of how dissimilar windows from the left are to windows from the right, where a high score means low similarity between segments. This degree of *intra-segment similarity* (self-similarity) is recorded for each offset i (line 8), together forming the classification score profile for T. Every local maximum in this profile represents a potential change point, as it is a point where the distinction between the TS part to the left and that to the right is the highest (compare Figure 1). ClaSP has a single hyper-parameter, the window-length w used to generate labelled windows for training and evaluation.

Definition 6. *Given a TS T and a window-length w, a ClaSP is a real-valued sequence S of length n. The i-th value in S is the cross-validation score $s \in [0, 1]$ of a classifier C trained on a binary classification problem with labels $Y = \{0, 1\}$. For index $i \in [w+1, n-w-1]$ training samples are created by assigning label $y = 0$ to all windows to the left $W_L = \bigcup_{j\in[1,\ldots,i-w]} T_{j,j+w}$ and $y = 1$ to all windows to the right $W_R = \bigcup_{j\in[i-w+1,\ldots,n-w+1]} T_{j,j+w}$. Values $S[1, \ldots, w]$ and $S[n-w-1, \ldots, n]$ are set to 0, giving a very small blind spot.*

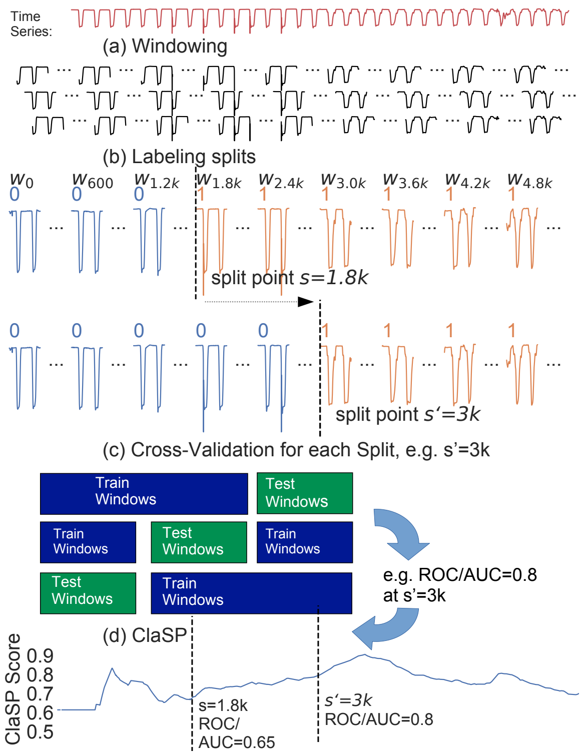


Figure 2: (a) A TS (in red) is split into overlapping windows (in black). (b) These windows are labelled with 0 for windows to the left and 1 to the right of hypothetical split points, exemplary depicted for indexes 1800 and 3000. (c) For each hypothetical split, a binary classifier is trained and evaluated using cross-validation. (d) Finally, the ClaSP is created from the cross-validation score of the classifiers.

In the following subsections we elaborate on (4.1) the choice of using a k-NN classifier to achieve high classification speed, (4.2) the pre-computation phase of ClaSP, (4.3) the computation of cross-validations for hypothetical split points, (4.4) the choice of a scoring metric for a classifier, and (4.5) the problem of selecting the proper values for hyper-parameter w. Section 4.6 describes the application of ClaSP for TSS, and Section 4.7 studies the computational complexity of our method.

4.1 k-NN classifier in ClaSP

ClaSP is computed using the k-nearest-neighbour (NN) classifier, which has some nice properties that can be exploited to reduce computations when training and evaluating models again and again for the different split points. Recall that a k-NN classifier classifies a given sample by finding the k nearest samples in the training data

Algorithm 1 Classification Score Profile

```
1:  procedure CALC_CLASP(T, CLF, $w$)
2:      ClaSP $\leftarrow$ initialize array of length $|T|$ with 0
3:      $Y_t \leftarrow$ initialize array of length $|T|$ with 1          ▷ all in $W_R$
4:      $W = \bigcup_{j \in [1,\ldots,n-w+1]} T_{j,j+w}$          ▷ all windows from $T$
5:      $kNNs$ = KNN_PROFILE(CLF, $W$)
6:      for $i \in [w+1, \ldots, |T| - w - 1)$ do          ▷ For each split
7:          $Y_t[i-w] \leftarrow 0$          ▷ add current window to $W_L$
8:          ClaSP$[i] \leftarrow$ CROSS_VALIDATE(CLF, $kNNs$, $Y_t$)
9:      end for
10:     return ClaSP
11: end procedure
```

using a predefined distance function. It then determines the predicted label by aggregating the labels of the k-NN training samples.

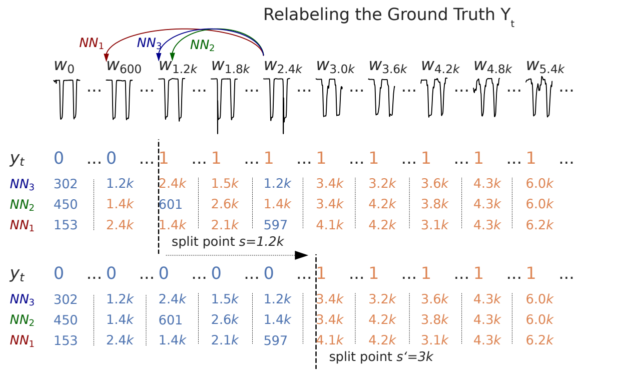
The important property of this procedure is that the pairwise distances between windows and thus the k-NNs for any window are *independent of the ground truth labels of the windows*. This means that we can actually precompute the k-NNs of any window once (Algorithm 2). The training and evaluation at a split point i then boils down to looking up the *labels* of these k neighbors in the train split of the cross-validation for every window in the test split, and aggregating them into a majority label. Thus, the expensive distance calculations have to be performed only once, and relabelling is performed on demand for every split point.

Figure 3 illustrates this idea for $k = 3$. First, 3-NNs are computed for each window. Note that an NN is represented by the offset of a window, i.e. 302 for $T_{302,302+w}$. At the left of the split (e.g. 1.2k in the centre and 3k in the bottom), we assign class label 0 (blue), and label 1 (orange) to the right of the split. For the split at 1.2k (centre) there are five windows, two to the left of the split and three to the right that have their 3-NN windows outside of their own segment, indicated by the orange and blue color. This leads to a classification error, as the classifier predicts the label of the windows in the wrong segment. When moving the split from 1.2k to 3k the ground truth Y_t changes for some windows (but not their distances), indicated by the change in colors. At the same time, the labels of the 3-NN of those windows, which were previously mislabeled, change in the left and right segment. Thus, for the bottom split, all 3-NN point to windows within the same segments, i.e. class label, resulting in a high classifier accuracy.

KNN_PROFILE(CLF, W) (line 5 of Algorithm 1) computes the k-NNs and assigns these to F. Once we have pre-computed these k-NN offsets for each window, we only need to change the ground truth labels of the NN windows (line 7) to be able to compute a new cross-validation score. We only have to change one label, equal to flipping one bit, when moving the split point from i to $i+1$.

4.2 Precompute k-NN-Profile

We discuss the details of KNN_PROFILE() (Algorithm 1 line 5) which pre-computes the k-NNs for every window from T using the z-normalized Euclidean distance on the raw values. Algorithm 2 takes the set W of windows and the k-NN classifier CLF as input, and returns an array of the offsets of the k-NNs for each window. First, it computes the pairwise window distance matrix (line 5). It then



Relabeling the Ground Truth Y_t
NN_1
NN_3
NN_2
w_0 … w_{600} … $w_{1.2k}$ … $w_{1.8k}$ … $w_{2.4k}$ … $w_{3.0k}$ … $w_{3.6k}$ … $w_{4.2k}$ … $w_{4.8k}$ … $w_{5.4k}$ …
y_t 0 … 0 … 1 … 1 … 1 … 1 … 1 … 1 … 1 … 1 …
NN_3 302 1.2k 2.4k 1.5k 1.2k 3.4k 3.2k 3.6k 4.3k 6.0k
NN_2 450 1.4k 601 2.6k 1.4k 3.4k 4.2k 3.8k 4.3k 6.0k
NN_1 153 2.4k 1.4k 2.1k 597 4.1k 4.2k 3.1k 4.3k 6.2k
split point s=1.2k
y_t 0 … 0 … 0 … 0 … 0 … 1 … 1 … 1 … 1 … 1 …
NN_3 302 1.2k 2.4k 1.5k 1.2k 3.4k 3.2k 3.6k 4.3k 6.0k
NN_2 450 1.4k 601 2.6k 1.4k 3.4k 4.2k 3.8k 4.3k 6.0k
NN_1 153 2.4k 1.4k 2.1k 597 4.1k 4.2k 3.1k 4.3k 6.2k
split point s'=3k

Figure 3: For each window the 3-NNs are computed. When iterating different splits s and s' the ground truth labels Y_t are modified (centre and bottom). The 3-NNs for each window reference new labels, leading to altered predictions for different splits - illustrated in blue and orange for the NNs.

Algorithm 2 k-Nearest-Neighbour Profile

```
1:  procedure FEATURE_EXTRACTION(CLF, W)
2:      $k \leftarrow$ CLF.$k$          ▷ k-nn hyper-parameter
3:      $kNNs \leftarrow$ array of shape$|W| \times k$
4:              ▷ Computes distance matrix of shape $|W| \times |W|$
5:      $D \leftarrow$ CLF.COMPUTE_DISTANCE_MATRIX($W$)
6:      $D \leftarrow$ APPLY_EXCLUSION_ZONE($D$)
7:      for $i \in [1, \ldots, |W|]$ do
8:          $A \leftarrow$ ARGSORT($D[i]$)          ▷ sort row by distance
9:          $kNNs[i] \leftarrow$ first $k$ offsets from $A$
10:     end for
11:     return KNNs
12: end procedure
```

selects the offsets of those k windows with the lowest distances (lines 7–10), which are returned. We make use of an exclusion zone around each window of length w, so that all windows overlapping with more than $w/2$ points are not considered during the search for NNs (line 6). This effectively sets the distances of windows within $\pm w/2$ of the diagonal of the distance matrix to infinity. The distance matrix can be computed using the dot-product. We use a fast implementation (line 5), outlined in [14, 36], requiring only $O(n^2)$-time (see Section 5.6). SCRIMP [36] implements this idea for computing pairwise z-normalized distances, but only for the overall 1-NN (and not the k-NN as used in ClaSP). Using advanced feature extraction techniques like ROCKET [11] is part of our future work.

4.3 Cross Validation

Line 8 in Algorithm 1 performs a leave-one-out cross-validation for a given offset i using the pre-computed k-NN offsets. Algorithm 3 illustrates the computation of a single score in ClaSP given the self-supervised ground truth labels Y_t and all k-NN offsets. It collects the k-NN offsets for each window (line 4), performs a lookup for their current class labels, and finally picks the majority label (line 5). The set of ground truth labels and the set of predicted labels are

Algorithm 3 Leave-One-Out Cross-Validation Score

```
1: procedure CROSS_VALIDATE(clf, kNNs, $Y_t$)
2:     $Y_{pred} \leftarrow$ array of length $|$kNNs$|$, initialized to zero
3:     for $i \in [1, \ldots, |Y_{pred}|]$ do                ▷ Iterate all windows
4:         offsets $\leftarrow$ kNNs$[i]$                    ▷ k-NN offsets
5:         $Y_{pred}[i] \leftarrow$ MAJORITY_LABEL $\left(\bigcup_{j \in \text{OFFSETS}} Y_t[j]\right)$
6:     end for
7:     return SCORING_FUNCTION$(Y_t, Y_{pred})$
8: end procedure
```

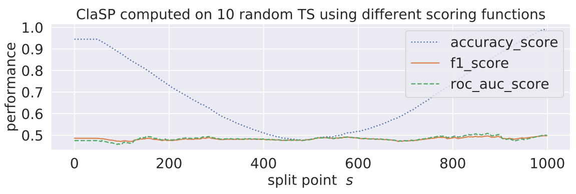


Figure 4: The averaged ClaSP computed from 10 randomly generated TS, i.e. without self-similar regions. The ClaSP using the optimal score should result in a straight line to avoid any bias, which is the case for F1 and ROC/AUC.

next passed to a scoring function (line 7). When a split moves, only Y_t changes, potentially resulting in a new score.

4.4 Classification Score Selection

ClaSP determines change points by finding local maxima in the classification score profile. To identify the most disruptive event, as required for solving CPD, we need to identify the global maximum; for solving the problem of TSS, we need to find the $C - 1$ highest local maxima, when C is the desired number of segments (see Section 4.6). However, special care has to be taken to make scores of different splits comparable, because for every potential split point in a TS the number of windows in the left and right segments differs. Accordingly, the binary classification problems are often highly class-imbalanced. This class-imbalance changes over the TS, with more severe imbalance towards the ends of the TS and low imbalance at the centre. As class imbalance can influence the performance of classifiers, a bias may emerge making the comparison of scores of different splits meaningless[1]. The impact of this bias is directly related to the particular evaluation metric being used. To illustrate this, we built a macro-averaged ClaSP from the profiles of 10 randomly generated TS and plotted the results of different evaluation metrics. These random TS should not contain self-similar segments, thus the optimal evaluation function should result in a constant line. Figure 4 shows results for accuracy, F1 and ROC/AUC. Accuracy is highly sensitive to class-imbalances and thus leads to a profile similar to a parabola, with high values on the left and right corners. F1 and ROC/AUC both perform much better as shown by the flat line even at the corners of ClaSP. This leads to the conclusion that macro F1 and macro ROC/AUC are both suitable.

[1]Note that FLOSS also has this issue which it solves using score normalization

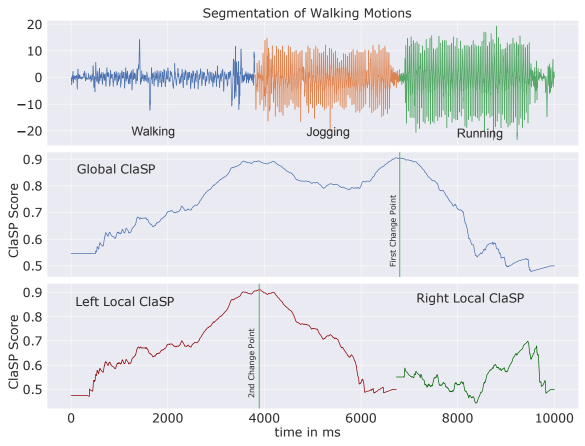


Figure 5: ClaSP is computed, and its global maximum defines the first change point. For each resulting disjoint segment a (local) ClaSP is computed, and its global maximum is chosen. This process is recursively repeated until the required number of change points (i.e. segments) is reached.

4.5 Window Size Selection

ClaSP takes the window size w as its only hyper-parameter. This parameter has data-dependent effects on ClaSP's performance. When chosen too small, all windows tend to appear similar; when chosen too large, windows have a higher chance to overlap adjacent segments, blurring their discriminative power. It is thus non-trivial to pick a suitable window size for a given TS. The ideal window size should correlate with the inherent statistical properties of the given dataset. In the literature, it is common to have the window size being set manually by a domain expert [17]. This implicitly allows for the usage of background knowledge, but is costly and slow, as such experts first have to be identified. An alternative is to perform window size selection automatically. In the experiments section, we compare manually defined window sizes with three automatic methods: **Ensembling ClaSPs:** The first approach is to use different window sizes, leading to multiple ClaSP scores per offset. The final ClaSP score for each offset is computed as the average over its ClaSP scores. **Highest maxima:** The second approach also calculates sets of ClaSP scores using different window sizes. It then chooses a change point as the offset with the overall highest value, i.e., the global maximum over all scores. **Dominant Frequency:** The third approach first transforms a TS using the Discrete Fourier Transform. It then selects the dominant Fourier coefficients with the highest magnitude. The corresponding frequency can be transformed to a window size.

4.6 Segmentation

Using ClaSP for the CPD problem is straightforward: We first compute the profile and then choose its global maximum as the change point. Application of the idea to TSS is more complicated. Let us assume one would want to segment a TS T into C segments. A naive idea is to run the algorithm once and then return the $C - 1$ highest values as change points. However, these are typically no

Algorithm 4 Segmentation

```
1:  procedure SEGMENTATION($T$, $n\_cpts$)
2:      $cpts \leftarrow$ INITIALIZE List
3:      $pq \leftarrow$ INITIALIZE Max-Priority-Queue
4:      $(cpt\_idx, cpt\_val) = \leftarrow$ CALC_CPT($T$)
5:      PQ.INSERT($cpt\_val$, ($cpt\_idx$, $T$))
6:      for $i \in [1, \ldots, \text{n\_cps}]$ do                ▷ desired number of cp
7:          $(cpt\_idx, S) \leftarrow$ PQ.REMOVEMAX()
8:          $cpts$.APPEND($cpt\_idx$)                  ▷ keep global maximum
9:          $T_L = S_{begin:cpt\_idx}$                     ▷ left+right segment
10:         $T_R = S_{cpt\_idx:end}$
11:         $(\text{idx}_L, \text{val}_L) = \leftarrow$ CALC_CPT($T_L$)
12:         PQ.INSERT($\text{val}_L$, ($idx_L$, $T_L$))
13:         $(\text{idx}_R, \text{val}_R) = \leftarrow$ CALC_CPT($T_R$)
14:         PQ.INSERT($\text{val}_R$, ($idx_R$, $T_R$))
15:     end for
16:     return $cpts$
17: end procedure
18: procedure CALC_CPT(T)                    ▷ index+value of change-pt
19:     $profile \leftarrow$ CALC_CLASP($T$)
20:     return (ARGMAX($profile$), MAX($profile$))
21: end procedure
```

local optima, but instead all very close to the highest value (global maximum). Another idea is to use a peak finding algorithm, such as those implemented in *scipy.signal.find_peaks*, and then return the $C-1$ highest peaks. However, this again often leads to clusters of peaks around the highest point; furthermore, it requires to determine parameters of the peak finder, such as minimal gap between peaks or minimal elevation over the local neighborhood.

Instead, we propose and evaluate a different *parameter-free* strategy. Our idea is to apply a recursive splitting algorithm. Given T, the algorithm first computes ClaSP and selects the maximal peak as the first change point. Next, it computes two new ClaSPs, one for the left and one for the right segment of the first split. Within these profiles, it picks the larger peak of the two as a second change point. It then computes ClaSP for the resulting three segments, computes three maxima, chooses the highest, etc. This process is recursively repeated until the desired number of change points is derived. Note that in every iteration, only two ClaSPs have to be computed, as all but one segment remain unchanged compared to the last iteration.

Pseudocode is shown in Algorithm 4, which takes as input a TS T and the desired number of change points $C = n\_cpts$. It first calculates the ClaSP over the entire T (line 4) and stores the index of the peak with the highest score in a priority queue (line 5). Within a loop (line 6) the offset of the largest local maximum cpt_idx is extracted and the two left and right segments of the split cpt_idx are derived (lines 9-10), for which the corresponding ClaSPs are computed (lines 11,13), and added to the priority queue (lines 12,14). The loop ends when the desired number of CP has been extracted.

An example is shown in Figure 5. The TS (top) shows the rotation of a subject's left calf while first walking, then jogging, and finally running (taken from [17]). Each type of activity represents one rather homogeneous segment, whereas different activities exhibit different frequencies of rotational deflections. The first profile (Figure 5, centre) already contains clear peaks for both CPs, but the second peak becomes more precise through the second iteration. The main advantage of this iterative process is that it is parameter-free and does not require to set an exclusion zone around peaks.

4.7 Computational Complexity

The computational complexity of ClaSP (Algorithm 1) using k-NN classifiers is dominated by the cost of KNN_PROFILE() and $O(n)$ calls to CROSS_VALIDATION(). KNN_PROFILE() is dominated by the time needed to compute the distance matrix (line 5). The distance matrix can be computed in $O(n^2)$ by a reformulation using the dot-product [14, 36], for TS of length n. Retrieving the k smallest window offsets (line 8) can be solved in $O(k \cdot n)$ using k sequential searches or $O(n + k \log n)$, when using a min-heap. Thus, a total of $O(n^2 + n \cdot k \log n)$ for lines 7–9. CROSS_VALIDATION can be performed in $O(n)$ for a single split index, with a total of $O(n)$ splits. Thus, the total complexity is in $O(n^2)$.

The time complexity to find a single change point using ClaSP is thus in $O(n^2 + n \cdot k \log n)$. SEGMENTATION() into $c = n\_cpts + 1$ different segments involves $O(c)$ calls to ClaSP. This has an overall WORST CASE complexity of $O(c \cdot (n^2 + n \cdot k \log n))$. In the BEST CASE however we always halve the segments after each split, therefore result in a complexity of $O(n^2 + 2(n/2)^2 + 2(n/4)^2 + \ldots) = O(n^2)$.

5 EXPERIMENTAL EVALUATION

We compare the accuracy and runtime of ClaSP with four state-of-the-art competitors and a simple baseline using a publicly available dataset of 98 TS annotated with change points. We also perform experiments to study the influence of the only hyper-parameter of ClaSP, the window size (see Section 5.3), and the influence of evaluation metrics and segment numbers (see Section 5.1). Finally, we discuss three challenging real-life data sets. To ensure reproducible results and to foster follow-up works, we provide the ClaSP source code, Jupyter-Notebooks, visualizations of the datasets, and the raw measurement sheets on our website [9].

5.1 Benchmark Setup

Datasets. Overall we use 98 benchmark datasets to asses the performance of ClaSP and to compare it to rivalling methods, which is the largest collection of datasets considered so far for change point detection. 32 datasets stem from public segmentation benchmark datasets [17] that capture biological, mechanical or synthetic processes. Change points as well as period sizes were annotated by human experts as described in [17]. Furthermore, these dataset-dependent period sizes were used in all experiments as window sizes for all competitors that set this parameter.

We created a semi-synthetic dataset from the UCR archive [34]. From the about 120 datasets (DS) we first removed all DS with missing values, too many classes, too little TS per class label, or too large DS. We finally visually inspected the remaining DS and selected a subset of 66. All of the remaining DS have in common that they show some obvious periodicity (compare our assumptions in Section 2). Each UCR dataset contains multiple labelled TS. We group TS by class label and concatenate all TS to create segments with repeating temporal patterns and characteristics. The location at which different classes were concatenated are marked as change

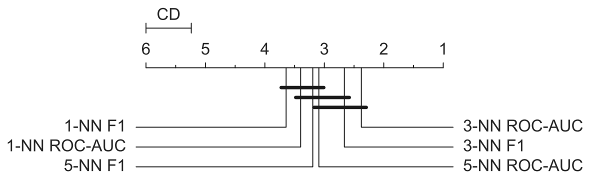


Figure 6: Influence of different design choices on average rank for 98 benchmark datasets.

points. We resample the resulting TS to control the TS resolution. The window sizes for these datasets are hand-selected to capture temporal patterns but are approximate and limited to the values [10,20,50,100] to avoid over-fitting. We have uploaded a Jupyter-Notebook to generate the benchmark to our website [9]. Out of all 98 datasets: 49 TS have 2 segments (1 CP), 22 datasets have 3 segments, 10 datasets have 4 segments, 11 datasets have 5 segments, 1 dataset has 6 segments, and 5 datasets have 7 segments. We used the pre-defined window sizes for our comparison results, but also study the performance of our suggested methods for determining this parameter automatically (see Section 5.3).

Competitors. We publish the results on these benchmark datasets for Autoplait [24], FLOSS [17], and the best performing two methods from [32], namely Binary Segmentation (BinSeg) [29] and Bayesian Online Changepoint Detection (BOCD) [1]. Additionally, we evaluated the dynamic programming, bottom-up segmentation, as well as the window-based change point algorithm from [30] with L1, L2, auto-regressive, kernel, and Gaussian cost functions using default parameters. In our comparison, we include as a simple baseline the results of the best-performing one of these, the window-based algorithm with L2 cost function (Window-L_2).

Evaluation Metric. We use the metric as defined in [17] for measuring the quality of a method: Given a TS T and sets of predicted CPs $cpts_{pred}$ and of ground truth CPs $cpts_T$, with each location in $[1 \dots n]$, we compute the normed $error \in [0 \dots 1]$ as:

$$error = \frac{1}{n \cdot |cpts_{pred}|} \cdot \sum_{p \in cpts_{pred}} \min_{p' \in cpts_T} |p - p'|$$

This measure sums up and normalizes the relative distances between every predicted change point $p \in cpts_{pred}$ and the closest ground truth change point. Note that this measure does not perform a bipartite matching, as multiple predicted CP may be matched to the same ground truth CP. This can be seen as a disadvantage, but we stick to this definition to allow comparisons with previously published results. We also use critical difference diagrams (as introduced in [12]) to compare ranks between approaches. The best approaches scoring the lowest (average) ranks are shown to the right of the diagram. Groups of approaches that are not significantly different in their ranks are connected by a bar, based on a Nemenyi two tailed significance test with $\alpha = 0.05$.

ClaSP Parameter Settings. There are three main design choices for ClaSP: (a) the evaluation metric and its treatment of class imbalance, (b) the number of k neighbours used for classification, and (c) the window size w. We performed ablation studies to fix values for parameters (a) and (b). We tested all combinations of F1 versus ROC/AUC and neighbours $k \in [1, 3, 5]$. Figure 6 shows the results. Overall, we found no significant differences between these settings. Because 3-NN in combination with ROC/AUC has the lowest average rank, we use this configuration in all subsequent experiments. Regarding (c) we primarily used the expert defined values, but also performed experiments with automatic selection strategies (see Section 5.3).

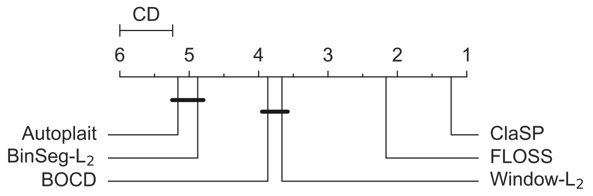


Figure 7: Segmentation ranks on 98 benchmark datasets for ClaSP (lowest rank) and the 5 state-of-the art competitors.

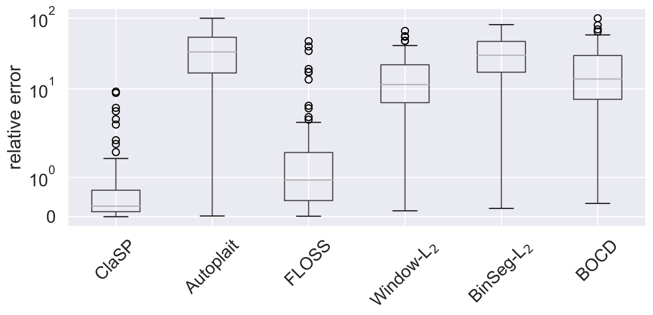


Figure 8: Boxplot on segmentation error on 98 benchmark datasets for ClaSP and the state-of-the art competitors.

Table 1: Summary wins/ties/losses of ClaSP over rivals.

	Autoplait	FLOSS	Window-L_2	BinSeg-L_2	BOCD
ClaSP	93/0/5	88/1/9	94/0/4	96/0/2	96/0/2

5.2 Segmentation Errors on Benchmark Set

The critical difference diagram in Figure 7 shows the average ranks for ClaSP, Autoplait, Window-L_2, BinSeg-L_2, BOCD, and FLOSS based on their errors for the segmentation task on each of the 98 benchmark TS. Note that Autoplait fails to return any result on 68 datasets (compare [17]). Overall, ClaSP shows by far the lowest rank (average 1.22), followed by FLOSS (average 2.16). ClaSP is significantly better than all other competitors. On the entire data set, ClaSP has 80 wins or ties (first position in error), followed by FLOSS (10), Autoplait (4) Window-L_2 (3), BOCD (2) and BinSeg-L_2 (0) (counts do not sum up to 98 due to ties). When looking at the results for the two classes of tasks in this benchmark individually, namely strong CPD (one change point, 49 instances) and TSS (more than two change points, 49 instances), the situation remains essentially the same (data not shown): ClaSP has the overall lowest rank.

ClaSP also scores the lowest median error and standard deviation over all TS (Figure 8). It has a 2.24 percentage points (pp) smaller

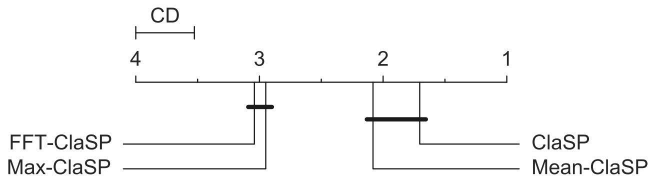


Figure 9: Results of different window size selection strategies on the benchmark datasets.

average error and 6 pp smaller standard deviation compared to its second best competitor FLOSS. On average, both algorithms score much less errors than the other methods, with more than 15%. In a pairwise comparison of ClaSP against every competitor, ClaSP achieves between 88 (vs FLOSS) and 96 (vs BOCD) wins (Table 1). For instance, it achieves a lower error than FLOSS for 88 of the 98 datasets, the same error in 1 case, and is beaten by FLOSS in 9 cases.

5.3 Effect of Window Size Selection

Results so far were obtained with window sizes as provided by the benchmark, which were manually determined by domain experts using domain knowledge [17]. Thus an approach probably leads to favorable results, but requires the availability of such domain experts that are also familiar with TS analysis, which often poses an obstacle to data analysis. In Section 4.5, we described three strategies for alleviating the problem of manual window size selection.

We performed an experiment to study the quality of these strategies and whether they can beat domain experts. We applied the three strategies with a set of window sizes $[10, 20, 50, 100]$. *Mean-ClaSP* refers to an ensembles of ClaSPs of different window sizes, *Max-ClaSP* refers to the highest global maximum, and *FFT-ClaSP* is the dominant Fourier Frequency approach. Figure 9 shows the results. ClaSP with domain expert annotations shows the lowest average rank. It achieves 60 first ranks, whereas the Mean-ClaSP leads to only 11 wins. The median errors of the two best strategies, Mean-ClaSP and ClaSP, are close by with 0.47% and 0.9%. In contrast, the mean errors differ, with 4.3% for Mean-ClaSP and 1.2% for ClaSP, which is in parts caused by a single outlier on the WalkJogRun2 dataset, where Mean-ClaSP scored an error greater than 30%. We believe these results indicate that further research into parameter-free methods is promising, and that human annotation of window sizes might not be necessary in future applications.

5.4 Complex Classifiers in ClaSP

We have focused on a 3-NN-based approach in this paper, and presented optimizations to avoid expensive distance calculations for every split that result in a complexity of $O(n^2)$. We also performed initial experiments evaluating more complex TS classifiers with r-fold CV including BOSS [27], TS-Forest [13] and ROCKET [11] but without any accuracy gains on the given 98 benchmark TS. These TS classifiers also result in much higher runtimes of $O(r \cdot n \cdot (training + testing))$. Nonetheless, we have published the results to our supporting website [9]. Optimizing complex classifiers is part of our future work.

5.5 Runtime Comparison on Benchmark Set

Figure 10 shows runtimes of FLOSS, ClaSP, BinSeg-L_2, BOCD, and Window-L_2 on each of the 98 benchmark TS. Window-L_2 has the lowest overall runtime, followed by BinSeg-L_2. FLOSS and ClaSP both have the same computational complexity. Nevertheless, ClaSP is always faster than FLOSS, often by a factor of at least 2. The marked peak in runtime for TiltABP and TiltECG for FLOSS and ClaSP are due to their much greater length (length 40k compared to length 19k of the next-longest TS InsectEPG4) and the quadratic runtime complexity of FLOSS and ClaSP. We published plots on scalability regarding length n or number of CPs to [9].

5.6 Results on Specific Use Cases

We describe results on three TS in more detail to show the strengths and limitations of ClaSP. Two (Insect EPG and Tilt Table) are also part of the benchmark used above, while NYC Taxi is not.

Insect EPGs. Monitoring the feeding behavior of sucking insects can reveal important insights into the transmission of pathogens [33]. Figure 11 shows the recording of an electrical penetration graph (EPG), for which an insect is tethered to an electrode while feeding from a food source. Different phases in the feeding lead to changes in electric flow through the electrodes, which are recorded and transformed into a TS. The TS in the figure contains one annotated change point, indicated by the transition in colors. From top to bottom, we highlight the segmentation given by ClaSP, FLOSS and Window-L2[2]. While ClaSP detects the change point very precisely, the other two methods predict it further to the left, i.e., too early.

Very small Segments. The New York City taxi passenger TS [2] contains six months of taxi passenger volumes aggregated at 30 minute intervals, annotated with windows that point to the NYC marathon, Labor Day, Thanksgiving, Christmas, and New Year Eve. This is a challenging dataset for TSS algorithms as all these events are very short, leading to very short phases of distortion in the TS. Furthermore, the background signal is complex, with strong but somewhat repetitive changes over the course of a day. We ran ClaSP and FLOSS on this dataset with a 8 hour window size to evaluate how they perform in such a difficult setting. Figure 12 shows the classification score profile and the CPs as detected by ClaSP and FLOSS, respectively. Events that last an entire day or more, like Labour Day, are clearly visible as distinctive local peaks. On the other hand, shorter events like the NYC marathon do not lead to clear peaks. Both methods find Labor Day as their first CP. ClaSP positions its second and third CP near New Year and Christmas and finds Thanksgiving as fourth CP. FLOSS predicts its second to fourth CPs in close proximity to its first CP, probably because the default setting of the exclusion zone is not appropriate for this dataset. Together, the segmentation by ClaSP is clearly meaningful yet certainly not optimal, while FLOSS with default parameters produces a much worse result.

Gradual vs Discrete Changes. Finally, we study a use case showing the limits of TSS algorithms for cases where changes are gradual and not sudden. We analyse TS data from an experiment where the arterial blood pressure of a person lying on a tilt table with foot rest

[2]Autoplait and HOG-1D had significantly worse results than ClaSP and FLOSS.

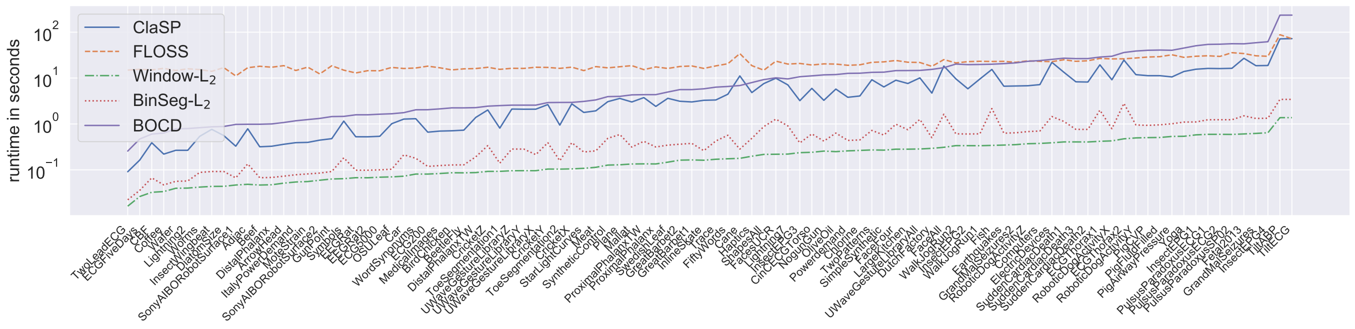


Figure 10: Runtime of ClaSP in comparison to its competitors. Data sets are ordered by size with highly irregular increases.

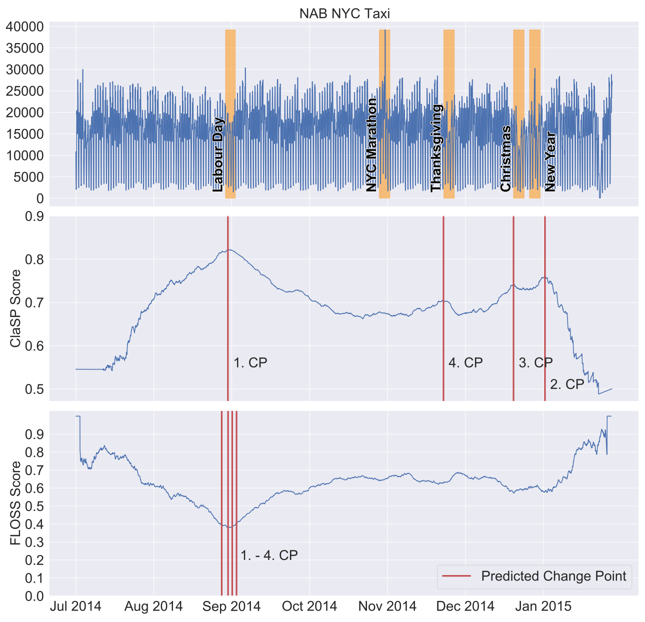


Figure 12: The New York City taxi passenger dataset shows the fluctuation of the number of passengers. Known anomalies closely resemble ClaSP's local maxima [2].

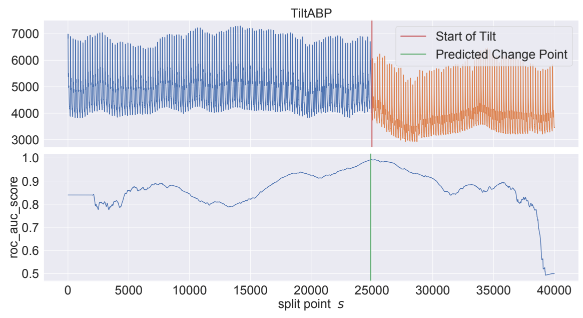


Figure 13: Segmentation of arterial blood pressure from a subject laying on a tilt table. The CP at 25s represents the start of the tilt to stand-up position.

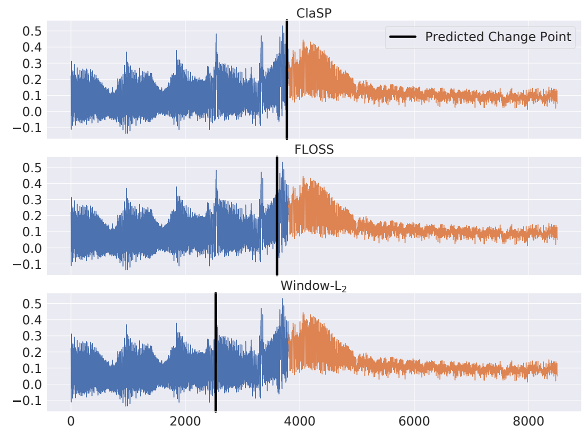


Figure 11: Segmentation of an EPG from a fluid-feeding insect. There is one annotated change point, indicated by the transition in colors at which the feeding state changes.

is measured [19]. The table is rapidly turned up, leading to a sudden rise in blood pressure. However, once reaching the upright position, the blood pressure only slowly goes back to normal. Figure 13 shows that ClaSP detects the point in time when the table started tilting, i.e., it detects the rapid change. However, it cannot detect the point where the upright position is reached and blood pressure starts to decrease, as this does not lead to sudden changes in the TS, but happens only gradually. Note that the detection of such slow changes is actually outside the scope of current TSS methods by their very definition (see Section 2).

6 CONCLUSION

We have introduced ClaSP, a new method for CPD and TSS based on the principle of self-supervision. ClaSP produces and analyses a classification score profile, which is also amendable to human inspection. Our experimental evaluation shows that ClaSP sets the new state-of-the-art on the benchmark set of [17] and is also faster than its second best competitor. Open research questions are how to extend ClaSP for more powerful classifiers and methods for the automatic detection of a suitable number of segments in a TS. One limitation of ClaSP is that it does not allow online segmentation in a streaming setting, unlike than e.g. FLOSS. We plan to investigate this in future work.

REFERENCES

[1] Ryan Prescott Adams and David JC MacKay. 2007. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742* (2007).

[2] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262 (2017), 134–147.

[3] Samaneh Aminikhanghahi and Diane J Cook. 2017. A survey of methods for time series change point detection. *KAIS* 51, 2 (2017), 339–367.

[4] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2016. The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version. *DMKD* (2016), 1–55.

[5] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. 2021. A Review on outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys (CSUR)* 54, 3 (2021), 1–33.

[6] Marcel Bosc, Fabrice Heitz, Jean-Paul Armspach, Izzie Namer, Daniel Gounot, and Lucien Rumbach. 2003. Automatic change detection in multimodal serial MRI: application to multiple sclerosis lesion evolution. *NeuroImage* 20, 2 (2003), 643–656.

[7] Sofiane Brahim-Belhouari and Amine Bermak. 2004. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis* 47, 4 (2004), 705–712.

[8] Hao Chen, Nancy Zhang, et al. 2015. Graph-based change-point detection. *The Annals of Statistics* 43, 1 (2015), 139–176.

[9] ClaSP Code and Raw Results. 2021. https://sites.google.com/view/ts-clasp/.

[10] Diane J Cook and Narayanan C Krishnan. 2015. *Activity learning: discovering, recognizing, and predicting human behavior from sensor data*. John Wiley & Sons.

[11] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1454–1495.

[12] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.

[13] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. 2013. A time series forest for classification and feature extraction. *Information Sciences* 239 (2013), 142–153.

[14] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. 2015. Euclidean distance matrices: essential theory, algorithms, and applications. *IEEE Signal Processing Magazine* 32, 6 (2015), 12–30.

[15] Jean-François Ducré-Robitaille, Lucie A Vincent, and Gilles Boulet. 2003. Comparison of techniques for detection of discontinuities in temperature series. *International Journal of Climatology* 23, 9 (2003), 1087–1101.

[16] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2018. Deep Learning for Time Series Classification: a Review. *arXiv preprint arXiv:1809.04356* (2018).

[17] Shaghayegh Gharghabi, Yifei Ding, Chin-Chia Michael Yeh, Kaveh Kamgar, Liudmila Ulanova, and Eamonn Keogh. 2017. Matrix profile VIII: domain agnostic online semantic segmentation at superhuman performance levels. In *ICDM*. IEEE, 117–126.

[18] Zaid Harchaoui, Félicien Vallet, Alexandre Lung-Yut-Fong, and Olivier Cappé. 2009. A regularized kernel-based approach to unsupervised audio segmentation. In *ICASSP*. IEEE, 1665–1668.

[19] T Heldt, MB Oefinger, M Hoshiyama, and RG Mark. 2003. Circulatory response to passive and active changes in posture. In *Computers in Cardiology, 2003*. IEEE, 263–266.

[20] Shohei Hido, Tsuyoshi Idé, Hisashi Kashima, Harunobu Kubo, and Hirofumi Matsuzawa. 2008. Unsupervised change analysis using supervised learning. In *PKDD*. Springer, 148–159.

[21] Yoshinobu Kawahara and Masashi Sugiyama. 2012. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5, 2 (2012), 114–127.

[22] Eamonn Keogh, Jessica Lin, and Ada Fu. 2005. Hot SAX: Efficiently finding the most unusual time series subsequence. In *ICDM*. Ieee, 8–pp.

[23] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. 2018. State representation learning for control: An overview. *Neural Networks* 108 (2018), 379–392.

[24] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2014. Autoplait: Automatic mining of co-evolving time sequences. In *SIGMOD*. 193–204.

[25] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover. 2009. Exact discovery of time series motifs. In *ICDM*. SIAM, 473–484.

[26] R Quian Quiroga, S Blanco, OA Rosso, H Garcia, and A Rabinowicz. 1997. Searching for hidden information with Gabor Transform in generalized tonic-clonic seizures. *Electroencephalography and clinical Neurophysiology* 103, 4 (1997), 434–439.

[27] Patrick Schäfer. 2015. The BOSS is Concerned with Time Series Classification in the Presence of Noise. *Data Mining and Knowledge Discovery* 29, 6 (2015), 1505–1530.

[28] Patrick Schäfer and Mikael Högqvist. 2012. SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets. In *EDBT*. ACM, 516–527.

[29] Andrew Jhon Scott and M Knott. 1974. A cluster analysis method for grouping means in the analysis of variance. *Biometrics* (1974), 507–512.

[30] Charles Truong, Laurent Oudre, and Nicolas Vayatis. 2020. Selective review of offline change point detection methods. *Signal Processing* 167 (2020), 107299.

[31] Yao-Hung Hubert Tsai, Yue Wu, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2020. Demystifying self-supervised learning: An information-theoretical framework. *arXiv preprint arXiv:2006.05576* (2020).

[32] Gerrit JJ van den Burg and Christopher KI Williams. 2020. An evaluation of change point detection algorithms. *arXiv preprint arXiv:2003.06222* (2020).

[33] Denis S Willett, Justin George, Nora S Willett, Lukasz L Stelinski, and Stephen L Lapointe. 2016. Machine learning for characterization of insect vector feeding. *PLoS computational biology* 12, 11 (2016), e1005158.

[34] Y Chen, E Keogh, B Hu, N Begum, A Bagnall, A Mueen and G Batista . 2015. The UCR Time Series Classification Archive. http://www.cs.ucr.edu/~eamonn/time_series_data.

[35] Jiaping Zhao and Laurent Itti. 2016. Decomposing time series with application to temporal segmentation. In *WACV*. IEEE, IEEE, 1–9.

[36] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, and Eamonn Keogh. 2018. Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. In *ICDM*. IEEE, 837–846.