Semi-supervised change point detection using active learning

Arne De Brabandere[1⋆], Zhenxiang Cao[2⋆], Maarten De Vos[2,3], Alexander Bertrand[2,4], and Jesse Davis[1]

[1] DTAI, Department of Computer Science, KU Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be
[2] STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, Department of Electrical Engineering, KU Leuven, Belgium
{firstname.lastname}@esat.kuleuven.be
[3] Department of Development and Regeneration, KU Leuven, Belgium
[4] Leuven.AI - KU Leuven institute for AI

Abstract. The goal of change point detection (CPD) is to find abrupt changes in the underlying state of a time series. Currently, CPD is typically tackled using fully supervised or completely unsupervised approaches. Supervised methods exploit labels to find change points that are as accurate as possible with respect to these labels, but have the drawback that annotating the data is a time-consuming task. In contrast, unsupervised methods avoid the need for labels by making assumptions about how changes in the underlying statistics of the data correlate with changes in a time series' state. However, these assumptions may be incorrect and hence lead to identifying different change points than a user would annotate. In this paper, we propose an approach in between these two extremes and present AL-CPD, an algorithm that combines active and semi-supervised learning to tackle CPD. AL-CPD asks directed queries to obtain labels from the user and uses them to eliminate incorrectly detected change points and to search for new change points. Using an empirical evaluation on both synthetic and real-world datasets, we show that our algorithm finds more accurate change points compared to existing change point detection methods.

Keywords: Change point detection · Active learning · Semi-supervised learning.

1 Introduction

Time series are time-ordered sequences that report the observed values of a variable of interest at each time step. The observed values depend on the underlying state of the system, which usually does not remain constant but changes over time. For example, when monitoring a person's physical activity using on-body accelerometers, the state of the system is the activity that is currently being

⋆ Equal contribution

performed, which affects the observed values in the acceleration signals. The problem of change point detection (CPD) is to locate abrupt changes in the underlying state of a time series [1]. In the example of physical activity monitoring, change points occur when the person transitions from one activity to another.

Existing CPD algorithms can be categorised into two groups: supervised and unsupervised approaches. Supervised CPD methods exploit labels to learn where the change points of a time series are located. These methods treat the CPD problem as a multi-class [20] or binary [10, 11] classification task. Multi-class methods classify each window of the time series as its corresponding state. Change points are detected when the predicted state changes between two consecutive windows. Binary classification methods learn whether a given location in the time series is a change point or not. The features used as input to supervised methods depend on the application and the type of data that is used. For example, supervised segmentation for transportation mode detection [20] uses application-specific features such as the magnitude of the acceleration measured by an accelerometer, or the speed derived from GPS data. Therefore, these methods are hard to generalise to other datasets. Moreover, they require a sufficient amount of labelled data in order to achieve good detection accuracy, and the resources and time needed for annotating the data may not always be available.

Unsupervised CPD methods can be subdivided into classical model-based approaches and data-driven model-free approaches. Classical approaches such as the cumulative sum (CUSUM) [3] and the generalised likelihood ratio (GLR) [2] use a sliding window approach to estimate the underlying statistical models of adjacent subsequences of the time series. The parameters in the estimated models are assumed to be constant if there is no change point in between. Hence, a change point is detected when the models significantly differ. Some other studies [14, 15] further improved the performance of these approaches by estimating the density ratio. The assumption of these methods is that the density ratio of consecutive window pairs remains constant when there is no change point. Approaches such as FLOSS [12] and ESPRESSO [9] rely on changes in temporal shape patterns, whereas AutoPlait [16] detects changes in the parameters of a hidden Markov model learned from the time series. However, all model-based algorithms face the same problem: their final performance heavily depends on whether the actual data follows the assumed parametric model. It is often hard to guarantee this condition in complex real-world datasets. Recently, a variety of unsupervised data-driven learning algorithms have been proposed, which are typically based on (deep) neural networks such as convolutional neural networks (CNN) [17] and graph neural networks (GNN) [22]. However, these methods also make assumptions about the changes in the underlying statistics of the time series. For example, the time-invariant representation (TIRE) framework [8] uses an autoencoder under the assumption that some latent features should remain constant in the absence of a change point.

Despite the wide range of existing algorithms, finding the correct change points of a time series remains a challenging task because the time series may have multiple possible definitions of change points. On the one hand, supervi-

sion enables tailoring the method to the problem at hand, but requiring fully annotated data imposes a huge time burden on a user. On the other hand, unsupervised approaches rely on assumptions which may not correspond to the user's intuitions or may not be appropriate for a specific problem. Hence a mismatch can arise between the change points found by the algorithm and the correct ones.

In order to fill this gap, we propose an active, semi-supervised approach to change point detection. By employing active learning, we can focus the labelling effort to specific locations in the time series that will be particularly informative in order to minimize the manual effort. In summary, our contributions are as follows:

1. We propose an active learning approach to CPD (AL-CPD) which asks a small number of directed queries to the user in order to obtain labels. AL-CPD exploits these labels to (1) eliminate incorrectly detected change points and (2) detect new change points in a semi-supervised setting.
2. We perform an empirical evaluation on both synthetic and real-world time series and show that AL-CPD outperforms existing CPD methods.

2 AL-CPD

In change point detection, the goal is to find abrupt transitions in the underlying state of a time series. More specifically, we define the problem as follows:

Given: A set of n time series $x_1, \ldots, x_n$
Find: Locations $t_i^1, \ldots, t_i^{s_i}$ of the change points of each series x_i

The input consists of n sequences $x_1, \ldots, x_n$ where each x_i is a numerical time series that can be univariate or multivariate. Instead of representing the data as a single long time series, our input format can represent time series collected over multiple batches. For example, an activity recognition dataset is typically collected from multiple subjects. Our data format can represent each subject's data as a separate sequence. Each sequence x_i consists of multiple segments that each correspond to an underlying state of the time series. The goal is to find the locations $t_i^1, \ldots, t_i^{s_i}$ of the change points, i.e., the transitions between the segments, for each sequence. Note that CPD can be tackled in an *offline* or *online* setting. Here, we only consider the offline case where all data is collected before running the algorithm.

Our algorithm approaches the offline CPD task by employing an active learning strategy that queries a human annotator in order to intelligently acquire labels that the algorithm can exploit to better identify the relevant change points. Because each query entails a manual effort from the user, the goal is to find good change points using a small number of queries. Designing such an algorithm poses two key challenges. First, given a set of potential change points, which ones should be queried to the user? Because this focuses on a fixed set of change points, this step of the algorithm is concerned with increasing the precision, that is, eliminating false positive change points. Second, how can the

acquired labels be used to improve the algorithm used to detect candidate change points? This requires moving from an unsupervised change point detection setting to a semi-supervised setting. The effect of this step is to identify new change points in order to increase the recall, i.e., the fraction of ground truth change points that are found by the algorithm.

2.1 Algorithm outline

Algorithm 1 shows the main steps of AL-CPD, our proposed change point detection algorithm. Initially, the algorithm has no labels and hence operates in an unsupervised setting. As shown on lines 1–3, we run TIRE [8] on each sequence to find the initial set of candidate change points C. By automatically learning features using an autoencoder (AE), TIRE makes no distributional assumptions about the change points. The AE takes a window of size s as input and learns two types of features: time-invariant features ($\mathbf{f}^{ti}$) which are used for detecting change points, and time-variant features ($\mathbf{f}^{tv}$) which are only used to reconstruct the time series. When no change point is present, the time-invariant features should remain constant. Therefore, the model minimises the dissimilarity between the time-invariant features extracted from adjacent windows using a time-invariant loss function:

$$\mathcal{L}^{ti} = \sum_{t} ||\mathbf{f}^{ti}_{t+1} - \mathbf{f}^{ti}_{t}||_2.$$

where $\mathbf{f}^{ti}_{t}$ and $\mathbf{f}^{ti}_{t+1}$ are the time-invariant features at time t and $t+1$, respectively. After training, TIRE detects candidate change points by finding peaks in the dissimilarity between the time-invariant features of consecutive windows.

On lines 4–16, the active learning phase of our algorithm improves the candidate change points using two steps: (1) selecting candidates (lines 8–11), and (2) finding new candidates (lines 12–15). These steps represent our key algorithmic contributions. The active learning phase asks queries one by one until the number of queries reaches a user-defined query budget b. When this phase terminates, the algorithm returns all selected candidate change points. In the following two subsections, we describe each step in detail.

2.2 Selecting candidate change points

The first step employs an active learning strategy to identify and remove incorrectly detected candidate change points. For this, we train a random forest classifier [4] m that classifies each candidate in C as a correct or incorrect change point and only keep the candidates predicted as correct change points. We use a model instead of TIRE's change point score in order to avoid querying the label for multiple similar candidate change points. While other classification methods could be relevant, we selected random forests due to their computational efficiency and their ability to select relevant features.

In order to train a model, we construct a training set as follows. First, we extract a feature representation from each candidate change point t in C using

Algorithm 1: AL-CPD

Input: Time series $X = \{x_1, \ldots, x_n\}$, window size s, query budget b
Output: Locations of the change points of each time series x_i

```
Initialisation
C = TIRE(X, s)
r = ⌊|C| * 0.1⌋
Active learning
Q = ∅
m = TrainClassifier(C, Q, s)
while |Q| < b do
    q = LeastCertainCandidate(C, m)
    a = Query(q)
    Q = Q ∪ {(q, a)}
    m = TrainClassifier(C, Q, s)
    if |Q| mod r = 0 then
        T = {t ∈ C | p_m(t) > 0.9}
        C = C ∪ Filter(STIRE(X, T, s), C, s)
    end
end
return {t ∈ C | p_m(t) > 0.5}
```

TSFuse [6]. While there exist feature extraction systems that compute a similar set of features, we employ TSFuse because this feature extraction system has an efficient implementation. In an active learning system, this is important to minimise the time that the user has to wait. Using the *fast* set of transformers listed in [7], we build a feature vector F_1 from the interval $[t-s, t]$ and F_2 from $[t, t+s]$ for each candidate t. We then compute the difference $\Delta F = F_2 - F_1$ to measure the change in the feature values. Second, because the number of labelled examples is initially small, we use the local and global consistency label spreading algorithm [23] to increase the amount of labelled data.[5] Third, every candidate that has a label or for which the propagated label has a certainty larger than 90% is added to the training set.

We employ an uncertainty sampling active learning strategy to acquire labels. In each iteration of Algorithm 1, the LeastCertainCandidate(C, m) function computes the certainty of each candidate t in C as $|p_m(t) - 0.5|$ where $p_m(t)$ is the probability predicted by the model m. It returns the candidate with the lowest certainty as the query q. The Query(q) function obtains the answer a

[5] Because the label propagation algorithm performs poorly when given high-dimensional data, we first reduce the dimensionality of the feature space using a principal component analysis (PCA) transformation (setting the number of components such that the explained variance is at least 0.9) and standardise the PCA components.

from the user, which is *true* if there is a change point close to queried candidate change point and *false* otherwise. We add each query-answer tuple (q, a) to Q.

2.3 Finding new candidate change points

Whereas the first step focuses on improving the precision by selecting change points, the second step aims to identify new change points in order to improve the recall. Using the unsupervised TIRE approach to find the initial candidate change points may result in some of the true change points being missed. Therefore, we propose a semi-supervised version of TIRE ("STIRE").

A key challenge to adapting TIRE is its time-invariant loss [8], which pushes the feature representations of neighbouring windows to be close to each other, even when labelled data indicates that this should not be the case due to the presence of a ground-truth change point. Therefore, we modify the time-invariant loss function such that it only forces the time-invariant features for consecutive windows to be close to each other in the latent space when the algorithm is confident that no change point occurs. To this end, we assign labels to all input time windows corresponding to their underlying state. The labels vary only at the temporal indices of confident accurate detections, i.e., the candidate change points for which the random forest model of step 1 predicts a probability larger than 0.9. We replace the time-invariant loss with the triplet loss [21] which is defined as follows:

$$\mathcal{L}^{tri} = \sum_t \max\{d_p(\mathbf{f}_t^{ti}) - d_n(\mathbf{f}_t^{ti}) + \gamma, 0\},$$

with

$$d_p(\mathbf{f}_t^{ti}) = \begin{cases} ||\mathbf{f}_{t+1}^{ti} - \mathbf{f}_t^{ti}||_2^2 & \text{if } p_m(t) > 0.9 \text{ or } a \text{ is } \textit{true} \\ ||\mathbf{f}_t^{ti} - \mathbf{f}_{t-1}^{ti}||_2^2 & \text{otherwise} \end{cases}$$

and

$$d_n(\mathbf{f}_t^{ti}) = ||\mathbf{f}_t^{ti} - \mathbf{f}_N^{ti}||_2^2,$$

where γ represents the pre-defined margin, and $p_m(t)$ is the probability that there is a change point at time t (more specifically a change between time $t-1$ and t) as predicted by the random forest model m. $\mathbf{f}_t^{ti}$ represents the time-invariant features at time t and $\mathbf{f}_N^{ti}$ denotes the time-invariant features of a negative time window sample, i.e., the time-invariant features extracted from a time window with a different label than the current window at time t. We always select this negative time window randomly from all the segments that come before the previous or after the next true positive change point that was selected by the random forest model of step 1 ($p_m(t) > 0.9$). Similar to the original TIRE model, we also include the reconstruction loss:

$$\mathcal{L}^{rec} = \sum_t ||\widehat{w}_t - w_t||_2^2,$$

which encourages the encoded features to contain all information needed for reconstructing the current input window w_t at time t, where $\widehat{w}_t$ denotes the reconstructed window. Finally, the reconstruction loss is combined with the triplet loss via a weighted sum:

$$\mathcal{L} = \mathcal{L}^{rec} + \lambda \mathcal{L}^{tri},$$

where λ controls the balance between the two losses.

After training STIRE, we filter the change points by removing all duplicates, i.e., all candidates that were previously identified. We only keep the candidates for which the time distance to any candidate change point in C is larger than the window size s. Since training STIRE can be time-consuming, we only run this step after every r queries, where we set r to 10% of the number of initial change point candidates: $r = \lfloor |C| * 0.1 \rfloor$.

3 Experiments

We evaluate our active change point detection algorithm on both synthetic and real-world time series datasets to answer the following research questions:

Q1: Can AL-CPD detect change points more accurately than existing methods?
Q2: How many labels does AL-CPD need to find accurate change points?
Q3: How much do each of AL-CPD's two components, (1) using a random forest to select candidates, and (2) using a semi-supervised TIRE variant to find new candidates, contribute to its overall performance?
Q4: What is the sensitivity of AL-CPD to its hyperparameter values?

Because we run our experiments on multiple different applications, we are unable to evaluate application-specific supervised methods. Therefore, we only compare our approach to unsupervised baselines:

GLR [2] The Generalised Likelihood Ratio method fits an auto-regressive model on each adjacent window pair of the time series and detects change points by measuring the dissimilarity of the parameters in the AR model.
RuLSIF [15] The Relative unconstrained Least-Squares Importance Fitting method detects change points by estimating the density ratio of each pair of consecutive windows.
KL-CPD [5] The Kernel Learning CPD method optimises a lower bound of test power using an auxiliary generative model. It learns features using a Seq2Seq model and measures the dissimilarity between neighbouring windows using the maximum mean discrepancy.
TIRE [8] The Time-Invariant REpresentation model maps overlapping windows of the time series onto a feature space using an autoencoder. Change points are detected based on the dissimilarity between windows in the learned feature space.
FLOSS [12] The Fast Low-cost Online Semantic Segmentation algorithm uses the Matrix Profile to find changes in the temporal shape patterns of the time series. It requires the number of ground truth segments as input.

3.1 Datasets

We run the experiments on seven datasets, of which four are artificially constructed and three are based on real-life measurements. Table 1 shows the properties of each dataset. The synthetic datasets are similar to those introduced in [8] and [15]. Three of these synthetic datasets are generated based on a 1-dimensional auto-regressive model:

$$s(t) = a_1 s(t-1) + a_2 s(t-2) + \epsilon_t \tag{1}$$

in which the error term ϵ_t follows a Gaussian distribution $\epsilon_t \sim \mathcal{N}(\mu_t, \sigma_t^2)$. In our experiments, we set the initial state in (1) as: $s(1) = s(2) = 0$ and the default values of the parameters are set to the same values as in [8]: $a_1 = 0.6$, $a_2 = -0.5$, $\mu_t = 0$, and $\sigma_t = 1.5$ unless explained otherwise. Each of these datasets consists of 10 randomly generated sequences. In each sequence, 48 change points are inserted along the temporal axis at each $t_n = t_{n-1} + \lfloor \tau_n \rfloor$, with $t_0 = 0$ and $t_n \sim \mathcal{N}(100, 10)$. We introduce 4 types of change points, leading to the following datasets:

Jumping Mean (JM) The Jumping Mean dataset is generated by changing the value of μ_t at each t_n.

Scaling Variance (SV) In the Scaling Variance dataset, the value of σ_t is changed at each t_n.

Changing coefficients (CC) Here, we set $a_2 = 0$ and alternately draw a_1 from two independent uniform distributions every time a change point is crossed.

Gaussian Mixtures (GM) In this dataset, the time samples in the consecutive segments are alternatively sampled from two different Gaussian mixture distributions.

We include three real-world datasets:

Activity Recognition 1 (AR1) The HASC Challenge 2011 dataset [13] consists of human activity recognition data collected by a triaxial accelerometer. Similar to [8], we select the data from one person (subject 671) and use the magnitude of the acceleration as input. Each segment corresponds to one of the following six activities: staying still, walking, jogging, skipping, ascending stairs, and descending stairs.

Activity Recognition 2 (AR2) We collected a second activity recognition dataset from 8 participants. Each participant performed a sequence of activities consisting of standing, walking, jogging, cycling, ascending stairs, and descending stairs. Similar to the AR1 dataset, the magnitude of the acceleration is collected by a triaxial accelerometer.

Bee Dance (BD) The bee dance dataset [18] consists of six sequences of a bee performing a three-stage waggle dance. Each sequence is a three-dimensional time series representing the location in 2D coordinates and angle differences.

All datasets except AR2 are the same datasets as those used in [8]. We did not include the well log dataset as the number of change points for this dataset is too small to evaluate the active learning step of our proposed algorithm.

Table 1. Dataset properties: number of sequences, and the length and number of change points per sequence (min.–max.).

	Sequences	Length	Change points
JM	10	4836–4925	48
SV	10	4834–4918	48
GM	10	4847–4932	48
CC	10	4864–4907	48
AR1	1	39397	36
AR2	8	15003–25103	22
BD	6	602–1124	15–28

3.2 Methodology

Hyperparameter settings The two steps of our algorithm rely on a window size s. For each dataset, we use a window size smaller than the expected interval between change points, but long enough to capture the statistics of the segments. We set s to 30 for the synthetic datasets (JM, SV, GM, CC), 300 for the activity recognition datasets (AR1, AR2), and 15 for the bee dance dataset (BD).

In addition to the window size, our algorithm has several parameters that are independent of the dataset. For the random forest model, we use the implementation of scikit-learn [19] with the default hyperparameter settings. For (S)TIRE, we learn both time-domain and frequency-domain features. Each auto-encoder learns 3 features: 2 time-invariant features and 1 time-variant feature. In the loss function, the values of γ and λ are set to 0.1 and 0.001, respectively. We train the networks for 200 epochs using the Adam optimiser.

Evaluation We compare the detected change points to the ground truth change points by computing the precision, recall, and F1 score. These metrics are defined as follows:

$$\text{precision} = \frac{TP}{TP + FP} \quad \text{recall} = \frac{TP}{TP + FN} \quad \text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The number of true positives (TP) is computed as the number of predicted change points that are within a distance s from one of the ground truth change points. Any predicted change point that is further than s from all ground truth change points is counted as a false positive (FP). The number of false negatives (FN) is the number of ground truth change points that have a distance larger than s to any predicted change point.

Typically, evaluating active learning methods involves reporting the performance in terms of the number of examples labelled by an annotator. In our setting, each example corresponds to one of the candidate change points. However, the number of candidate change points identified by AL-CPD varies as more labels are acquired. Therefore, we report the performance relative to the

amount of work a human would have to do to fully annotate the data. This would require partitioning each sequence into non-overlapping windows of size s and then labelling each window as either containing a change point or not. We refer to the total labelling effort as the number of potential change points in a dataset:

$$P = \sum_{i=1}^{n} \left\lfloor \frac{N_i}{s} \right\rfloor$$

where n ranges over sequences and N_i is the length of the ith sequence. The number of potential change points P for each dataset is as follows:

	JM	SV	GM	CC	AR1	AR2	BD
P	1622	1617	1622	1621	131	516	328

When running active learning, AL-CPD receives a *true* answer if its queried candidate change point t is within s samples from at least one ground truth change point. Otherwise, it receives a *false* answer.

3.3 Q1: Comparison to existing change point detection algorithms

Table 2 shows the precision, recall, and F1 score for the baselines and AL-CPD. We run AL-CPD for three different query budgets that correspond to 5%, 10% and 20% of all potential change points. In terms of the F1 score, AL-CPD substantially outperforms the baselines on all datasets after querying 20% of the potential change point locations. On the four synthetic datasets, querying only 5% of the potential change points leads to better results compared to the baselines. Because of the well-defined underlying process, the synthetic datasets contain many similar change points. Hence, learning the definition of a change point requires fewer labelled examples (i.e., fewer queries) compared to the more complex real-world datasets.

GLR, RuLSIF, and KL-CPD achieve a perfect recall on all datasets except GM and BD for RuLSIF. However, these methods find many false positives, which results in a low precision. Hence, for the 20% query budget, the improved performance of AL-CPD over these baselines can be attributed to the better precision. FLOSS performs worse than all other baselines and AL-CPD in terms of all evaluation metrics. Compared to TIRE, our algorithm achieves a better precision on all datasets and a better recall on 5 out of 7 datasets after querying 10% of the potential change point locations.

Note that for the AR1 dataset, AL-CPD scores zero for all metrics for the 5% query budget. This occurs because none of the queried candidate windows contained a change point. Hence, the learned random forest predicts that no other windows contain a change point, leading to all candidate change points being discarded.

Table 2. Precision, recall, and F1 score of each baseline and AL-CPD after querying 5%, 10%, and 20% of all possible change point locations. For each dataset, we highlight the best-performing baseline in bold and annotate each baseline outperformed by AL-CPD after querying 20%, 10%, and 5% of the change point locations with |, ||, and |||, respectively.

Precision

	Baselines					AL-CPD		
	GLR	RuLSIF	KL-CPD	FLOSS	TIRE	5%	10%	20%
JM	0.584$^{\|\|\|}$	0.590$^{\|\|\|}$	0.573$^{\|\|\|}$	0.548$^{\|\|\|}$	0.861$^{\|\|\|}$	0.923	0.945	**0.990**
SV	0.585$^{\|\|\|}$	0.589$^{\|\|\|}$	0.581$^{\|\|\|}$	0.556$^{\|\|\|}$	0.702$^{\|\|\|}$	0.756	0.810	**0.908**
GM	0.578$^{\|\|\|}$	0.596$^{\|\|\|}$	0.582$^{\|\|\|}$	0.565$^{\|\|\|}$	0.906$^{\|\|\|}$	0.982	**1.000**	**1.000**
CC	0.583$^{\|\|\|}$	0.057$^{\|\|\|}$	0.574$^{\|\|\|}$	0.571$^{\|\|\|}$	0.738$^{\|\|\|}$	0.790	0.836	**0.954**
AR1	0.354$^{\|\|}$	0.366$^{\|\|}$	0.382$^{\|\|}$	0.389$^{\|\|}$	0.500$^{\|\|}$	0.000	0.889	**0.941**
AR2	0.485$^{\|\|\|}$	0.532$^{\|\|\|}$	0.523$^{\|\|\|}$	0.415$^{\|\|\|}$	0.630$^{\|\|\|}$	0.826	0.856	**0.973**
BD	0.670$^{\|\|\|}$	0.679$^{\|\|\|}$	0.682$^{\|\|\|}$	0.474$^{\|\|\|}$	0.741$^{\|\|\|}$	0.809	0.833	**0.869**

Recall

	Baselines					AL-CPD		
	GLR	RuLSIF	KL-CPD	FLOSS	TIRE	5%	10%	20%
JM	**1.000**	**1.000**	**1.000**	0.548$^{\|\|\|}$	0.944$^{\|\|\|}$	0.962	0.977	0.979
SV	**1.000**	**1.000**	**1.000**	0.556$^{\|\|\|}$	0.852$^{\|\|\|}$	0.919	0.933	0.940
GM	**1.000**	0.994	**1.000**	0.565$^{\|\|\|}$	0.987$^{\|\|}$	0.985	0.994	0.994
CC	**1.000**	**1.000**	**1.000**	0.571$^{\|\|\|}$	0.783$^{\|\|\|}$	0.831	0.875	0.875
AR1	**1.000**	**1.000**	**1.000**	0.556$^{\|}$	0.861	0.000	0.361	0.611
AR2	**1.000**	**1.000**	**1.000**	0.591$^{\|\|}$	0.830	0.551	0.722	0.807
BD	**1.000**	0.852	**1.000**	0.453$^{\|\|\|}$	0.717$^{\|\|\|}$	0.764	0.775	0.852

F1 score

	Baselines					AL-CPD		
	GLR	RuLSIF	KL-CPD	FLOSS	TIRE	5%	10%	20%
JM	0.738$^{\|\|\|}$	0.742$^{\|\|\|}$	0.729$^{\|\|\|}$	0.548$^{\|\|\|}$	0.900$^{\|\|\|}$	0.942	0.961	**0.985**
SV	0.738$^{\|\|\|}$	0.741$^{\|\|\|}$	0.735$^{\|\|\|}$	0.556$^{\|\|\|}$	0.770$^{\|\|\|}$	0.829	0.867	**0.923**
GM	0.733$^{\|\|\|}$	0.745$^{\|\|\|}$	0.736$^{\|\|\|}$	0.565$^{\|\|\|}$	0.945$^{\|\|\|}$	0.983	**0.997**	**0.997**
CC	0.736$^{\|\|\|}$	0.108$^{\|\|\|}$	0.729$^{\|\|\|}$	0.571$^{\|\|\|}$	0.760$^{\|\|\|}$	0.810	0.855	**0.912**
AR1	0.523$^{\|}$	0.536$^{\|}$	0.553$^{\|}$	0.458$^{\|\|}$	0.633$^{\|}$	0.000	0.514	**0.741**
AR2	0.649$^{\|\|\|}$	0.692$^{\|\|}$	0.684$^{\|\|}$	0.486$^{\|\|\|}$	0.709$^{\|\|}$	0.650	0.774	**0.880**
BD	0.799$^{\|}$	0.752$^{\|\|\|}$	0.806$^{\|}$	0.453$^{\|\|\|}$	0.726$^{\|\|\|}$	0.779	0.793	**0.857**

3.4 Q2: Labelling effort of AL-CPD

We investigate the effect of the number of acquired labels on AL-CPD's performance. Specifically, we evaluate how many queries are required to obtain an F1 score that is larger than a chosen percentage of the final F1 score achieved when using an unlimited query budget.

Table 3 shows the percentage of change point locations needed to achieve an F1 score of at least 80%, 90% and 95% of the final F1 score. Achieving an F1 score of at least 80% of the final F1 score requires labelling between 0.1% and 21.4% of all potential change point locations. In other words, the user saves between 78.6% and 99.9% of the labelling effort compared to manually labelling each window. Even to achieve an F1-score of 95% of the final one, AL-CPD still reduces the effort compared to completely labelling the data by at least 68.7%.

Table 3. Percentage of change point locations that the user has to label in order to obtain an F1 score of at least 80%, 90% and 95% of the final F1 score.

	JM	SV	GM	CC	AR1	AR2	BD
80%	0.1%	1.9%	0.7%	0.2%	21.4%	7.2%	3.7%
90%	0.1%	10.9%	0.7%	9.3%	30.5%	16.1%	10.7%
95%	5.1%	19.3%	0.9%	15.7%	31.3%	22.9%	25.0%

3.5 Q3: Contribution of each component of AL-CPD

Our algorithm has two components: (1) selecting candidates by training a classifier, and (2) finding new candidates by training TIRE in a semi-supervised setting. For research question **Q3**, we analyse the effect of each component on the algorithm's performance. To do so, we perform an ablation study that compares the AL-CPD algorithm to two variants:

1. The **A** variant includes only component 1 of our algorithm, i.e., the active learning step for selecting candidates.
2. The **S** variant includes only component 2 of our algorithm, i.e., the semi-supervised setting of TIRE for finding new candidates.

In order to compare our algorithm to the two variants, we evaluate the area under the learning curve (ALC) of the precision, recall, and F1 score. The ALC is defined as follows:

$$\text{ALC} = \frac{1}{n} \sum_{i=0}^{n} e(i)$$

where n is the total number of queries and $e(i)$ is the evaluation metric (i.e., precision, recall, or F1 score) computed after the ith query.

Table 4 shows that the **A** variant results in a better precision than the **S** variant on most datasets. However, because the candidate selection component of the **A** variant removes some of the true positive change points, this variant has the lowest recall. By searching for new candidates, the **S** variant improves the recall on all datasets. In terms of the F1-score, the full AL-CPD algorithm outperforms both **A** and **S** on four datasets. For the other three datasets, the **S** variant outperforms AL-CPD due to a better recall.

Table 4. Area under the learning curve of the precision, recall, and F1 score for each variant. The performance of the best variant is highlighted in bold.

	Precision			**Recall**			**F1 score**		
	A	S	AL-CPD	A	S	AL-CPD	A	S	AL-CPD
JM	**0.973**	0.961	0.968	0.941	**0.974**	0.972	0.956	0.967	**0.970**
SV	0.885	0.832	**0.895**	0.847	0.926	**0.927**	0.862	0.875	**0.909**
GM	0.974	**0.977**	0.974	0.965	**0.991**	0.971	0.973	**0.984**	0.976
CC	0.910	0.849	**0.914**	0.767	**0.874**	0.857	0.829	0.860	**0.883**
AR1	0.757	0.759	**0.798**	0.515	**0.868**	0.560	0.592	**0.802**	0.637
AR2	0.923	0.840	**0.933**	0.700	**0.861**	0.760	0.782	**0.844**	0.827
BD	0.864	0.836	**0.877**	0.701	0.805	**0.821**	0.769	0.819	**0.845**

3.6 Q4: Sensitivity analysis

For research question **Q4**, we analyse the hyperparameter sensitivity of AL-CPD. Our algorithm has three main hyperparameters: the window size s and two hyperparameters specific to STIRE: the balance between the reconstruction and triplet loss λ and the margin γ.

Figure 1 compares the ALC of the F1 score for three different values of each hyperparameter. For the window size s, we multiply the default window sizes by a factor 0.5, 1 and 1.5 and report the average ALC of the F1-score over all datasets. On average, shorter window sizes decrease the F1 score. This is expected since capturing the characteristics of a segment requires a sufficiently long portion of the time series. Longer windows do not further improve the F1 score, and may even decrease the performance when exceeding the distance between consecutive change points. The performance of our algorithm is robust w.r.t. the reconstruction and triplet loss λ and the margin γ, since the ALC of the F1 score is almost not affected by the values of these hyperparameters.

4 Conclusion

This paper presented AL-CPD, a change point detection algorithm that combines active and semi-supervised learning. Instead of only relying on assumptions

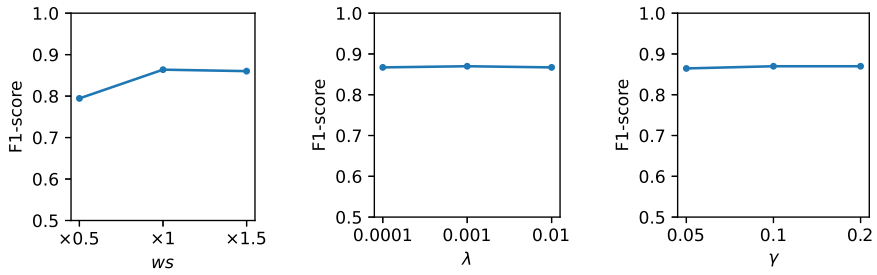

Fig. 1. ALC of the F1 score for three different values of the window size s, the balance between the reconstruction and triplet loss, and the margin γ. The ALC is averaged over all datasets.

about the changes in the underlying statistics of the given time series, AL-CPD asks directed queries to the user in order to obtain labels. Our algorithm exploits these labels to eliminate incorrectly detected change points and to search for new change points. In an empirical evaluation, we compared the performance of AL-CPD to existing unsupervised CPD methods and showed that AL-CPD is able to find more accurate change points with a query budget of at most 20% of all potential change points.

Acknowledgements

This work is supported by the Research Foundation Flanders (FWO) under TBM grant number T004716N, by the Flemish government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, and by VLAIO ICON-AI CONSCIOUS (HBC.2020.2795).

References

1. Aminikhanghahi, S., Cook, D.J.: A survey of methods for time series change point detection. Knowledge and information systems **51**(2), 339–367 (2017)
2. Appel, U., Brandt, A.V.: Adaptive sequential segmentation of piecewise stationary time series. Information sciences **29**(1), 27–56 (1983)
3. Basseville, M., Nikiforov, I.V., et al.: Detection of abrupt changes: theory and application, vol. 104. prentice Hall Englewood Cliffs (1993)
4. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
5. Chang, W.C., Li, C.L., Yang, Y., Póczos, B.: Kernel change-point detection with auxiliary deep generative models. arXiv preprint arXiv:1901.06077 (2019)
6. De Brabandere, A., Op De Beéck, T., Hendrickx, K., Meert, W., Davis, J.: TSFuse: Automated feature construction for multiple time series data. Machine Learning (2022)
7. De Brabandere, A., Robberechts, P., Op De Beéck, T., Davis, J.: Automating feature construction for multi-view time series data. In: ECMLPKDD Workshop on Automating Data Science (2019)

8. De Ryck, T., De Vos, M., Bertrand, A.: Change point detection in time series data using autoencoders with a time-invariant representation. IEEE Transactions on Signal Processing **69**, 3513–3524 (2021)
9. Deldari, S., Smith, D.V., Sadri, A., Salim, F.: Espresso: entropy and shape aware time-series segmentation for processing heterogeneous sensor data. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **4**(3), 1–24 (2020)
10. Desobry, F., Davy, M., Doncarli, C.: An online kernel change detection algorithm. IEEE Transactions on Signal Processing **53**(8), 2961–2974 (2005)
11. Feuz, K.D., Cook, D.J., Rosasco, C., Robertson, K., Schmitter-Edgecombe, M.: Automated detection of activity transitions for prompting. IEEE transactions on human-machine systems **45**(5), 575–585 (2014)
12. Gharghabi, S., Yeh, C.C.M., Ding, Y., Ding, W., Hibbing, P., LaMunion, S., Kaplan, A., Crouter, S.E., Keogh, E.: Domain agnostic online semantic segmentation for multi-dimensional time series. Data mining and knowledge discovery **33**(1), 96–130 (2019)
13. Kawaguchi, N., Yang, Y., Yang, T., Ogawa, N., Iwasaki, Y., Kaji, K., Terada, T., Murao, K., Inoue, S., Kawahara, Y., et al.: HASC2011corpus: Towards the common ground of human activity recognition. In: Proceedings of the 13th international conference on Ubiquitous computing. pp. 571–572 (2011)
14. Kawahara, Y., Sugiyama, M.: Sequential change-point detection based on direct density-ratio estimation. Statistical Analysis and Data Mining: The ASA Data Science Journal **5**(2), 114–127 (2012)
15. Liu, S., Yamada, M., Collier, N., Sugiyama, M.: Change-point detection in time-series data by relative density-ratio estimation. Neural Networks **43**, 72–83 (2013)
16. Matsubara, Y., Sakurai, Y., Faloutsos, C.: Autoplait: Automatic mining of co-evolving time sequences. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. pp. 193–204 (2014)
17. Munir, M., Siddiqui, S.A., Dengel, A., Ahmed, S.: Deepant: A deep learning approach for unsupervised anomaly detection in time series. Ieee Access **7**, 1991–2005 (2018)
18. Oh, S.M., Rehg, J.M., Balch, T., Dellaert, F.: Learning and inferring motion patterns using parametric segmental switching linear dynamic systems. International Journal of Computer Vision **77**(1), 103–124 (2008)
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
20. Reddy, S., Mun, M., Burke, J., Estrin, D., Hansen, M., Srivastava, M.: Using mobile phones to determine transportation modes. ACM Transactions on Sensor Networks (TOSN) **6**(2), 1–27 (2010)
21. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 815–823 (2015). https://doi.org/10.1109/CVPR.2015.7298682
22. Zhang, R., Hao, Y., Yu, D., Chang, W.C., Lai, G., Yang, Y.: Correlation-aware unsupervised change-point detection via graph neural networks (2020)
23. Zhou, D., Bousquet, O., Lal, T., Weston, J., Schölkopf, B.: Learning with local and global consistency. Advances in neural information processing systems **16** (2003)