An Experimental Comparison between Time2State and FLOSS

Chengyu Wang, Kui Wu, Tongqing Zhou, Zhiping Cai

Abstract

The objectives of this experimental study include: (1) clarifying confusion and misconceptions, (2) highlighting the challenges involved in augmenting FLOSS with state detection capabilities, and (3) comparing the performance of Time2State and (augmented) FLOSS.

1 Clarification of Terminologies

In Time2State Wang et al. (2023), we classify related works into two categories: change point detection (CPD) methods and time series segmentation (TSS) methods.

1.1 Change Point Detection (CPD)

The term CPD is ambiguous, and its use in Wang et al. (2023) and Gharghabi et al. (2019) carries entirely distinct connotations, leading to confusion and misinterpretation.

In Gharghabi et al. (2019), CPD is defined as "*a method for detecting various changes in statistical properties of time series, such as the mean, variance or spectral density.*" In this sense, FLOSS is not considered CPD, but rather a tool for identifying the boundary points between semantic segments.

Conversely, in Wang et al. (2023), "change points" refer specifically to the boundary points between semantic segments, and as such, FLOSS and ClaSP Schäfer et al. (2021) are classified as CPD methods. This inconsistent use of terminology in Wang et al. (2023) may cause confusion and misunderstanding. To avoid the problem, we will use the term "boundary point detection" in place of "change point detection" throughout the remainder of this report.

In summary, **FLOSS and ClaSP are boundary point detection methods.**

1.2 Time Series Segmentation (TSS)

The term TSS is also ambiguous, and its use in Wang et al. (2023) and Gharghabi et al. (2019) carries distinct connotations.

In Gharghabi et al. (2019), TSS refers to methods that divide a time series into semantic segments, with each segment having a specific meaning. However, FLOSS only detects the boundary points between these semantic segments, leaving it to humans to provide a state label for each segment. As an example, consider the time series shown in Fig. 1. FLOSS detects the boundary points between the "NM" segments but cannot determine if they have the same semantic meaning (i.e., the same state) without human input. In other words, it depends on humans to label whether or not the first and the second "NW" segments have the same semantic meaning.

In Wang et al. (2023), TSS refers to methods that not only divide a time series into semantic segments but also automatically assign a state label to each segment. This automated labeling feature allows the same label to be assigned to the two "NM" segments in Fig. 1. This difference explains why FLOSS and ClaSP are not considered TSS in Wang et al. (2023). We refer to this automatic labeling capability as "**state detection**" throughout the following discussion.

With these distinctions clarified, we can understand the critical difference between FLOSS and Time2State. FLOSS divides a time series into semantic segments by detecting boundary points between them but relies on humans to determine if different segments share the same state. Time2State, on the other hand, not only divides a time series into semantic segments but also automatically labels the state of each segment.

Therefore, strictly speaking, FLOSS and Time2State are distinct methods. To make FLOSS comparable to Time2State, we need to augment it with state detection capabilities.

Preprint. Under review.

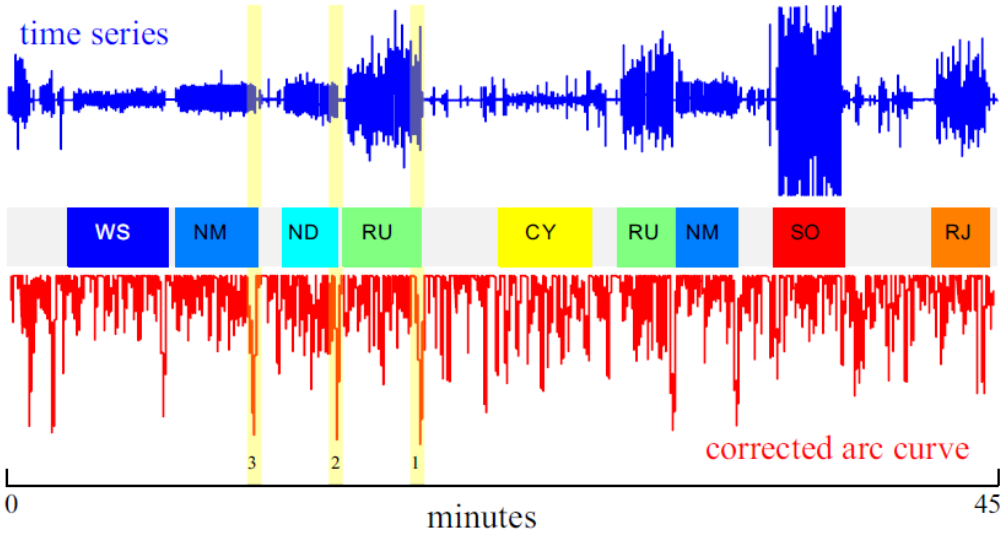


Figure 1: An example of semantic segmentation Gharghabi et al. (2019). FLOSS can find out the boundaries between semantic segments, but the semantic labels (or states) in the middle are given by humans.

1.3 Segment, Regime, and State

In Gharghabi et al. (2019), the authors used the terms “regime" and “state" interchangeably:

> “*The height of the Arc Curve should be the lowest at the location of the boundary between the change of regimes/states.*” and “*Assume we have a system S, which can be in two or more discrete states (or regimes).*”

Following the above description, it is clear that regime and segment are distinct concepts in Gharghabi et al. (2019), and the phrase “Extracting regimes from the CAC" seems to suggest that FLOSS is capable of identifying states. This may be misleading because FLOSS just finds out the boundary points between segments. Of course, if all the states in a time series are unique, e.g., a time series of state sequence A, B, C, D, E, finding the boundary points is equivalent to finding the states. Nevertheless, if the time series consists of recurring states, e.g., state sequence A, B, C, A, B, A, C, FLOSS cannot find the states. It even has difficulties in finding the boundary points without using a parameter *temporal constraint*:

> “*We assume here that the regimes are distinct, for example walk, jog, run. If a regime can be repeated, say walk, jog, walk, our algorithm may have difficulties; that issue will be dealt with in Sect. 3.5."*

To further explain why the statements in Gharghabi et al. (2019) are misleading, we quote the content from Gharghabi et al. (2019):

> “*Our basic regime extracting algorithm requires* ***the user to input k, the number of regimes****. This is similar to many popular clustering algorithms, such as k-means, which require the user to input the k number of clusters. Later we will demonstrate a technique to remove the need to specify k, given some training data to learn from (see Sect. 4.3).*” and
>
> “*While this algorithm is obvious and intuitive, for concreteness, we formally outline in Table 2. Note that numRegimes is not a parameter of the segmentation algorithm per se; it is a parameter of our regime extraction algorithm. By analogy, you can build a dendrogram without specifying parameters, but any algorithm to convert it to partitional clusters will always have a parameter. There may be other ways to extract regime; Table 2 just offers a concrete and simple method.*”

If we follow the above description, for a time series of state sequence A, B, C, A, B, A, C, the number of regimes (states) should be 3. Nevertheless, using $k = 3$ as input to FLOSS would lead to completely wrong results. To make FLOSS work correctly, the input should be $k = 6$, as shown in Fig. 2. With the help of *temporal constraint*, FLOSS can identify the boundary points, but it cannot know that the first A segment and the fourth A segment belong to the same state. We have not found any part of the FLOSS source code that assigns state labels to segments.

The dataset (UCR-SEG) considered by the authors of Gharghabi et al. (2019) is special. In the UCR-SEG dataset, the states/regimes are (nearly) unique, i.e., each state occurs only once and does not recur in each time series, with only one

synthetic time series in the dataset[1] as an exception. In this special case, identifying segment boundaries is (nearly) equivalent to identifying states.

Table 2 REA: Algorithm for extracting regimes

Procedure ExtractRegimes (CAC, numRegimes, *L*)
Input: *CAC – a Corrected Arc Curve*
numRegimes – number of regime changes
L – length of the subsequence
Output: *locRegimes – the locations of the regimes*

1 *locRegimes* = empty array of length *numRegimes*
2 **for** *i*=1 : *numRegimes*
3 *locRegimes*(*i*) = indexOf(min(*CAC*))
4 Set exclusion zone of 5 ×*L* // To prevent matches to "self"
5 **end**
6 **return** *locRegimes*

(a) The algorithm for extracting regimes in Gharghabi et al. (2019). This figure is borrowed from Gharghabi et al. (2019).

```
function [minV, ind]= findLocalMinimums(data, length, n)
%% length
%% n the number of minimum
minV(1:n) = inf;
ind(1:n) = -1;
for i=1:n
    [minV(i), ind(i)] = min(data);
    data(ind(i)-length:ind(i)+length) = inf;
end
end
```

(b) The implementation of the algorithm (`http://www.cs.ucr.edu/~eamonn/FLOSS/`).

Figure 2: The pseudo code and implementation of the regime extraction method of FLOSS.

2 Augmenting FLOSS for State Detection

2.1 Generalizing FLOSS to Multi-dimensional Time Series

We have adopted the approach proposed in Gharghabi et al. (2019) for applying FLOSS to multivariate time series data, which involves averaging the CAC curves. However, as this step was not included in the original FLOSS code[2], we implemented this method ourselves, using the *RunSegmentation* function defined in the *RunSegmentation.m* file.

We also notice that the FLOSS paper mentioned that "*Just using two (or some other small subset) of all the dimensions can segment the activities much more accurately than either using all dimensions or a single dimension. This can be seen as a classic 'goldilocks' observation, and a similar observation is made in the context of time series classification in (Hu et al. 2016). This begs the question of which small subset of dimensions to use. We leave such considerations for future work.*"

Nevertheless, we have not found an automated way to select channels for better segmentation. We thus simply input all channels to FLOSS, just like the same way we treat all other baselines in the Time2State paper.

2.2 Detecting States with Clustering

We follow the same settings used in the Time2State paper to add a clustering component to FLOSS. The clustering component is implemented in Python, using the Time Series KMeans[3] (TSKMeans) algorithm, which is a standard time series clustering method.

We used *euclidean* and *dtw* as the distance measure. When we tested *euclidean*, we pad shorter segments with zeroes to ensure they have the same length, as the segments detected by FLOSS may not have the same length. Note that *dtw* requires downsampling for handling some datasets. This is because (1) the running time of *dtw* is high if the datasets consist of long segments. For example, the running time of *dtw* on a time series in ActRecTut exceeds 30 minutes; (2) the memory overhead of *dtw* is also high. We encountered a memory allocation error while using *dtw* on the synthetic dataset. The details of these errors are provided in Section B.1.

We also tested another standard time series clustering method, namely, KShape[4], to cluster the segments found by FLOSS. Nevertheless, KShape did not produce meaningful results as KShape simply output an error message "*Resumed because of empty cluster*".

3 Evaluation

3.1 Experimental Environment

All experiments are conducted on a machine equipped with a 11th Gen Intel(R) Core(TM) i7-11700K CPU @ 3.60GHz and 64 GB RAM, running Windows 11 with access to an NVIDIA 3090Ti GPU. The version of Matlab is 2020b.

[1]The dataset includes 32 time series.
[2]The FLOSS implementation used in this study is available at `http://www.cs.ucr.edu/~eamonn/FLOSS/`
[3]The TSKMeans implementation can be found at `https://tslearn.readthedocs.io`
[4]The KShape implementation can be found at the same site with TSKMeans

3.2 Dowmsampling

We need to downsample some large datasets in order to obtain results in a reasonable time. To be specific, PAMAP2 is of length 255K$\sim$408K and USC-HAD is of length 25.4K$\sim$56.3K. We thus conduct $2\times$ and $4\times$ downsampling on PAMAP2 and USC-HAD, respectively, in order to speed up the experiments. Note that The other datasets have not been downsampled. The results reported in the original paper of Time2State come from raw data that has not been downsampled.

In order to use *dtw* as the distance measure for clustering on USC-HAD, Synthetic, and ActRecTut, we conducted $5\times$ downsampling on these datasets again. These $5\times$ downsampled datasets are only used to evaluate FLOSS+TSKMeans *when using* dtw *as the distance measure*, because otherwise, we cannot obtain a result even after a long time (e.g., 24 hours).

3.3 Hyperparameter Optimization

3.3.1 FLOSS

The number of segments (denoted as N), and the number of states (denoted as K) are specified according to the ground truth. Step size is set to 3 according to the default setting in the FLOSS code. *slWindow* (*subsequenceLength*) is set to 100, 80, 50, 40, 16 for MoCap, ActRecTut, Synthetic, USC-HAD and PAMAP2. respectively. For PAMAP2, the subsequence length is set to 65 in the FLOSS paper (Section 4.8), so we set *slWindow* to 65/4$\approx$16 on $4\times$ downsampled PAMAP2. Each time series in the same dataset shares the same set of hyperparameters as these data come from the same domain.

The parameters on the UCR-SEG dataset are the same as the original code without any changes. The hyperparameters are specified for each time series respectively.

It is worth mentioning that there might be a bug in the FLOSS source code that prevented us from searching for a better window size on these datasets. The details of the bug are described in Section 3.5.

3.3.2 Time2State

Time2State does not need to specify K and N. Nevertheless, it requires two parameters, namely *window size* (denoted as *w*) and *step size* (denoted as *s*). For each dataset, we adjust the *w* and *s*, following the method introduced in Wang et al. (2023).

3.4 Metrics

We use Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) as the evaluation metrics, which are used for evaluating clustering results. ARI and NMI are calculated on the labels of each time point. These metrics intuitively show the matching degree and state assignment performance.

3.5 Bug Reoprt

When the value of *slWindow* exceeds a certain threshold (such as 100 on the USC-HAD dataset), FLOSS generates an error that originates from the *findLocalMinimums* function. This issue is consistently reproducible and occurs on the ActRecTut, Synthetic, and USC-HAD datasets. This error has prevented us from further exploring the possibility of improving performance on these datasets by adjusting the *slWindow* to large values.

3.6 Results

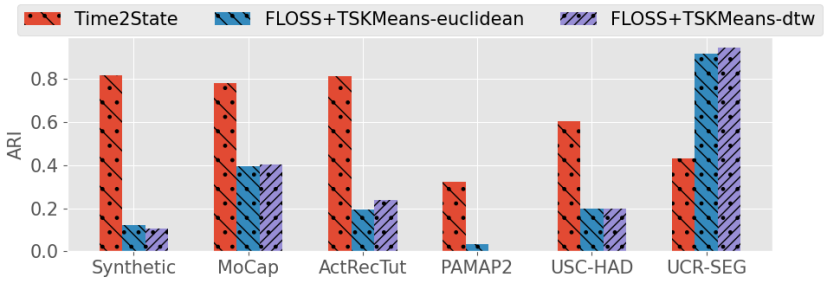
3.6.1 State Detection Results

We implemented a visualization tool for visualizing all results of FLOSS and Time2State. Part of the visualization results are displayed in A.1. The visualization results of all time series for FLOSS and Time2State are saved in *.png* format in *figs/* and *figs_Time2State/*, respectively. The quantitative results are shown in Fig. 3. Note that we cannot obtain the results for FLOSS+TSKMeans using DTW as the distance measure for PAMAP2 dataset due to the long running time. Even if we downsampled PAMAP2 by $10\times$, TSKMeans-dtw did not return any result after 24 hours.

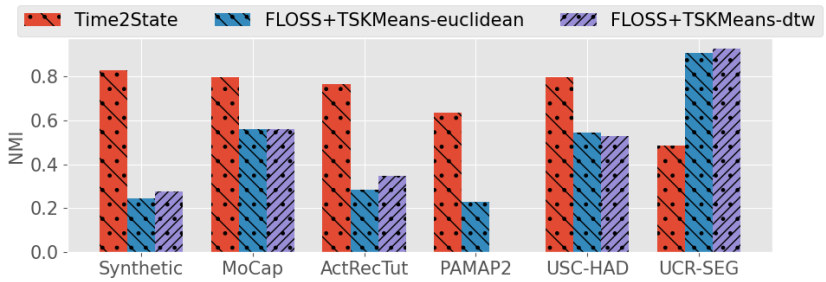
Time2State The performance of Time2State is consistent with that in Wang et al. (2023). It is worth noting that the experiment is only conducted once here; the result reported in Wang et al. (2023) is the average of ten independent executions.

FLOSS+TSKMeans For UCR-SEG dataset, FLOSS performs extremely well. This is because for this dataset, $N \approx K$ (Note that K and N are specified according to the ground truth). In this case, each segment has a (roughly) unique state. Our test shows that using *dtw* or *euclidean* for clustering does not cause any performance differences on UCR-SEG.

While FLOSS performs well on UCR-SEG, its performance on other datasets is not satisfactory. This might be due to the inappropriate setting of *slWindow*, but the bug reported above prevents us from searching for a more extensive range of



(a) ARI



(b) NMI

Figure 3: Performances of (Augmented) FLOSS and Time2State

slWindow values. Another possible reason may be related to section 4.3 of the FLOSS paper, which involves adding a temporal constraint. However, we were unable to locate where to specify the temporal constraint (TC) parameter in the implementation code (*Time_series_Self_Join_Fast.m*). It appears that TC has not been exposed through the FLOSS API.

In conclusion, FLOSS has the potential for further improvement, but it is necessary to address the bug we reported in Section 3.5 and provide a way to adjust the TC parameter. We would greatly appreciate the authors of FLOSS's help in fully realizing FLOSS's potential.

3.6.2 Results in the Presence of Outliers

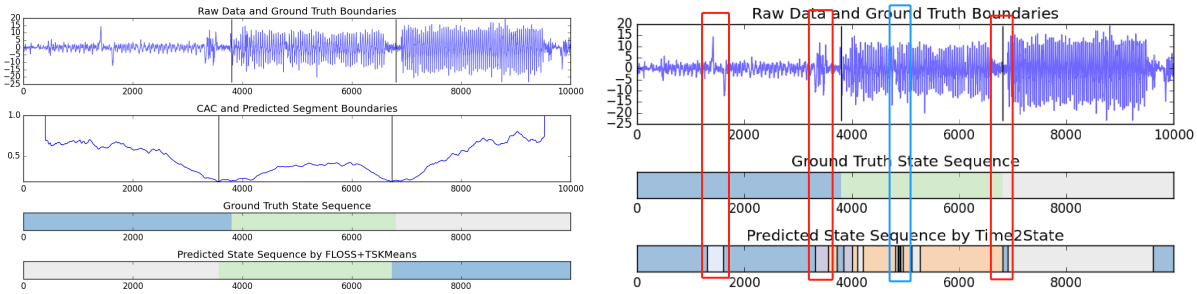
We further investigate the two methods in the presence of some outliers in the dataset. Time2State is more sensitive to outliers. As a case study shown in Fig. 4, Time2State is affected by some outliers (visually) and outputs the subsequences near these outliers as new states (the part framed in red boxes). FLOSS, in contrast, is more resilient to the outliers. This resilience comes with the help of its input parameter given in advance, i.e., the number of segments.

The sensitivity to outliers is a double-edged sword: on the one hand, it means the capability of detecting rare states, e.g., the states marked in red boxes; on the other hand, it means more work on checking some unwanted state switches, e.g., the state marked by the blue box.

4 Conclusion and Discussion

Although both FLOSS and Time2State can identify boundary points between semantic segments, their initial design goals and solution methodologies make them distinct. FLOSS is specifically designed for detecting segment boundaries, but it relies on humans to assign a semantic label (a.k.a. state) to each segment. Time2State is designed for automatic state detection and, in the meantime, returns the boundaries between segments. Thus, to make the two methods comparable, we must augment FLOSS by clustering the segments found by FLOSS into states.

Another subtle but critical difference between the two methods is their input parameters. Augmented FLOSS requires the input of N and K to work effectively, while Time2State does not need these parameters. This difference makes them suitable for different scenarios. Augmented FLOSS is more suitable for homogeneous datasets where the values of N and K apply to each time series within the dataset and can be predetermined. On the other hand, Time2State is better suited for heterogeneous datasets where each time series may have a different number of segments and states, making a method that doesn't require the definition of N and K more desirable.



(a) FLOSS (b) Time2State

Figure 4: Results of FLOSS and Time2Series in the presence of outliers. This time series is from the UCR-SEG dataset, namely, *WalkJogRun2*. On the positive side, Time2State outputs the subsequences near the outliers as new states (the parts framed in red boxes). On the negative side, Time2state outputs more boundaries when there are some subtle changes in the raw data, resulting in an unwanted state switch (the part framed in the blue box).

References

Shaghayegh Gharghabi, Chin-Chia Michael Yeh, Yifei Ding, Wei Ding, Paul Hibbing, Samuel LaMunion, Andrew Kaplan, Scott E Crouter, and Eamonn Keogh. 2019. Domain agnostic online semantic segmentation for multi-dimensional time series. *Data mining and knowledge discovery* 33 (2019), 96–130.

Patrick Schäfer, Arik Ermshaus, and Ulf Leser. 2021. ClaSP - Time Series Segmentation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 1578–1587.

C.Y. Wang, K. Wu, T.Q. Zhou, and Z.P. Cai. June 2023. Time2State: An Unsupervised Framework for Inferring the Latent States in Time Series Data. In *ACM SIGMOD*.

A Appendix

A.1 Result Visualization

The visualization results of FLOSS and Time2State are as follows. We selected the first time series from each dataset. The results of all time series for FLOSS and Time2State are saved in *figs/* and *figs_Time2State/*, respectively.

A.1.1 Results on MoCap

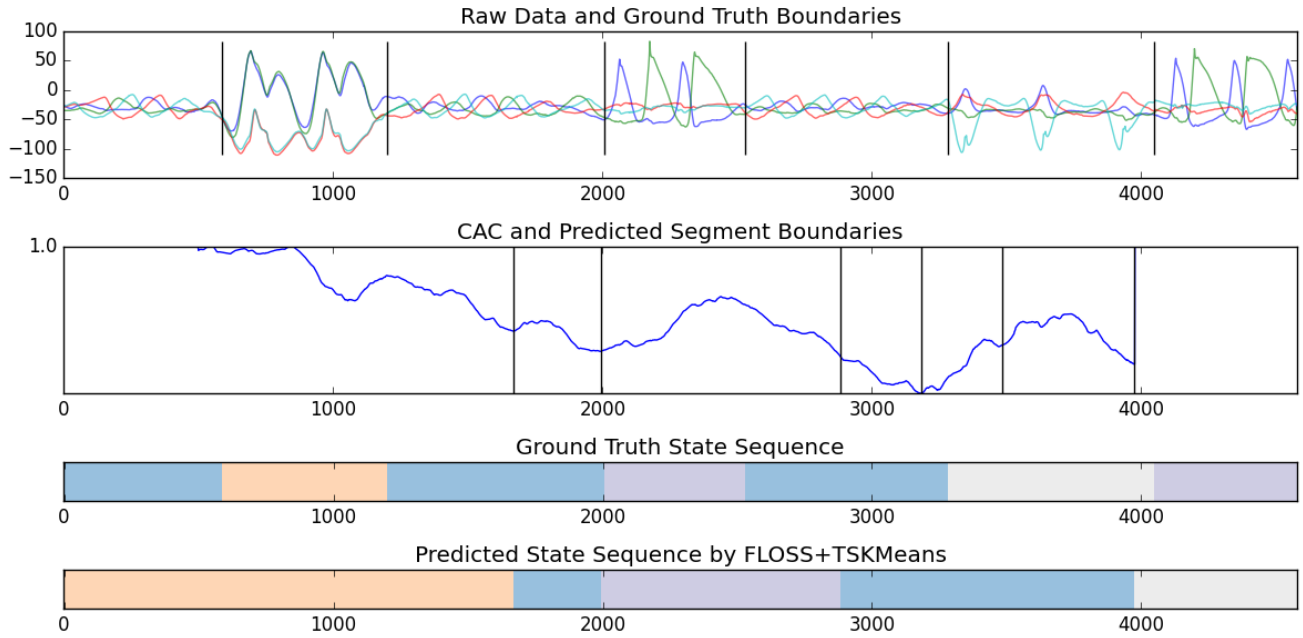


Figure 5: FLOSS+TSKMeans-dtw on MoCap.

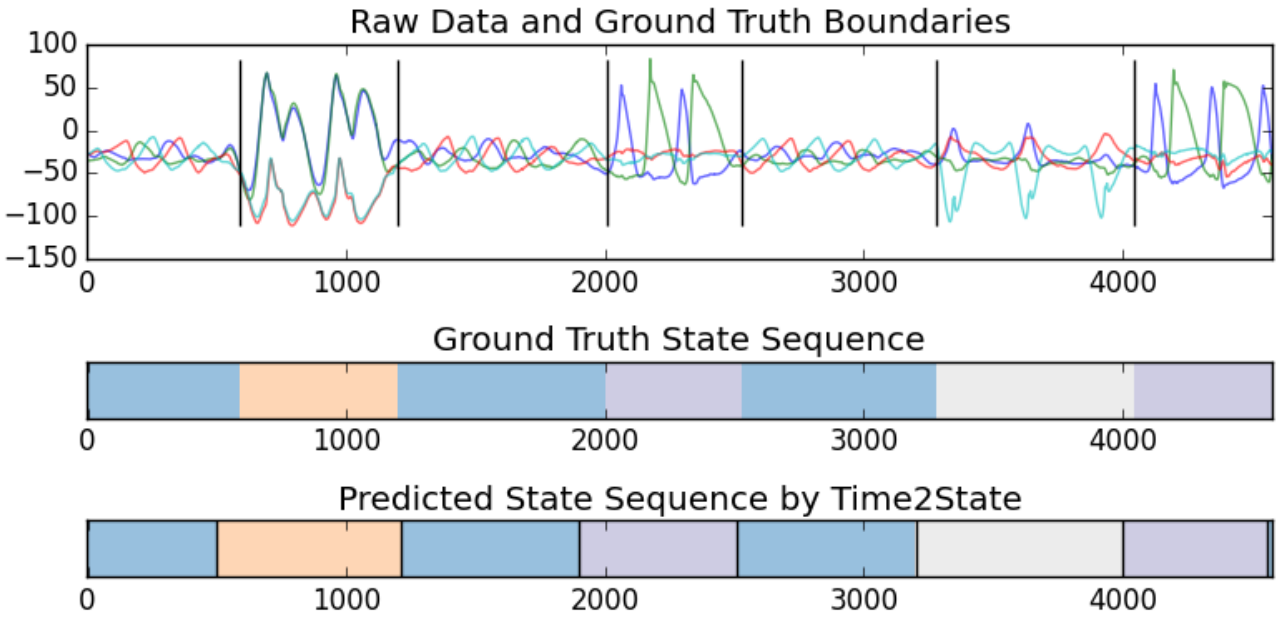


Figure 6: Time2State on MoCap.

A.1.2 Results on Synthetic

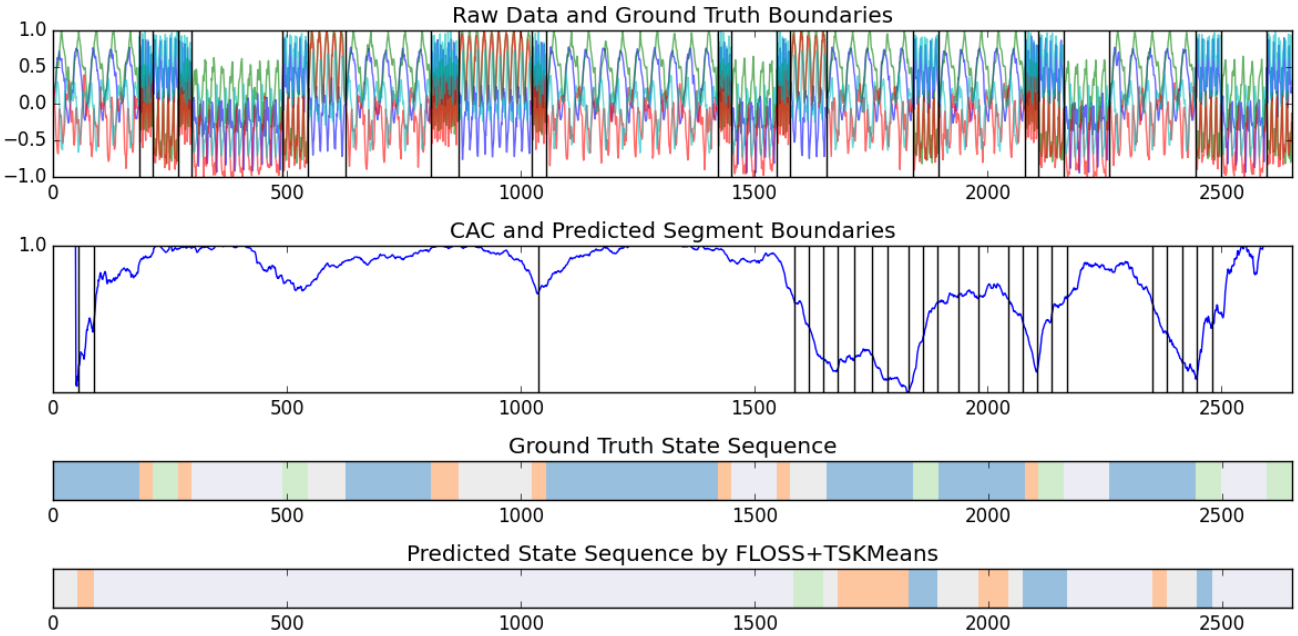


Figure 7: FLOSS+TSKMeans-dtw on Synthetic.

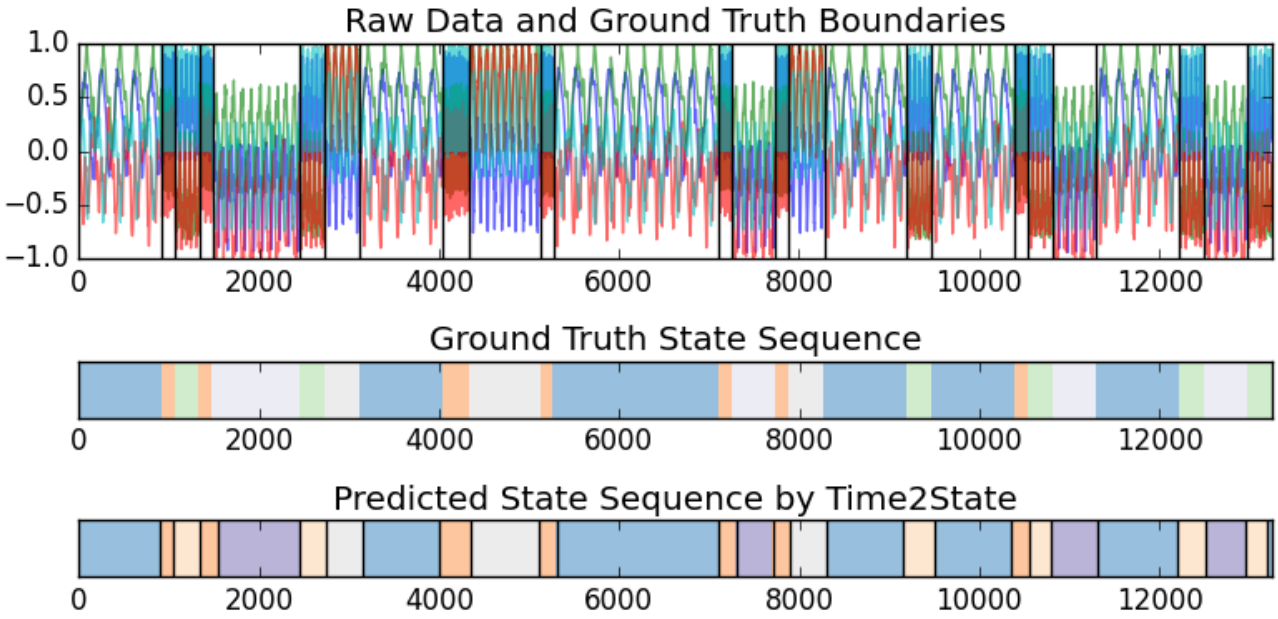


Figure 8: Time2State on Synthetic.

A.1.3 Results on ActRecTut

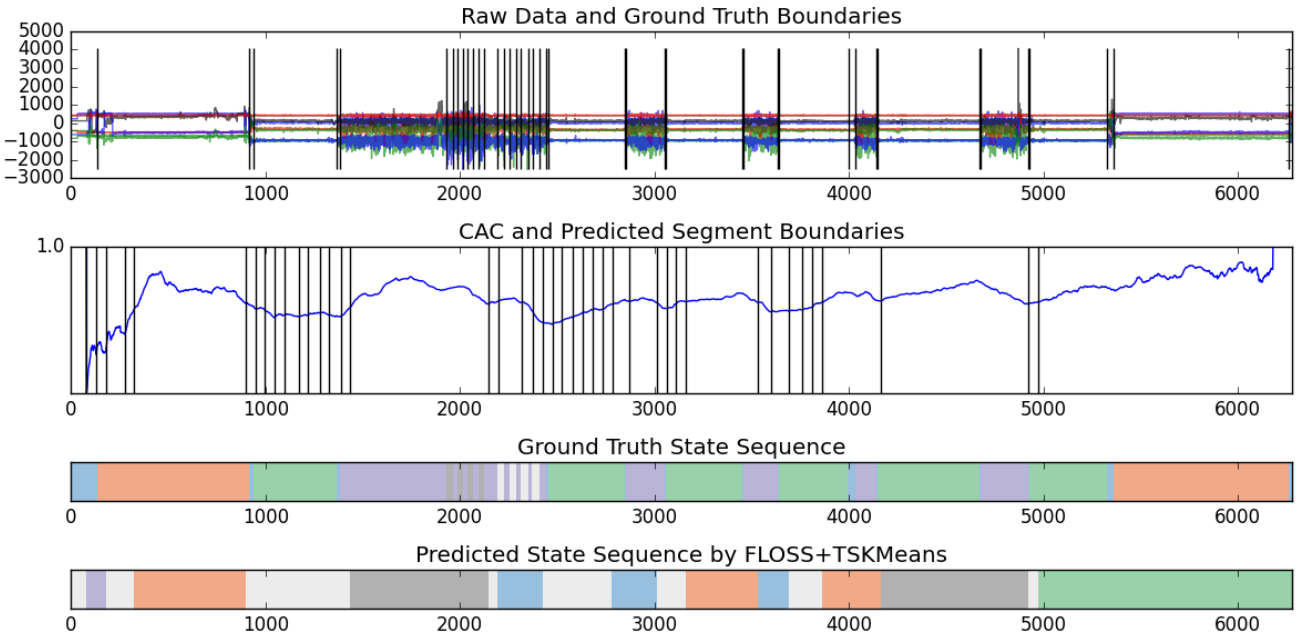


Figure 9: FLOSS+TSKMeans-dtw on ActRecTut.

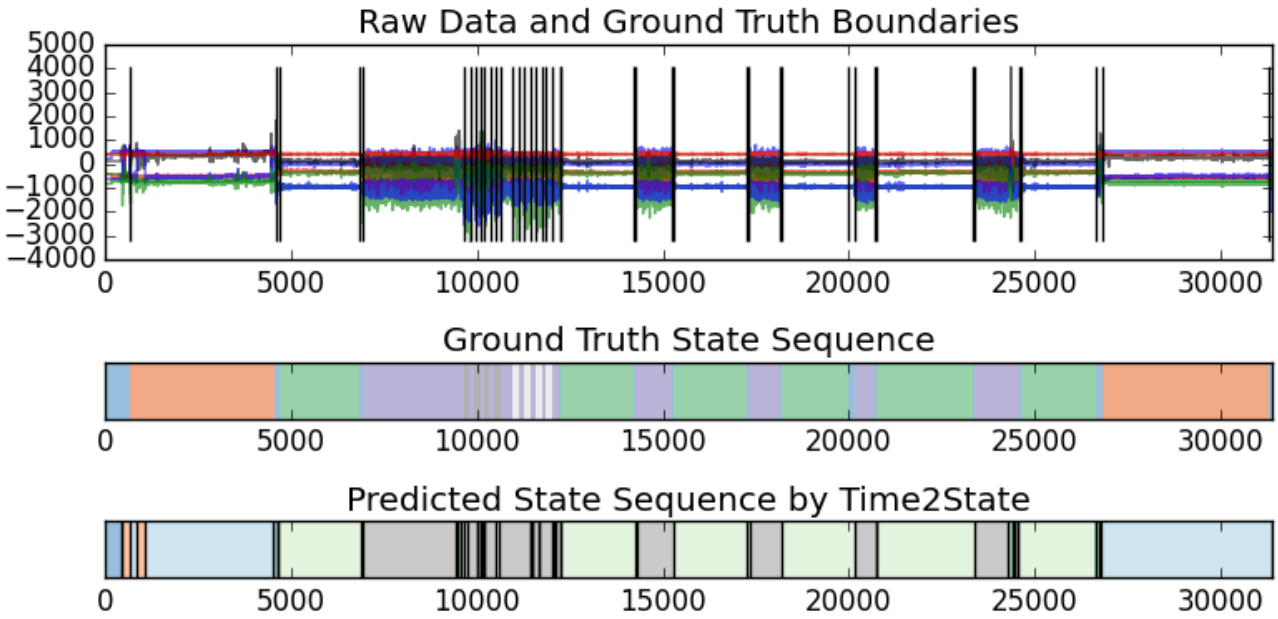


Figure 10: Time2State on ActRecTut.

A.1.4 Results on PAMAP2

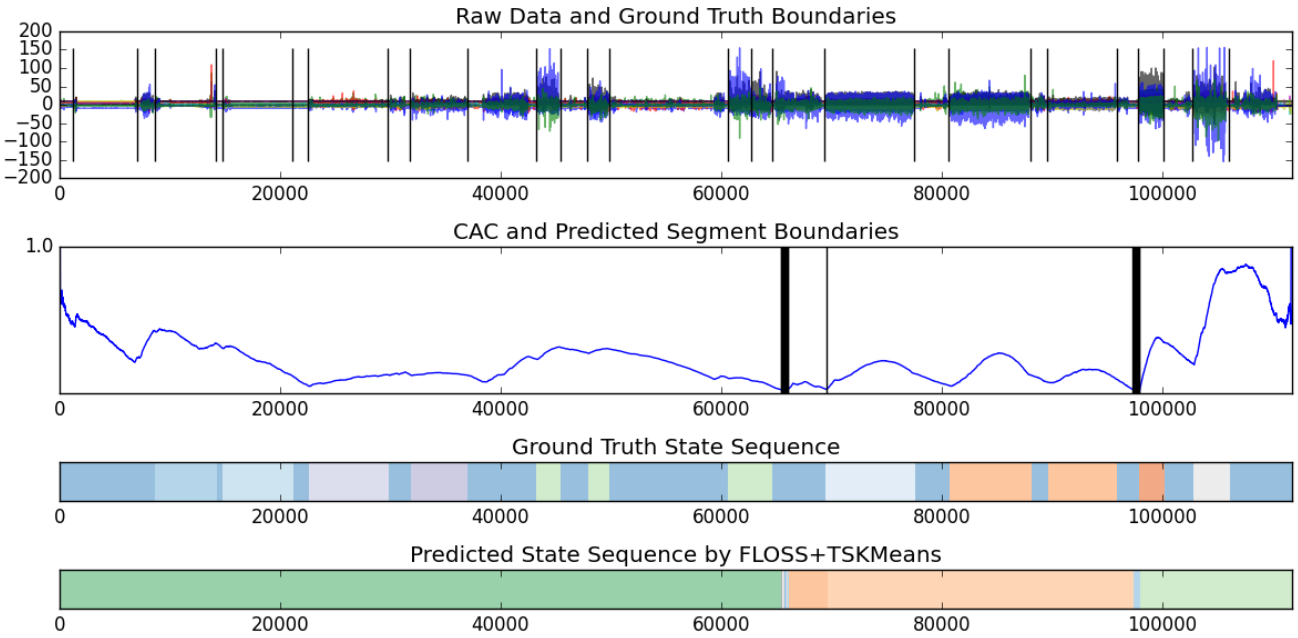


Figure 11: FLOSS+TSKMeans-euclidean on PAMAP2.

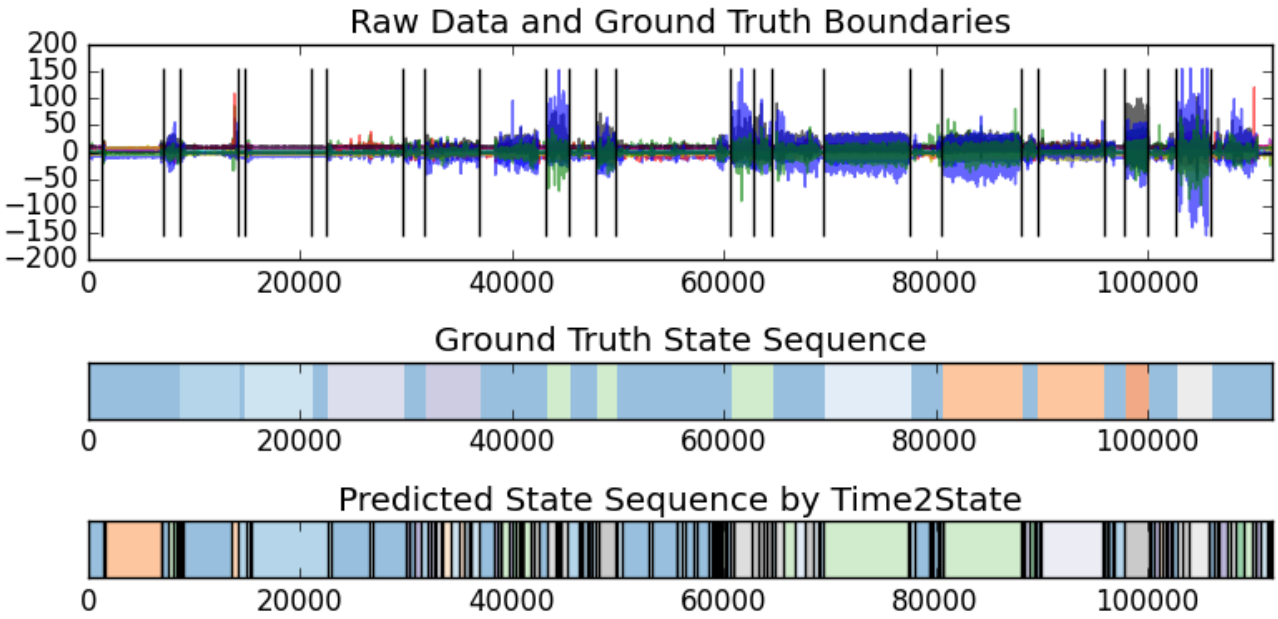


Figure 12: Time2State on PAMAP2.

A.1.5 Results on USC-HAD

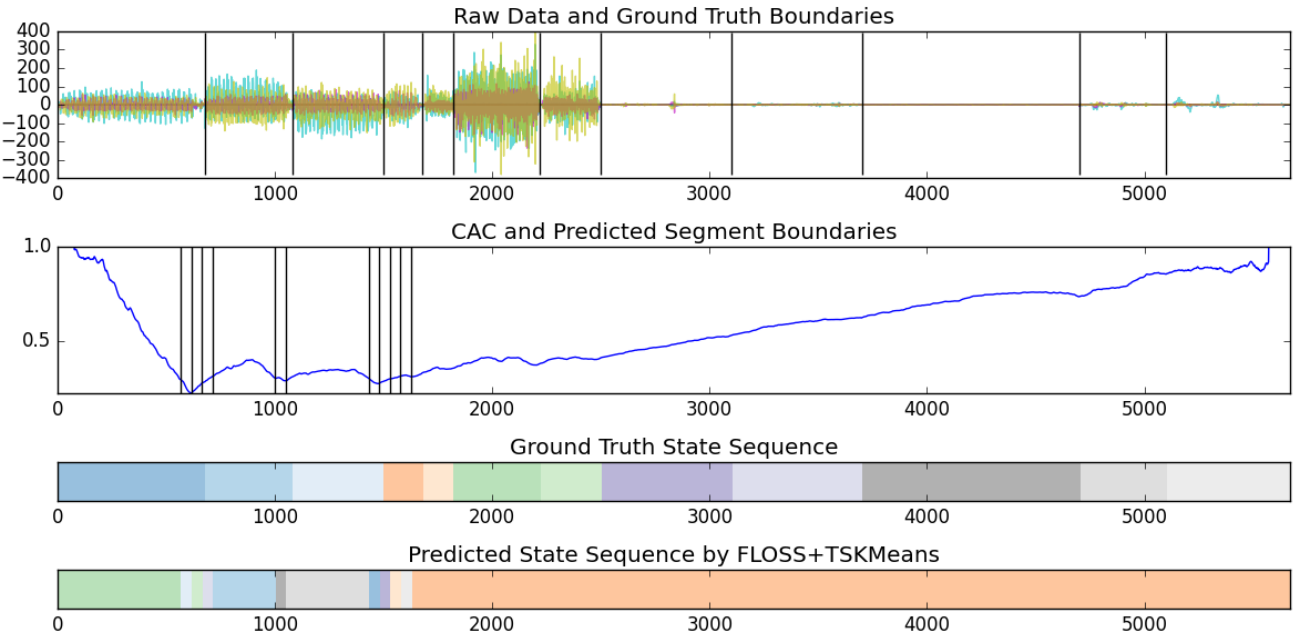


Figure 13: FLOSS+TSKMeans-dtw on USC-HAD.

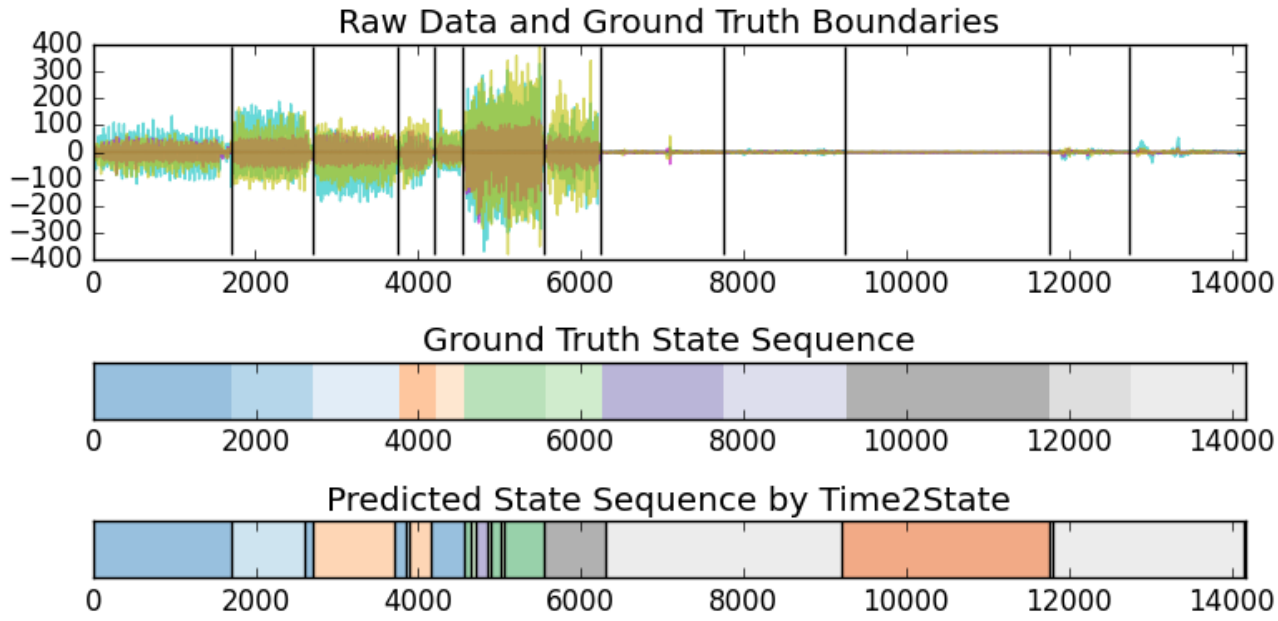


Figure 14: Time2State on USC-HAD.

A.1.6 Results on UCR-SEG

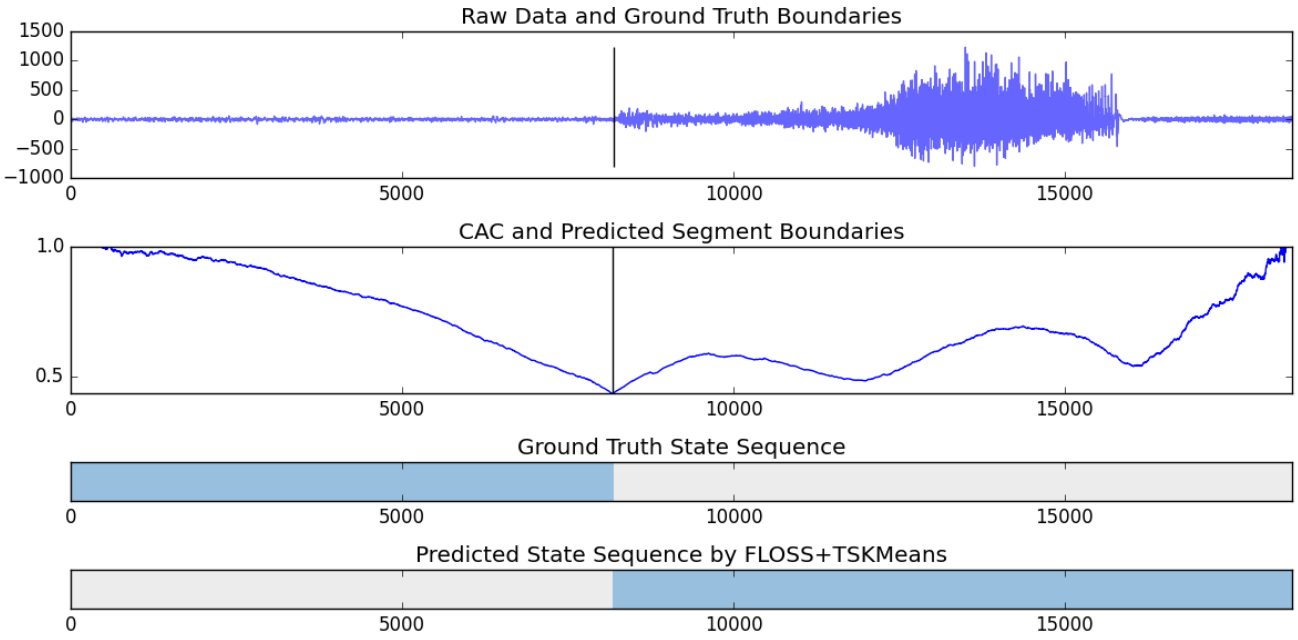


Figure 15: FLOSS on UCR-SEG.

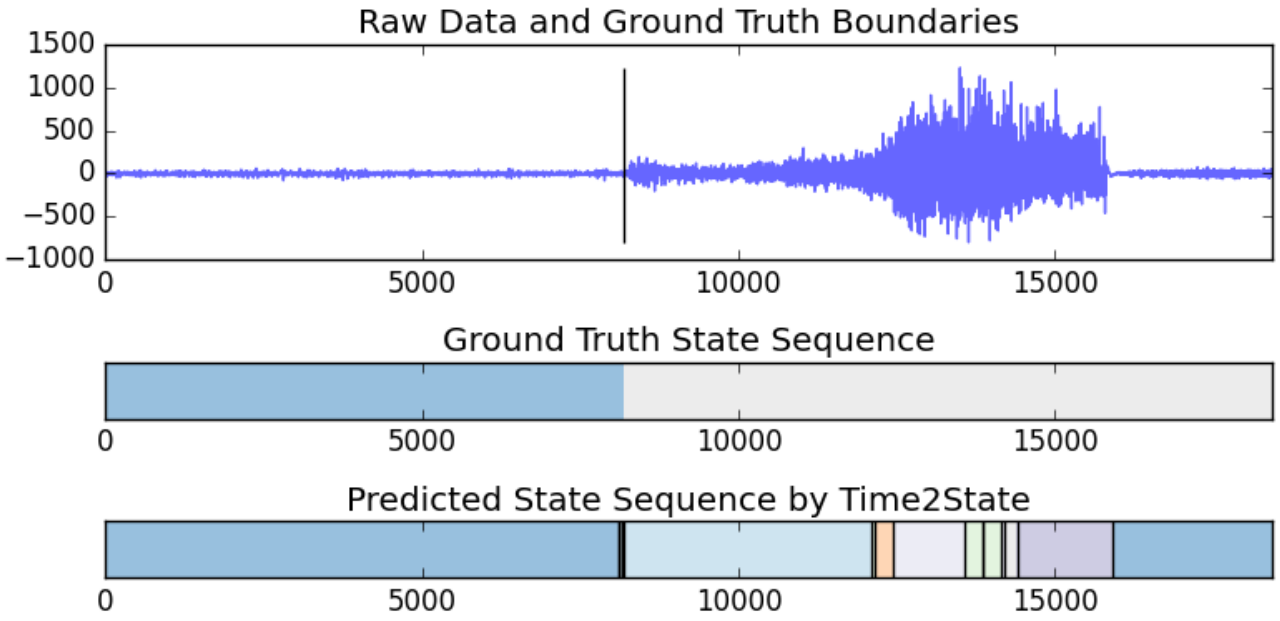


Figure 16: Time2State on UCR-SEG.

B Error Information

B.1 Error Infromation of TSKMeans with 'dtw' as distance measure

```
Traceback (most recent call last):
File "c:\Users\USER\Desktop\Compare_With_FLOSS\cluster_segs_of_FLOSS.py", line 38, in <module>
cluster_on(dataset)
File "c:\Users\USER\Desktop\Compare_With_FLOSS\cluster_segs_of_FLOSS.py", line 32, in cluster_on
clustering_result = cluster_segs(data, found_cps=cps_json[file_name], n_states=num_states)
File "c:\Users\USER\Desktop\Compare_With_FLOSS\miniutils.py", line 73, in cluster_segs
ts_kmeans = TimeSeriesKMeans(n_clusters=n_states,metric='dtw').fit(segments)
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\clustering\kmeans.py", line 780, in fit
self._fit_one_init(X_, x_squared_norms, rs)
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\clustering\kmeans.py", line 665, in _fit_one_init
self._update_centroids(X)
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\clustering\kmeans.py", line 713,in _update_centroids
self.cluster_centers_[k] = dtw_barycenter_averaging(
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\barycenters\dba.py", line 498, in dtw_barycenter_averaging
bary, loss = dtw_barycenter_averaging_one_init(
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\barycenters\dba.py", line 591, in dtw_barycenter_averaging_one_init
diag_sum_v_k, list_w_k = _mm_valence_warping(list_p_k, barycenter_size,
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\barycenters\dba.py", line 303, in _mm_valence_warping
list_v_k, list_w_k = _subgradient_valence_warping(
File "D:\Anaconda3\envs\ml\lib\site-packages\tslearn\barycenters\dba.py", line 260, in _subgradient_valence_warping
w_k = numpy.zeros((barycenter_size, sz_k))
numpy.core._exceptions.MemoryError: Unable to allocate 1.82 GiB for an array with shape (15650, 15650) and data type float64
```

B.2 Error Information of KShape

```
Resumed because of empty cluster
```