

Online Unsupervised State Recognition in Sensor Data

Julien Eberle, Tri Kurniawan Wijaya, and Karl Aberer

School of Computer and Communication Sciences

École Polytechnique Fédérale de Lausanne (EPFL)

CH-1015 Lausanne, Switzerland

Email: {julien.eberle, tri-kurniawan.wijaya, karl.aberer}@epfl.ch

Abstract—Smart sensors, such as smart meters or smart phones, are nowadays ubiquitous. To be “smart”, however, they need to process their input data with limited storage and computational resources. In this paper, we convert the stream of sensor data into a stream of symbols, and further, to higher level symbols in such a way that common analytical tasks such as anomaly detection, forecasting or state recognition, can still be carried out on the transformed data with almost no loss of accuracy, and using far fewer resources. We identify states of a monitored system and convert them into symbols (thus, reducing data size), while keeping “interesting” events, such as anomalies or transition between states, as it is. Our algorithm is able to find states of various length in an online and unsupervised way, which is crucial since behavior of the system is not known beforehand. We show the effectiveness of our approach using real-world datasets and various application scenarios.

I. INTRODUCTION

In the near future, the role of sensor networks is expected to become pervasive. Smart home and smart city concepts require sensor deployment in almost every corner of our living space, to make sure that homes and cities are “smart enough” to assist and support the needs of their inhabitants. In addition, the ever increasing capabilities of smart phones to record and sense raises the promise of more personalized and contextually relevant services. Given the numerous built-in sensors in today’s smart phones (accelerometer, GPS, microphone), and their access to their owner’s calendar, to-do list, SMS/call history and internet it is not hard to imagine that in the near future they will be capable of completely analyzing our daily schedule and giving specifically tailored and highly personalized recommendations. These could include what clothes to wear (depending on predicted activities and weather forecast), which route to take when to avoid traffic jams, and which items to buy at the supermarket (depending on food preferences or whether one plans to have guests). Owing to the rapidly changing human behavior, then unsupervised learning is more appropriate compared to supervised learning, whose ability is often limited by its training context and environment.

Another interesting scenario for pervasive sensors and computation are smart electronic appliances. These appliances are equipped with sensors to measure their power use (including patterns) and some (limited) storage, computational and communication capabilities. Here also a generic and unsupervised algorithm is more suited due to the huge amount of newly introduced appliances and sensors [1], [2]. Additionally, their ability to predict their own future states or usage patterns could be useful, e.g., for participating in a demand response program [3], or if electricity prices follow real-time dynamic pricing schemes, then drawing/storing energy from the grid when the price is cheap and using it later when the price is expensive

would save money [4], [5]. Energy consumption measurement at the level of electronic appliances also enables us to detect their different states. Owners can be notified if an unusual operation or anomaly is detected, e.g. unusually long *on* states detected on an appliance could mean that the owner forgot to turn the appliance off. Detecting this unintended event leads to energy and cost savings. Another example is an atypical power consumption pattern, which could mean that the device requires maintenance or indicate an imminent failure.

However, the data generated by all these sensors (from buildings, public infrastructure, smart phones, smart electronic appliances), is big data in every dimension: big volume, big velocity and big variety. A centralized approach typically involves communicating data from sensors to a central server for processing. Most of the today’s popular location-based services are examples of centralized solutions, i.e. GPS information from a mobile phone is communicated to a central server, computation for the service requested is performed in that server and the result is sent back to the phone. Similar processes are envisioned for participatory sensing scenarios. Additionally, since sensor data contain sensitive personal information, this solution is susceptible to breach of privacy. It also requires vast data storage and significant computation and communication costs. With an increasing number of sensors, scalability becomes a serious issue. Furthermore, the high cost of large storage requirements could potentially cause sensor data processing to be dominated by a few big players (Big Brothers holding most of our data).

In this paper, we focus on a decentralized approach where we store and process the data as close to the sensors as possible. Since only a little data (usually aggregated) leaves the sensor or smart phone, privacy is preserved, communication costs are greatly reduced and the whole system is more scalable. However, the advantages of a decentralized approach require that other challenges be solved: smart devices (such as smart appliances and smart phones) only have limited storage and computational power. Solving these challenges means processing big (streaming) data as efficiently as possible while maintaining data usability. To this end, the solution should satisfy the following requirements: 1) For more efficient processing, streaming sensor data should be processed online (efficiently, some limited delay can be tolerated); 2) For more efficient storage, data can be transformed into another representation. However, it should allow efficient processing of data mining algorithms, without “reverting” the representation; Encoding can be lossy, but the states must be preserved as much as possible; 3) Since a sensor is able to measure different things, the solution should be able to handle multidimensional time series; 4) Since not all states could be observed apriori,

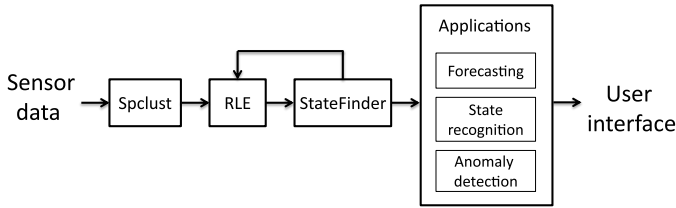


Fig. 1: An overview of our solution. We start with the initial symbolization process (Splust, see Section II). Next, we repeatedly compress the representation using run-length encoding (RLE) and recognize states using StateFinder (see Section III). The output is then used in various applications (see Section IV and V).

the solution should be able to learn in an unsupervised way. 5) Since the length of states might vary, the solution should also be able to find states of various length. Thus, algorithms that assume a fixed/predefined length of states (or sequence or sliding window) might not be suitable.

Hitherto, there is no solution in the literature which fulfills all the requirements above (see our literature review in Section VI). This paper fills the gap and proposes a framework which satisfies them (see Figure 1). We summarize the key contributions of this paper as follows.

- We propose a novel *state preservation index* which quantifies how good a cluster/symbolic configuration is (given the measurements) in preserving system’s states (Section II).
- We propose a novel *StateFinder* algorithm which combines the effectiveness of some well-known techniques, such as Markov chain and subsequences clustering (Section III). The algorithm is able to identify the states along a data stream in an online and unsupervised way, using a limited memory and computational footprint.
- We carry out experiments using real-world datasets from different domains, e.g., human energy consumption and activity recognition, and present different application scenarios, e.g., state recognition, forecasting, and anomaly detection, to showcase the versatility of our approach (Section V).

II. STATE-PRESERVING SYMBOLIC REPRESENTATION

In this section, we outline an initial step needed to deal with noisy n -dimensional time series.

Definition 1: We define n -dimensional time series formally as $S = \{s_1, s_2, s_3, \dots\}$. Each $s_i \in S$ is a tuple (t_i, \vec{V}_i) , where t_i is a timestamp and $\vec{V}_i \in \mathbb{R}^n$ is a vector of measurements. We have $i < j$ iff t_i is earlier than t_j .

Next, we explain how we learn to map measurement vectors (or *measurements*) onto symbols to reduce data granularity and process it more efficiently, while preserving system’s states as much as possible. This process can also be viewed as a preprocessing step (see Figure 1). The learning algorithm below, *Splust*, can be run either during the sensor calibration phase or on top of historical data that capture the measurement distribution. As similar measurement vectors are converted to the same symbol, the symbolization process can also be thought as a clustering process, where measurement vectors are replaced by their cluster labels.

A. State Preservation Index

Measurements which are more likely to belong to the same state, should be converted into the same symbols. Thus, we consider the following two principles when developing

our symbolic representation: (i) measurements with similar values are more likely to belong to the same state, and (ii) a system is more likely to stay in its current state than change to another state. Let $C = \{c_1, \dots, c_{|C|}\}$ be a cluster configuration (a set of clusters), where each cluster $c \in C$ is an n -polytope. Since a measurement \vec{V} is essentially a point in an n -dimensional space, we write $\vec{V} \in c$ if the point \vec{V} lies in the n -polytope c . We express the existence of a connection between clusters c_1 and c_2 through s_i as a binary function:

$$is_conn(c_1, c_2, s_i) = \begin{cases} 1 & \text{if } \vec{V}_i \in c_1 \wedge \vec{V}_{i+1} \in c_2 \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

and its distance as:

$$dist_conn(c_1, c_2, s_i) = \begin{cases} dist(\vec{V}_i, \vec{V}_{i+1}), & \text{if } \vec{V}_i \in c_1 \wedge \vec{V}_{i+1} \in c_2 \\ 0, & \text{otherwise} \end{cases}. \quad (2)$$

In our experiments (see Section V), we use the Euclidean distance. However, any distance metrics for two vectors could also be used here.

We define the number of connections between c_1 and c_2 as $\#conn(c_1, c_2) = \sum_{s_i \in S} is_conn(c_1, c_2, s_i)$. And, for a cluster c , we define its diameter, $diam(c)$, as the farthest distance between any two measurements in c . We measure how *compact* the clusters in configuration C are by:

$$intra_coeff(C) = \sum_{c \in C} \frac{\#conn(c, c)}{diam(c)}, \quad (3)$$

i.e., the higher the *intra_coeff* value, the more compact the clusters are. Note that, the division by the cluster diameter is needed, so that the measure is cluster-size invariant.

We define the average connection length between two clusters, c_1 and c_2 , as:

$$avgConnLen(c_1, c_2) = \frac{\sum_{s_i \in S} dist_conn(c_1, c_2, s_i)}{\#conn(c_1, c_2)}.$$

Then, we measure the connection density between clusters in configuration C as:

$$inter_coeff(C) = \sum_{c_1 \in C} \left[\sum_{\substack{c_2 \in C, \\ c_2 \neq c_1}} \frac{\#conn(c_1, c_2)}{avgConnLen(c_1, c_2)} \right], \quad (4)$$

i.e., the fewer the connections between clusters and the farther the connections between them, the lower the *inter_coeff* value. As we prefer to have clusters which are separated as far as they possibly could, and fewer connections between them, lower *inter_coeff* is preferred in general.

Given a cluster configuration C , we design a scoring function, *state preservation index*, which aims to measure how good the configuration is in preserving system’s state transition:

$$spi(C) = \frac{intra_coeff(C)}{inter_coeff(C)}, \quad (5)$$

For a cluster configuration C , the higher its *spi* value, the better. That is, we prefer clusters that are compact, densely intra-connected, well separated and sparsely inter-connected.

B. The Splust Algorithm

In general, finding the best cluster configuration for a time series (which maximizes *spi*) is intractable. In order to approximate it, we propose Splust (see Algorithm 1), which is

Algorithm 1: Spclust

Input: n -dimensional measurement space M , grid size G , threshold T
Output: cluster configuration

- 1 quantize M into n -dimensional hypercubes of length G
- 2 **forall** the hypercubes h **do**
- 3 count measurements in h
- 4 **if** $\text{count} \geq T$ **then**
- 5 $\text{density}(h) \leftarrow 1$
- 6 **else**
- 7 $\text{density}(h) \leftarrow 0$
- 8 find connected component (*clusters*) on all hypercubes h , where $\text{density}(h) = 1$
- 9 assign distinct label to each cluster
- 10 enlarge the clusters incrementally to include hypercubes h with $\text{density}(h) = 0$ until all hypercubes are clustered
- 11 **return** *clusters*

based on a spatial clustering algorithm, Wavecluster [6]. This algorithm detects clusters of arbitrary shapes; it is fast, designed for large databases and insensitive to noise. In contrast to Wavecluster, which uses wavelet transformation, Spclust uses a simple step function transformation (see line 3-7). Then, it expands the resulting clusters to include hypercubes with zero density (see line 10). In Wavecluster, those hypercubes are not clustered since they are translated into noise or white spaces. In our context, however, we need to partition the space of measurements, as we might encounter these values later in the future. To this end, we borrow the idea from maximum margin classifiers to enlarge the resulting clusters incrementally until all hypercubes are clustered. Next, from a set of possible grid size, \mathcal{G} , and threshold, \mathcal{T} , we select $G^* \in \mathcal{G}$ and $T^* \in \mathcal{T}$ that maximize the *spi* value of the resulting configuration. Finally, we transform each measurement in the n -dimensional time series S with its cluster labels and obtain a symbolic time series \hat{S} . We define a symbolic time series formally as follows.

Definition 2: Let \mathcal{A} be our alphabet or a set of symbols. We define a *symbolic time series* as $\hat{S} = \{\hat{s}_1, \hat{s}_2, \hat{s}_3, \dots\}$, where each $\hat{s}_i \in \hat{S}$ is a tuple (t_i, a_i) , t_i is a timestamp, and $a_i \in \mathcal{A}$ is a symbol. Additionally, t_i is earlier than t_j iff $i < j$.

C. Run Length Encoding

To improve storage space and access efficiency, we apply a run length encoding compression algorithm over symbolic time series. As our input time series contains only timestamp-value tuples, we assume that a measurement stays valid until the next one. However, special care was taken, in the experiments, so that no huge gap (sensor down-time) was filled this way.

Definition 3: Let $\{\hat{s}_1, \hat{s}_2, \hat{s}_3, \dots\}$ be a symbolic time series, where each element is a tuple (t_i, a_i) , as defined earlier. The output of RLE is a sequence of $\{\bar{s}_1, \bar{s}_2, \bar{s}_3, \dots\}$, each element being a triple $(t_j^b, t_j^e, \bar{a}_j)$ which is defined as the shortest sequence of triples such that:

$$\text{RLE} : (\mathbb{R}, \mathbb{R})^n \mapsto (\mathbb{R}, \mathbb{R}, \mathbb{R})^m$$
$$\text{RLE}(\{\hat{s}_1, \hat{s}_2, \hat{s}_3, \dots\}) := \{\bar{s}_1, \bar{s}_2, \bar{s}_3, \dots\}$$

where $\forall i \in [1, n], \forall j \in [1, m]$, if $t_i \in [t_j^b, t_j^e]$, then $\bar{a}_j = a_i$.

This lossless process transforms a sequence of symbols into a single triple composed of a starting time t^b , ending time t^e and a symbol. Hereafter, unless stated otherwise, we represent time series as RLE compressed triples.

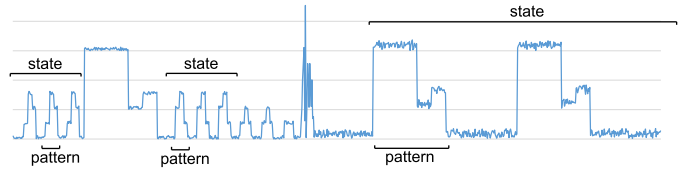


Fig. 2: Illustration: several consecutive occurrences of a pattern indicate that the system is in a certain state.

III. THE STATEFINDER ALGORITHM

We define a *state* as a consecutively repeating pattern of symbols that has several (significant) occurrences along the time series. In previous works, the term *pattern* can be found termed as *primitive shape* [8], *frequent temporal pattern* [9] and *motif* [10], and what we call *states* is more similar to the definition of the *operating modes* [11] or *process states* [12] (illustrated in Figure 2). Finding states at various scale in an online manner, however, is not trivial. Additionally, the complete list of system states might be unknown beforehand. Thus, the solution must find the states in an unsupervised manner. However, current solutions in the literature (see also our literature review in Section VI) aim to find either some predefined patterns or patterns (which can also be extended into states) of a fixed length only.

Our goal here is to find states of various length in an online and unsupervised way, and replace them with new symbols (hence reduce the overall data size, see Figure 3 for an illustration), while allowing various inference to be performed on top of the symbolic-state representation (see application examples in Section IV and V). Additionally, more complex patterns can be found by applying multi-pass StateFinder, e.g., 1st pass of the StateFinder is applied to RLE triples, 2nd pass of the StateFinder is applied to the outcome of the 1st pass, and so on. The StateFinder algorithm consists of two steps: (1) identifying subsequences with repeating pattern, (2) clustering relevant subsequences and assigning a (new) symbol to them. We explain the steps in more detail below.

A. Segmentation

One straightforward option is to take all possible subsequences and cluster them to directly separate the repeating ones from others, seen as noise. However, it has been shown in [13] that this approach does not actually work and produces meaningless output. Therefore, we first need to isolate the potential subsequences that show a regular pattern. Since the patterns that we are interested in are characterized not by their length, but by their dynamics, we use a Markov model as the basis of our approach below.

We use a first-order Markov chain, but higher orders can also be used to identify more complex states. In our case, the use of a first-order Markov chain as model for the underlying process is a low-cost and generic approximation and does not imply that the process itself has such properties. If it is known that the time series' process is of higher order and enough computational resources are available, it is possible to use higher order Markov chains.

Length-frequency set. For each symbol transition, we define an iteratively updated frequency of the next symbol length.

Definition 4: Let \bar{a}_{u-1} and \bar{a}_u be the symbol of the $u-1$ th and the u th entry of the RLE represented input. Additionally, let $\lambda \in [0, 1)$ be a fading factor. For $j \in \mathcal{A}$ and $k \in \mathbb{N}$, we denote the (decaying) *length-frequency* of symbol j with

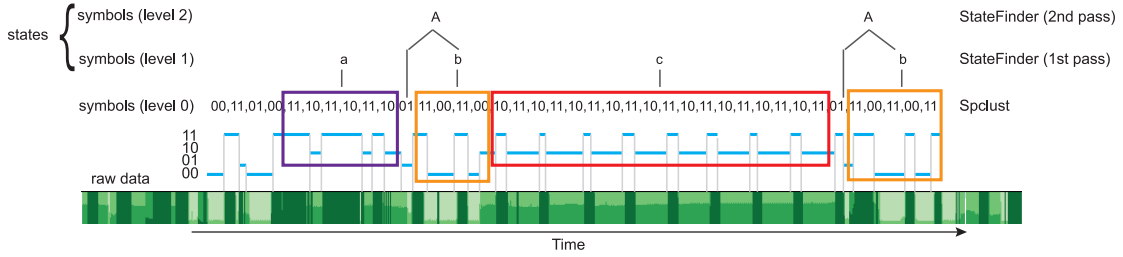


Fig. 3: We apply the Splust algorithm to the raw data (sensor measurements) to produce the symbolic time series. After applying RLE to the symbolic time series (not visible in this figure), we apply StateFinder to find the states. The StateFinder algorithm can be applied several times to build a multi-level state representation. The raw data is displayed using the horizon graph [7].

length k following symbol i , as $h_{i,j}^k$, and update it over time:

$$h_{i,j}^k(0) := 0$$

$$h_{i,j}^k(u) := \begin{cases} h_{i,j}^k(u-1) \cdot \lambda + 1 & , \text{ if } i = \bar{a}_{u-1} \wedge j = \bar{a}_u \wedge \\ & k = t_u^e - t_u^b \\ h_{i,j}^k(u-1) \cdot \lambda & , \text{ if } i = \bar{a}_{u-1} \\ h_{i,j}^k(u-1) & , \text{ otherwise.} \end{cases}$$

Hereafter, we refer to the set of all $h_{i,j}^k$ s as the *length-frequency set*.

In the definition above, we use the fading factor to forget the old entries instead of using the sliding window approach, since the searched patterns could have different length. Thus, setting a predefined window length would be problematic, as too small window would fail to capture long patterns and vice versa, too big window would fail to capture short patterns.

Segments. When the underlying process generating the symbols is in a stationary state, the elements of the length-frequency set tend to be stationary as well, whereas between the states, they may change a lot. In general, the length-frequency set can be used to predict the next symbol (and its length). To give a “smooth” estimation of the likelihood of the u^{th} entry of the RLE triples, we compute $Pr(u)$ the probability that symbol \bar{a}_u of length $t_u^e - t_u^b$ follows \bar{a}_{u-1} using the Gaussian Kernel Density Estimation (KDE). Let $i = \bar{a}_{u-1}$, $j = \bar{a}_u$, and $\mathcal{B}_{i,j}(u) = \{k \mid h_{i,j}^k(u) > 0\}$ be the set of symbol j 's lengths, following symbol i , with non-zero (decaying) frequency. Then, we have:

$$Pr(u) := \frac{\sum_{k \in \mathcal{B}_{i,j}(u-1)} \left(h_{i,j}^k(u-1) \cdot e^{-\frac{((t_u^e - t_u^b) - k)^2}{2 \cdot \sigma_{i,j}(u-1)^2}} \right)}{\sum_{\alpha \in \mathcal{A}} \left[\sum_{k \in \mathcal{B}_{i,\alpha}(u-1)} h_{i,\alpha}^k(u-1) \right]}, \quad (6)$$

where \mathcal{A} is the (symbol) alphabet set and $\sigma_{i,j}(u) = 0.9An^{-1/5}$ is the kernel bandwidth computed using the Silverman's rule of thumb [14]. That is, $n = |\mathcal{B}_{i,j}(u)|$ and $A = \min(sd, \frac{IQR}{1.34})$, where sd and IQR are the standard deviation and the interquartile range of the set $\{h_{i,j}^k(u) \mid k \in \mathcal{B}_{i,j}(u)\}$, respectively. The gaussian term in the nominator is useful to smoothen the symbol length estimation, while the denominator normalizes it over all symbol (and symbol length) that occur after \bar{a}_{u-1} . After computing $Pr(u)$, we define the prediction error as:

$$\mathcal{E}(u) := 1 - Pr(u). \quad (7)$$

Then, a *segment* is a predictable subsequence of RLE triples:

Definition 5: A subsequence of m contiguous RLE triples starting at the u^{th} entry is a *segment*, denoted as $\tilde{S}(u, m) = \{(t_j^b, t_j^e, \bar{a}_j)\}$, iff $t_j^e = t_{j+1}^b$ and $\mathcal{E}(j) < p$ for all $u \leq j \leq u + m - 1$, where p is a threshold parameter. A *maximum segment*

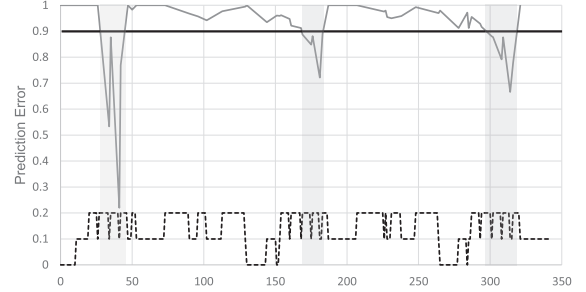


Fig. 4: Illustration on how segments are formed, using the data from the birds' calls dataset in [15]. The dashed line below shows the (three) symbols level 0, plotted as values 0.0, 0.1, and 0.2, the thin solid line is the prediction error over time, and the thick solid line is the threshold parameter $p = 0.9$. The grey areas shows the segments.

is a segment that cannot be lengthen by adding the preceding or following symbols. By definition, maximum segments are non-overlapping.

Hereafter, whenever the context is clear, we omit the parameters u and m and refer to a segment simply as \tilde{S} . Figure 4 shows an example on how segments are formed.

Complexity. To analyze the time complexity of processing each RLE triple, we need to consider both, the time to update the length-frequency set (Definition 4) and the time to compute the prediction error (Eq. 7). Both of them iterate over the set of non-zero length-frequencies, namely, at the u^{th} entry, we iterate over the set $\mathcal{H}_{\bar{a}_{u-1}}(u) = \{h_{i,j}^k(u) \mid k \in \mathcal{B}_{i,j}(u), i = \bar{a}_{u-1}\}$ for $j \in \mathcal{A}$ and $k \in \mathbb{N}$. We refer to this set as \mathcal{H} for brevity. This gives us $\mathcal{O}(|\mathcal{H}|)$ time complexity. Below, we show that $|\mathcal{H}|$ is bounded.

Lemma 1: Let λ be the fading factor in updating the length-frequency set and ϵ be the smallest representable positive value greater than zero (either predefined or limited by machine precision), i.e., for a value v where $0 \leq v \leq \epsilon$, we have $v = 0$. Then, $|\mathcal{H}| \leq \log_{\lambda} \epsilon$.

Proof: Since for a given (StateFinder-) pass the size of the alphabet set is constant, \mathcal{H} can only grow (linearly) when we process an RLE triple by adding a new value for k that has never seen before for a given transition. In the worst case scenario, each incoming triple has a new length k . If this happens, however, the value of the oldest element in \mathcal{H} would be $\lambda^{|\mathcal{H}|}$, since we apply a fading factor when updating the length-frequency set. That is, when $|\mathcal{H}| > \log_{\lambda} \epsilon$, the oldest value would be less than ϵ , and thus keeping $|\mathcal{H}| \leq \log_{\lambda} \epsilon$. ■

Thus, we have $\mathcal{O}(\log_{\lambda} \epsilon)$ time complexity, which is essentially $\mathcal{O}(1)$, since ϵ is a constant. Regarding the space complexity, at any u^{th} entry, following Lemma 1, we need to store $|\mathcal{H}_i(u)|$ entries for each $i \in \mathcal{A}$. Thus, we have space complexity $\mathcal{O}(|\mathcal{A}| \cdot \log_{\lambda} \epsilon)$, which is also $\mathcal{O}(1)$.

B. Clustering

Following the definitions of segments, we represent a segment as a transition matrix of its symbols, i.e., the Segment Transition Matrix.

Definition 6: The *Segment Transition Matrix* of segment \tilde{S} is an $n \times n$ matrix $M(\tilde{S}) = \{m_{i,j}(\tilde{S})\}$, where $n = |\mathcal{A}|$ is the size of the alphabet and $m_{i,j}(\tilde{S})$ is defined as follows, using x as a temporary variable for better readability:

$$x_{i,j}(\tilde{S}) := \sum_{(t_k^b, t_k^e, \bar{a}_k) \in \tilde{S}} \begin{cases} t_k^e - t_k^b - 1 & , \text{ if } i = j = \bar{a}_k \\ 1 & , \text{ if } i = \bar{a}_k \wedge j = \bar{a}_{k+1} \\ 0 & , \text{ otherwise} \end{cases}$$

$$m_{i,j}(\tilde{S}) := \frac{x_{i,j}(\tilde{S})}{\sum_{q=0}^n x_{i,q}(\tilde{S})}$$

The row i , column j of the resulting matrix contains the probability to have a transition from symbol i to j , the diagonal being the probability to stay on that symbol. As our notation excludes the ending time (t_k^e) from the interval, we need to subtract 1 in the top case.

To select which maximum segments are relevant and will be replaced by new symbols, we group them into clusters. The clustering algorithm should: (1) be able to run online (as in the segmentation step), thus limiting its computational needs, (2) make no assumption in the number of clusters in advance, as it may increase over time, (3) be consistent, i.e., once a segment is assigned to a cluster, it should not change its assignation in future iterations. To satisfy all those requirements above, we adapt the clustering algorithm described in [11] by computing the distance between two segments \tilde{S}^a and \tilde{S}^b , using the cosine distance of their vectorized transition matrices:

$$\delta(\tilde{S}^a, \tilde{S}^b) = \frac{\sum_{i=1}^n \sum_{j=1}^n m_{i,j}(\tilde{S}^a) \cdot m_{i,j}(\tilde{S}^b)}{\sqrt{\sum_{i=1}^n \sum_{j=1}^n (m_{i,j}(\tilde{S}^a))^2} \cdot \sqrt{\sum_{i=1}^n \sum_{j=1}^n (m_{i,j}(\tilde{S}^b))^2}}$$

When a new segment is identified, the algorithm assigns it to the nearest cluster within a maximum distance $d_{max} \in [0, 1]$. If there is no cluster within d_{max} , then a new cluster for the segment is created. Each cluster is associated with a distinct ‘‘higher level’’ symbol (see also Figure 3). If a segment is associated with an already existed cluster (within d_{max}), it is replaced with a new (higher level) RLE triples as follows. Let $\tilde{S}(u, m)$ be the segment to be replaced and Δ be the symbol of the cluster associated with the segment $\tilde{S}(u, m)$. Then, we replace $\tilde{S}(u, m)$ with a single triple $(t_u^b, t_{u+m-1}^e, \Delta)$. Finally, we call the clustered segment as a *state*.

IV. APPLICATIONS

In this section, we show how various higher level applications can be built on top of our state representation without converting it back to the original values, i.e., state forecasting, anomaly detection, and compression. Additionally, since state recognition is a ‘‘natural’’ capability of StateFinder, we refer the readers directly to Section V-B. Due to the 8-page limitation, we discuss state forecasting in supplementary material [16].

State Anomaly Detection. As the normal behavior of the system (i.e., things sensed by the sensors, such as human or natural phenomena) is characterized by some stability, it would be converted to higher level symbols (or states) by the StateFinder algorithm. An anomaly, however, would not be converted. In other words, a capability to detect anomaly is

also implicitly built into the symbolic conversion. This makes the anomaly detection task using our symbolic representation rather straight-forward, without reverting them to their original values. Additionally, RLE triple representation and StateFinder also make the whole process more efficient since the normal behavior is now much shorter (see also Figure 3 for an illustration).

To detect anomalies, we adapt the algorithm in [17], where the authors use the distribution of symbols’ occurrences in the whole training set (in the supervised case), or in the previous time window (in the unsupervised case) as the normal reference. Unfortunately, this does not work in our case, since human behavior is daily periodical. Thus, instead of using the whole training set or only the previous time window, we use the behavior of the same time of day/week (in the training set) as the normal reference. In addition, the Segment Transition Matrix implicitly contains the symbol distributions and thus can be directly used for estimating them.

More formally, let r be a time duration (e.g. 15, 30, or 60 minutes), as our detection resolution. We assume that the system of interest inherits daily periodical behavior. We then divide the a day into time intervals $\{R_1, \dots, R_n\}$, each of length r . Then, we compute, over the entire training set, $\{D_1, \dots, D_n\}$, the distribution of symbols’ lengths which occur within each time interval, where D_i is the symbol length distribution within R_i . We also compute $\{D'_1, \dots, D'_n\}$, the distribution of symbol length in the test set for each time interval. Next, the anomaly score of time interval R_i in the test set is computed as the dissimilarity between D'_i and D_i . The less similar D'_i is to D_i , then the more likely R_i in the test set contains anomalous behavior. Moreover, setting r differently allows us to see anomalies with a different granularity.

Data Compression Because symbolization is a kind of time series summarization technique, StateFinder can also be used for compressing the time series. If one increases d_{max} at each level, it is possible to compress further, but some information will be lost. Compressing a completely random time series, however, would be very challenging using this method.

V. EXPERIMENTS

In this section, we demonstrate the applications of our approach to real-world dataset. To show the versatility of StateFinder, we deliberately use datasets from various domain, such as household energy usage, human activity recognition, and location tracking. Unless stated otherwise, we use the fading factor $\lambda = 0.9$ to update the length-frequency set (Definition 4).¹ The state forecasting and running time experiments and our source code can be found in [16].

A. State Preservation Index

In this experiment, we use the REDD dataset [19], containing smart meter readings for 6 houses, divided by appliances, with a 3-second sampling rate. We show here the result from channel 11 of house 1 which representing the microwave oven, since (compared to others) it contains more heterogeneous states. The time series plotted in Figure 5a has 745,878 values, representing one month of data. Figure 5b represents one hour of this same time series. On the left of the Figure, when the

¹According to Papadimitriou et al. [18], a value around 0.96 is reasonable for a stationary time series, however as we are dealing with switching between several stationary states, a lower value is preferred.

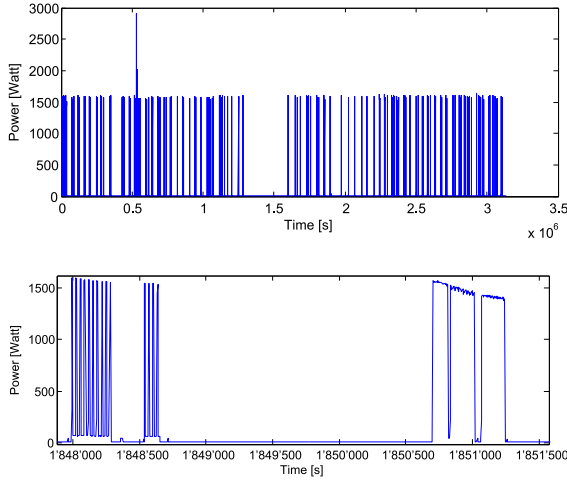


Fig. 5: Complete microwave dataset, showing one month of usage (top) and one hour of microwave oven usage (bottom), showing different power levels.

microwave oven is not set on full power, as it is on the right, there are distinguishable heating cycles.

Figure 6 shows the *state preservation indices* (or spi, see also Eq. 5) of different symbolization methods. For Spclust, Figure 6a shows that the highest state preservation index is obtained using grid size=20 and thresholds equal to 100, 200, or 300. In addition, we also compare the symbolization results from Spclust using other symbolization methods, i.e uniform, median, distinct median [20] and KMeans clustering. Figure 6b shows how Spclust and the other symbolization methods map the microwave power usage into symbols. In this case, Symbol 1 can be interpreted to decodes the *off* or *standby* state, Symbol 3 decodes the *on* state, whereas Symbol 2 decodes the *intermediary* state. Spclust appears to be the most suitable method to perform the symbolization since it has the highest spi value (see Figure 6c).

Driven by the fact that Spclust significantly outperforms other methods, while having rather similar power-to-symbol maps, we carefully analyzed the mapping and the dataset. There are some small power usages around 30 watts which are put in the off state by almost all methods (except median). This power reading appears only once in a while, and in a short period (3 or 4 seconds) between the off states. Thus, it makes sense to assume that it indeed belongs to the off state. We also observe that there are some intermediary power usages around 60 watts. Unlike the 30 watt power usages, this 60-watt power usages appear in longer duration (around 20 or 30 seconds) and systematically between the on states (see e.g., Figure 7A). While other methods include these usages in Symbol 1 (off or standby states, which is not the case), Spclust includes them in Symbol 2 (intermediary state, which confirms our insight). After consulting our colleagues from the Electrical Engineering department, they confirmed that Figure 7A indeed shows how a microwave works when it is set to a *medium* power. Rather than emit the exact medium power, it emits high power intermittently (proportional to the power setting). The 60-watt power usages are used for lighting, rotating the plate, and (possibly) turning on its fan.

B. State Recognition

As StateFinder is designed to extract states in the input data, *state recognition* is the most direct use of its output. The resulting states can then be mapped to a specific semantic. For

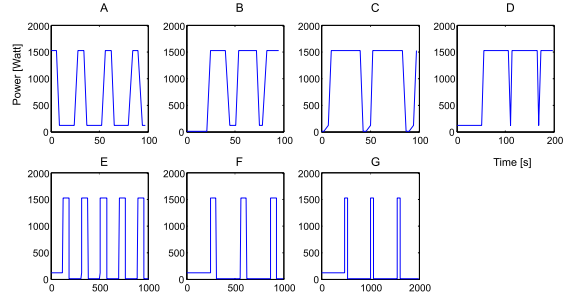


Fig. 7: Prototypes of the states, generated from the transition matrices of the microwave time series in Figure 5a.

instance, we show below how StateFinder is able to identify different heating levels of a microwave oven.

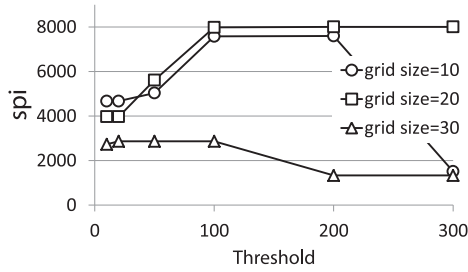
Using Spclust grid size=20 and threshold=200, we extracted the states using StateFinder on the same dataset as in Section V-A. Figure 7 shows the prototypes of each state found. The four heating levels that were used during the experiment, are noticeable as the states A, B, C and D, while the other states shows different microwave usage patterns. Overall, Figure 7 also shows that StateFinder is able to identify states of various length or time scales (e.g., seconds, minutes, hourly). Note that, the capability to automatically discover appliances' states in an unsupervised way is crucial, especially since there are countless new electrical appliances released every day. For instance, a good smart home/energy management systems should be able to learn and recognize the states of an appliance that is newly bought by the home owner.

In addition, our algorithm is not specific to one domain, but rather to a certain definition of the searched states. In the next experiment, we illustrates another use-case involving mobile phone accelerometer data. We use the dataset provided on the UCI Machine Learning Repository by Anguita et al. [21]. The data of user 6 is particularly interesting since the patterns are clearly distinguishable by human eyes, and therefore makes it possible to validate the output of our algorithm. In the Figure 8 (top) we show the tBodyGyro-mean()-X feature which, as its named implies, represents the X axis of the gyroscope with the 50Hz data being smoothed over a sliding window of 128 readings. The data is first symbolized using 3 symbols and then given to the StateFinder. Different darkness levels indicate the different symbols: 0 to 2 for the symbol level 0, and 3 to 4 for the symbol level 1, which represent the states generated by the StateFinder. To better understand the state identification, Figure 8 (bottom) shows on the y-axis symbols level 0.

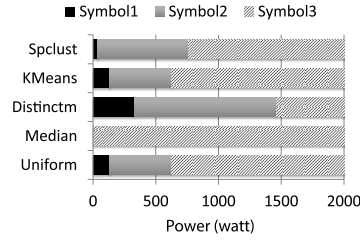
The states found can be matched to the user's activity. From the ground truth, provided with the dataset, we are able to associate a semantic with the two identified states, namely symbols 3 and 4. The first is "going upstairs" while the second one is "going downstairs". As other labels like "sitting" or "standing" are not linked to any specific pattern on this axis of the gyroscope, no state is linked to them.

C. State Anomaly Detection

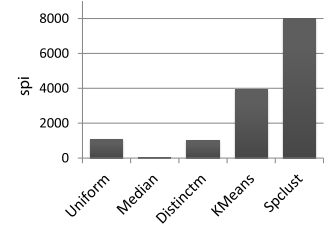
We run our anomaly detection algorithms on symbols level 1, using the same smart-meter dataset and symbolization process as in Section V-B. For this experiment, we add a synthetic anomaly, as if the microwave oven was left running for around 2 hours. In Figure 9, the top plot shows the raw data where the anomaly was added in the first quarter. The bottom plot shows the output of our anomaly detection algorithm that



(a) State preservation indices of Splust using different grid size and threshold. The higher the better.



(b) Power to symbol mapping of different methods.



(c) State preservation indices of different methods. The higher the better.

Fig. 6: Evaluation of symbolic representation on microwave dataset. In (a) the highest eval value is achieved using grid size=20 and threshold=100, 200, or 300. Next, for (b) and (c), we set Splust grid size to 20 and threshold to 200. The alphabet size of uniform, median, and distinct median is set to 3. For KMeans, we set $k=3$.

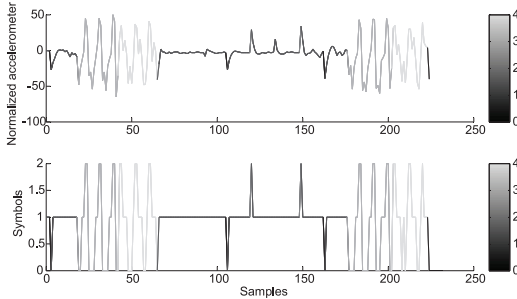


Fig. 8: Gyroscope value on x-axis, recorder on a smartphone while the user was performing different activities. Below is the symbolized version using 3 symbols and the color indicates the state detected by the StateFinder.

indicates the probability of an anomaly happening at any given time, clearly shows a peak at the time when the anomaly happened.

VI. RELATED WORK

Time series segmentation. A large family of methods for segmenting time series aim to summarize them based on piecewise approximations using different functions (see, e.g., [22]–[24]), where segmentation points are found either by minimizing the model’s fitting error or at regular intervals. Chung et al. [25] map a predefined set of patterns onto the time series to identify the segments dynamically, while Guo et al. [26] directly use a set of approximation functions to select the best adapted models for each segment. Srinivasan et al. [12] use the states found in a feedback loop, as contextual information to classify the patterns in the time series in a supervised manner. The classification output then allow the time series to be segmented into different states. Kohlmorgen and Lemm [11] presented an online algorithm for segmenting time series using HMM and a modified Viterbi algorithm. All the methods above, however, assume that every part of the time series must belong to a state or a segment. On the contrary, we keep the transitional part (as well as the rare sequences and anomalies) as it is, so as to lose less information.

Time series subsequences clustering. Ramoni et al. [27] present the BCD clustering algorithm, which is able to cluster time series subsequences, using “event markers” (simultaneous changes on at least three sensors) as segments separators. Denton et al. [28] cluster time series subsequence using a kernel-density-based algorithm. However, the size of the subsequences has to be predefined and only the subsequences with similar length can be compared, as they use the Euclidean

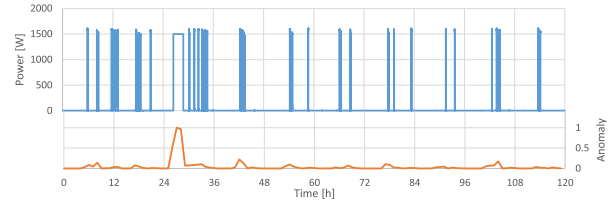


Fig. 9: Anomaly detection on states and symbols generated from the microwave dataset. The top plots shows the raw data including the anomaly around hour 30. The bottom plot show the anomaly probability computed from the the StateFinder output symbols.

distance. Rakthanmanon et al. [15] introduced the idea, which we also applied, that not all the data should be clustered and transition phases should be ignored. They developed a clustering algorithm for discrete time series, based on the Minimum Description Length principle. It requires, however, approximate length of the motifs (or subsequences) that the user interested in as input. Wang et al. [29] represent their clusters as states of a Markov Chain, which is similar to our approach. However, all time series segments, including the transition and meaningless phases, are associated with a state. Additionally, they require an extended Viterbi algorithm and iterative refinement steps, which are both computationally expensive and less suitable for online processing.

Online frequent pattern mining. In addition to segmentation, online processing has also been pushed into the field of mining frequent patterns. Han et al. [30] surveyed some methods for mining frequent itemsets over data streams (see Chapter 5.3). Mueen and Keogh [31] proposed an algorithm to maintain *exact* motifs over a time-window for time series summarization/compression. Together with other more recent works [32], [33], and those mentioned in the survey by Aggarwal [34], all these algorithms identify size-dependent patterns. Furthermore, there is a fundamental difference between our definition of *state* and what they refer to as *patterns*, i.e., patterns are characterized by their length, whereas states are by their dynamics. Additionally, depending on the dynamics of the time series, a pattern could also appear in different states.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented an online and unsupervised state recognition algorithm *StateFinder* that is able to detect and identify states in symbolic time series. Since initial symbolization determines StateFinder performances, we proposed a *state preservation index* to assess how good a symbolic representation is at preserving systems’ state transitions. We

also proposed the *Spclust* algorithm to build the initial symbolization, also for multidimensional time-series. In addition, learning on top of higher level symbols is faster, requires much less space, and produce comparable accuracy. It also enable us to perform analytics locally on smart devices and appliances, which have limited computational and storage capabilities. However, further refinement should be done for non-stationary time series. Next, we plan to deploy our algorithms in real environment [35]. Additionally, since *Spclust* and *StateFinder* preserve interesting events and reduce data size, they could be used to complement Round Robin Database, i.e., to store data further in the past given the same amount of space available.

VIII. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful feedback. This project is evaluated by the Swiss National Science Foundation and partially funded by Nano-Tera.ch with Swiss Confederation financing and the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 288322 (Wattalyst) and 288021 (EINS). Tri Kurniawan Wijaya is supported in part by IBM PhD Fellowship.

REFERENCES

- [1] D. E. Phillips, R. Tan, M.-M. Moazzami, G. Xing, J. Chen, and D. K. Yau, "Supero: A sensor system for unsupervised residential power usage monitoring," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2013, pp. 66–75.
- [2] A. Rai, Z. Yan, D. Chakraborty, T. K. Wijaya, and K. Aberer, "Mining complex activities in the wild via a single smartphone accelerometer," in *Proceedings of the Sixth International Workshop on Knowledge Discovery from Sensor Data*, ser. SensorKDD '12. New York, NY, USA: ACM, 2012, pp. 43–51.
- [3] T. K. Wijaya, M. Vasirani, and K. Aberer, "When bias matters: An economic assessment of demand response baselines for residential customers," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1755–1763, July 2014.
- [4] A. Mishra, D. Irwin, P. Shenoy, and T. Zhu, "Scaling distributed energy storage for grid peak reduction," in *Proceedings of the Fourth International Conference on Future Energy Systems*, ser. e-Energy '13. New York, USA: ACM, 2013, pp. 3–14.
- [5] P. Vytelingum, T. D. Voice, S. D. Ramchurn, A. Rogers, and N. R. Jennings, "Agent-based micro-storage management for the smart grid," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '10, 2010, pp. 39–46.
- [6] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: A multi-resolution clustering approach for very large spatial databases," in *VLDB*, 1998, pp. 428–439.
- [7] J. Heer, N. Kong, and M. Agrawala, "Sizing the horizon," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, USA: ACM, Apr. 2009.
- [8] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule discovery from time series," in *KDD*, R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro, Eds. AAAI Press, 1998, pp. 16–22.
- [9] F. Höppner, "Discovery of temporal patterns. Learning rules about the qualitative behaviour of time series," in *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, ser. PKDD '01. London, UK: Springer-Verlag, 2001, pp. 192–203.
- [10] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in *Proceedings of the ninth International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 493–498.
- [11] J. Kohlmorgen and S. Lemm, "An on-line method for segmentation and identification of non-stationary time series," in *Neural Networks for Signal Processing XI, 2001. Proceedings of the 2001 IEEE Signal Processing Society Workshop*, 2001, pp. 113–122.
- [12] R. Srinivasan, C. Wang, W. Ho, and K. Lim, "Context-based recognition of process states using neural networks," *Chemical Engineering Science*, vol. 60, no. 4, pp. 935–949, Feb. 2005.
- [13] E. Keogh, J. Lin, and W. Truppel, "Clustering of time series subsequences is meaningless: implications for previous and future research," in *Third IEEE International Conference on Data Mining (ICDM)*, 2003, pp. 115–122.
- [14] B. W. Silverman, *Density estimation for statistics and data analysis*. London New York: Chapman and Hall, 1986.
- [15] T. Rakthanmanon, E. Keogh, S. Lonardi, and S. Evans, "Time series epenthesis: Clustering time series streams requires ignoring some data," in *IEEE 11th International Conference on Data Mining (ICDM)*, Dec 2011, pp. 547–556.
- [16] "Statefinder repository," <https://github.com/LSIR/StateFinder>.
- [17] L. Wei, N. Kumar, V. Lolla, E. J. Keogh, S. Lonardi, and C. Ratanamahatana, "Assumption-free anomaly detection in time series," in *Proceedings of the 17th International Conference on Scientific and Statistical Database Management*. Berkeley, CA, USA: Lawrence Berkeley Laboratory, 2005, pp. 237–240.
- [18] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," pp. 697–708, Aug. 2005.
- [19] J. Z. Kolter and M. J. Johnson, "REDD: A public data set for energy disaggregation research," in *Proceedings of the SustKDD workshop on Data Mining Applications in Sustainability*, 2011.
- [20] T. K. Wijaya, J. Eberle, and K. Aberer, "Symbolic representation of smart meter data," in *Proceedings of the Joint EDBT/ICDT Workshops*, ser. EDBT '13. New York, USA: ACM, 2013, pp. 242–248.
- [21] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Proceedings of the 4th International Conference on Ambient Assisted Living and Home Care*, ser. IWAAL'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 216–223.
- [22] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 107–144, 2007.
- [23] D. Lemire, "A better alternative to piecewise linear time series segmentation," in *SIAM Data Mining*, 2007.
- [24] Z. Yan, J. Eberle, and K. Aberer, "Optimos: Optimal sensing for mobile sensors," in *IEEE 13th International Conference on Mobile Data Management (MDM)*, July 2012, pp. 105–114.
- [25] F.-L. Chung, T.-C. Fu, V. Ng, and R. Luk, "An Evolutionary Approach to Pattern-Based Time Series Segmentation," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 5, pp. 471–489, Oct. 2004.
- [26] T. Guo, Z. Yan, and K. Aberer, "An adaptive approach for online segmentation of multi-dimensional mobile data," in *Proceedings of the 11th International Workshop on Data Engineering for Wireless and Mobile Access*. New York, USA: ACM, May 2012, pp. 7–14.
- [27] M. Ramoni, P. Sebastiani, and P. Cohen, "Bayesian Clustering by Dynamics," *Machine Learning*, vol. 47, no. 1, pp. 91–121, Apr. 2002.
- [28] A. M. Denton, C. A. Besemann, and D. H. Dorr, "Pattern-based time-series subsequence clustering using radial distribution functions," *Knowledge and Information Systems*, vol. 18, no. 1, Mar. 2008.
- [29] P. Wang, H. Wang, and W. Wang, "Finding semantics in time series," in *Proceedings of the International Conference on Management of Data - SIGMOD '11*. New York, USA: ACM Press, Jun. 2011, pp. 385–396.
- [30] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 55–86, Jan. 2007.
- [31] A. Mueen and E. Keogh, "Online discovery and maintenance of time series motifs," in *Proceedings of the 16th International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. New York, USA: ACM, 2010, pp. 1089–1098.
- [32] L. Chen, Y. Chen, and L. Tu, "A Fast and Efficient Algorithm for Finding Frequent Items over Data Stream," *Journal of Computers*, vol. 7, no. 7, pp. 1545–1554, Jul. 2012.
- [33] J. S. Tan, Z. F. Kuang, and G. G. Yang, "An Efficient Frequent Patterns Mining Algorithm over Data Streams Based on FPD-Graph," *Advanced Materials Research*, vol. 433-440, pp. 4457–4462, Feb. 2012.
- [34] C. C. Aggarwal, "Mining sensor data streams," in *Managing and Mining Sensor Data*, C. C. Aggarwal, Ed. Springer US, 2013, pp. 143–171.
- [35] "Opensense2," <http://www.nano-tera.ch/projects/423.php>.