<http://www.diva-portal.org>

Postprint

This is the accepted version of a chapter published in *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning: Second International Workshop, IoT Streams 2020, and First International Workshop, ITEM 2020, Co-located with ECML/PKDD 2020, Ghent, Belgium, September 14-18, 2020, Revised Selected Papers*.

Citation for the original published chapter:

Fanaee Tork, H., Bouguelia, M-R., Rahat, M. (2020)

CycleFootprint: A Fully Automated Method for Extracting Operation Cycles from Historical Raw Data of Multiple Sensors

In: Gama, J., Pashami, S., Bifet, A., Sayed-Mouchawe, M., Fröning, H., Pernkopf, F., Schiele, G., Blott, M.öning Franz Pernkopf Gregor Schiele Michaela Blott (ed.), *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning: Second International Workshop, IoT Streams 2020, and First International Workshop, ITEM 2020, Co-located with ECML/PKDD 2020, Ghent, Belgium, September 14-18, 2020, Revised Selected Papers* Switzerland: Springer Publishing Company

Communications in Computer and Information Science

<https://doi.org/10.1007/978-3-030-66770-2>

N.B. When citing this work, cite the original published chapter.

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-43659>

CycleFootprint: A Fully Automated Method for Extracting Operation Cycles from Historical Raw Data of Multiple Sensors

Hadi Fanaee-T¹, Mohamed-Rafik Bouguelia¹, Mahmoud Rahat¹, Jonathan Blixt², and Harpal Singh²

¹ Center for Applied Intelligent Systems Research, Halmstad University, Sweden

² Alfa Laval Tumba AB, Sweden

{hadi.fanaee, mohamed-rafik.bouguelia, mahmoud.rahat}@hh.se
{jonathan.blixt, harpal.singh}@alfalaval.com

Abstract. Extracting operation cycles from the historical reading of sensors is an essential step in IoT data analytics. For instance, we can exploit the obtained cycles for learning the normal states to feed into semi-supervised models or dictionaries for efficient real-time anomaly detection on the sensors. However, this is a difficult problem due to this fact that we may have different types of cycles, each of which with varying lengths. Current approaches are highly dependent on manual efforts by the aid of visualization and knowledge of domain experts, which is not feasible on a large scale. We propose a fully automated method called CycleFootprint that can: 1) identify the most relevant signal that has the most obvious recurring patterns among multiple signals; and 2) automatically find the cycles from the selected signal. The main idea behind CycleFootprint is mining footprints in the cycles. We assume that there should be a unique pattern in each cycle that shows up repeatedly in each cycle. By mining those footprints, we can identify cycles. We evaluate our method with existing labeled ground truth data of a real separator in marine application equipped with multiple health monitoring sensors. 86% of cycles extracted by our method match fully or with at least 99% overlap with true cycles, which sounds promising given its unsupervised and fully automated nature.

Keywords: Cycle detection · Sensors · IoT

1 Introduction

Health monitoring of machines plays a significant role in improving their safety, reliability, and their effective lifetime. The cost of machines' failures is usually high and potentially fatal [3]. Anomaly detection is a central component of self-maintenance systems, and it provides failure warnings in advance, which ultimately prevents emergency shutdown and catastrophic consequences.

Based on the availability of labeled data, the methods for anomaly detection can be divided into unsupervised, semi-supervised, and supervised methods. Unsupervised methods such as statistical process control (SPC) methods are typically applied when there is no prior information about data. The central assumption of these methods is

that the significant part of operation goes normal so that they look for those instants that fit least to the rest of the data. When data is partially labeled, for instance, regular operation, we can use semi-supervised methods. These methods construct a model representing normal behavior and then compute the deviation of a test instant from the learnt model. The third group of approaches is supervised learning that assumes that we have the labeled data for both "normal" and "abnormal" states. The general rule is that supervised methods outperform the other two methods, and the semi-supervised methods outperform unsupervised methods [6,3,4].

In the industrial setting, labeling of normal states is much easier and cheaper than acquiring the abnormal states. The reason is that the machines are healthy in the majority of times, while faults occur on rare occasions. This setting is more relevant for semi-supervised models, where labeled is required for only normal states.

However, in practice, collecting a high-quality dataset of regular operation is not that straightforward. For instance, the clients of our industrial partner have some machines that produce a huge volume of data provided by various sensors. The majority of devices are in operation for a long time. Now our partner is interested in using this raw historical data for real-time anomaly detection. The main issue is that the raw data includes a mixture of normal and abnormal states and neutral states where the machine has been shut down due to some reasons. The more challenging issue is that the length of cycles in each operation is not constant and spans from 2 minutes to 2 hours. Besides, we may have different shapes of regular periods.

The current in-use approach for annotating regular cycles relies on some parameters that machines have collected during the operation. A domain expert then goes through the visualization plot of the time series of different sensors in combination with those parameters to annotate regular cycles. This is a very time-consuming task, and it is infeasible for more massive data sets, which is the case for the majority of machines. The second issue is that those parameters used for the identification of cycles sometimes do not match appropriately with the right periods and are not reliable for automation of this process.

To automate the process of regular cycle identification, we propose a fully automated approach that makes no assumption about the distribution of data and does not require any expert knowledge about sensors and their utility. The user can feed into the algorithm as many as sensors she wants. Our algorithm will first find the right sensor that exhibits more recurring behavior, and then it segments the data into some initial cycles. Finally, it prunes the irrelevant sequences by applying the cycle validation condition set by the user.

Besides the main contribution, for the first time, we propose a novel data discretization for transferring time series into a sequence of discrete states.

So, our contributions are as follows.

- We propose a fully automatic approach for the identification of operation cycles. Our method is conservative, in the sense that it is strict in small deviations. Thus is an ideal approach for producing normal labeled data required by semi-supervised methods.
- We propose a new method for converting time series into discrete sequences, which is an alternative to Symbolic Aggregate approximation (SAX) [7]. Although SAX

is a state-of-the-art method for time series discretization, our initial investigation showed poor performance in this application. We explain this in detail later.

Fig. 1: Historical Raw signals include mixture of cycles: Normal operation (Green), failures (red) and missing values/ inactivity (blue)

2 Problem Definition

Fig. 1 illustrates a simplified time series of a sensor in a particular machine. The historical time series of sensors include a mixture of cycles: regular operation, failures, and neutral states (where the device has been inactive due to shutdown or repair). In the example figure, we can see twelve regular cycles with different lengths, one abnormal cycle, and two neutral cycles (one with missing values and one with a fixed amount over a period). All sequences, though appear differently, typically share a sub-segment with a unique pattern. For instance, in the time of discharge, the measurements show a sudden change, and then goes back to normal. However, abnormal cycles might have different causes. Thus the shape and measurements during an irregular period can be completely different from another unusual situation.

We are interested in identifying cycles that share a unique repeating pattern. An example of a simple cycle footprint is illustrated in Fig. 2. However, footprints can

Fig. 2: An example of cycle footprint

be more complicated than this, sometimes not identifiable by eye. The advantage of extracting cycles is that we can then use clustering to separate normal cycles from abnormal cycles and construct an abstract model or dictionary from normal cycles and exploit these in semi-supervised anomaly detection. But this is a challenging problem because footprints are not necessarily the same length. For instance, in one cycle, footprint might appear during one minute, while in the next cycle, we may observe it during a longer or shorter period.

3 Related work

Perhaps the most relevant solutions can be motif discovery methods from time series [8,2]. However, for various reasons, these types of solutions are not an ideal fit for our problem.

The first reason is that motif discovery methods look for most similar frequent motifs, and not all similar patterns. For instance, the design of state-of-the-art techniques such as Matrix Profile [9,10] allows us to detect only the top-k motifs with the time complexity of $O(N^2)$. A modification to enable these methods to obtain all frequent patterns not only elevate the time complexity but also is complicated to implement. Because, for instance, in matrix profile, k most similar segments are assumed to be motifs. But since our target is to identify all motifs of different types, then we cannot merely pick the closest sections. Because, for instance, among those top-k, we may have different motif types. The separation of multiple kinds of motifs is not trivial with these methods. It makes sense because the objective of their initial design is to find only top-k frequent patterns, which is relevant in some applications.

The second reason is that the majority of motif discovery methods concentrate on the shape similarity of sub-segments. Thus they are less sensitive to those motifs that share value change sequence behavior, which is more relevant in sensors.

For instance, assume a temperature sensor. Assuming that the reading is always between 0 and 100, we are confident that the sensor reading never goes lower or upper than this limit. In this context, temperature sequence like [20, 30, 20] is different from [80, 90, 80]. However, motif discovery methods identify these two patterns as motifs, because they appear in U-shape in a sliding window. However, they refer to completely different machine states (one normal and one abnormal).

To understand the problem better, let's look at another example. Suppose a machine that, at the end of cycles, enters into a discharge phase. So if we look at the pressure sensor, we would see a U-shape pattern in the last part of each cycle. By further investigation, we see that in one cycle, the pressure stays in 2 for 30 seconds, goes down to 1 for 5 seconds, and then return to 2 and remain at that level for 9 seconds. In the next cycle, we see that it stays at 2 for 70 seconds, goes down to 1 and stays there for 7 seconds, and then return to 1 and stays at that level for 7 seconds. Now, suppose that in the third operation cycle, at the last part of the cycle the pressure stays at 6 for 30 seconds, goes down to 5 and stays there for 5 seconds, and returns to 6, and it remains at that level for 9 seconds. The third pattern is an anomaly if repeated only a few times. However, motif discovery methods return all these tree patterns as motifs since they appear in the same shape in the sliding window.

4 Proposed Solution

Our solution is based on mining footprints in the cycles. An ideal method should be able to track unique patterns with different lengths and should have some tolerance in the small variations in the patterns. For instance, in the example provided in the last paragraph of the previous section, the range of values is different, but the difference is at the tolerance level.

Finding such footprints in each cycle from a sizeable raw signal is an NP-hard problem. The exhaustive solution is to move windows of varying lengths over the whole signal and make pairwise comparisons between all varying-length windows. If we could assume that footprints appear in the same size, then the solution could be solved with $O(N^2)$ comparisons. However, this is not the case, and footprints can be in different lengths.

Given the above arguments, we have no choice to seek an approximation solution for the problem. We propose to discretize the time series and then reformulate the problem into mining frequent patterns. SAX is perhaps state-of-the-art for this problem, but due to previously-mentioned reasons, SAX is not appropriate for this problem since it focuses on the shape similarity without being able to check the range of values within the shape.

To solve this problem, we propose CycleFootprint, a new algorithm for the detection of operation cycles. CycleFootprint is composed of two main parts: state transformation and mining footprints. For the first part, we propose a new method for converting signals into a sequence of states. Then we mine the discrete space of states to find footprints. After finding the positions of footprints, it is straightforward to detect cycles, since cycles are expected to be found between the two consecutive footprints.

In the following section, we describe our algorithm CycleFootprint in more detail.

5 Algorithm CycleFootprint

CycleFootprint algorithm is presented in Algorithm 1. It is composed of two modules: Timeseries2States and FootprintMiner. The input of algorithm are: x_n : time series of multiple sensors; w_{min} and w_{max} : minimum and maximum of the length of sliding

window; ϵ : The minimum number of members for one state be considered valid; δ : The minimum number of cycles for being considered a valid output. a and b : The minimum and maximum length of cycle to be identified as valid.

The module Timeseries2States transforms the time series of each sensor to state sequences. Then state sequences is fed into module FootprintMiner for identification of primary cycles. The algorithm tests different sliding window sizes and various sensors. We omit those outputs where the number of cycles is lower than the threshold δ . After that, we check the correctness ratio (P) of cycles obtained for different window sizes and various sensors. The final output corresponds to the maximum P obtained within different window sizes and sensors.

In the next subsections, we describe the modules Timeseries2States and FootprintMiner.

Algorithm 1 CycleFootprint

Input $x_n, w_{min}, w_{max}, \epsilon, \delta, a, b$
Output: $Cycles$

```

1: for each sensor  $x_n$  do
2:    $s \leftarrow Timeseries2States(x_n, \epsilon)$ 
3:    $C_n, P_n \leftarrow FootprintMiner(s, w_{min}, w_{max}, a, b)$ 
4:   for  $L = w_{min}$  to  $w_{max}$  do
5:     if  $|C_n^L| < \delta$  then
6:        $P_n^L \leftarrow 0$ 
7:     end if
8:   end for
9: end for
10:  $L', n' \leftarrow \max(P)$ 
11:  $Cycles \leftarrow C_n^{L'}$ 

```

5.1 Transformation of Signal to State Sequences

Due to previously mentioned reasons, we cannot use popular time-series discretization methods such as Piecewise Aggregate Approximation (PAA) [5]. Besides, we want to keep our approach non-parametric. Any assumption about the distribution of data can lead to poor performance if it is not met for some reason.

The architecture of our method is not window-based like SAX. We believe that since the measurements are in a finite range, a global approach should work better.

We use this principle that in a time series that measurements vary in a finite range, the values normally stay within a particular group of ranges. If we somehow manage to identify those meaningful ranges, then we can allocate each range group a number representing that state. Then the transformation of time series is just to read elements and put their corresponding state at each instant. Fig .3 illustrates the idea. The time series in the figure has 55 instants, but it turns out that the values are with three principal ranges of (10-15), (25-35), and (50-60). So our state transformation method converts

Fig. 3: A principle idea behind our discretization approach. Our method applies to whole signal and we do not assume any specific distribution, contrary to methods such as PAA.

this time series to $[1,2,3]$, each number representing one state. In the following, we introduce algorithm `Timeseries2State`, and describe how it finds meaningful range groups and subsequently transform time series to state sequences.

Algorithm `Timeseries2States` Algorithm `Timeseries2States` (Algorithm 2) transforms any time series x into state sequences s . The algorithm has one input in addition to x , and that is ϵ , which is a parameter that allows the user to control the minimum number of required members belonging to one state. For instance, if we find a specific range that has matching members lower than ϵ that would not be considered as a basis for forming a state, and the corresponding elements simply will be ignored.

We first start by re-scaling the time series (line 4) using floor operator over min-max normalization with a range of $(1,3)$. So, any value in time series is converted to either 1, 2, or 3. The rationale behind this choice is that one reserves the places for low measurements, two for median quantities, and 3 for large values.

After $[1,2,3]$ transformation we count the number of elements whose corresponding transformed value is either 1, 2 or 3 (line 5). Afterward, we select two minority counts. For instance, assuming the number of 1, 2, 3 are respectively repeated 100, 50, and 20 times, we operate only on those elements in x that their corresponding projection is either 2 or 3. Then, if the count was upper than a pre-defined threshold ϵ we create a state and allocate those elements to that state. Otherwise, we simply ignore those elements. Then we replace elements corresponding to minority groups (in above example 2 and 3) with NaN.

We repeat this procedure iteratively until all elements of x become NaN. $y \in [1, \dots, d]$ keeps the state sequences. The final step (line18) accounts for converting y to s by keeping only non-repeating elements. For instance, state sequence such as 122233334 is transformed to 1234 at this step. This is a crucial part that allows us

to find semantic footprints, i.e., footprints like the example that are similar but appear in different lengths.

Algorithm 2 TimeSeries2States

Input x, ϵ
Output s

```

1:  $d \leftarrow 0$ 
2:  $y \leftarrow \text{zeros}(\text{size}(x))$ 
3: while  $x$  is not empty do
4:    $z = 1 + \text{floor}(\frac{2(x - \min(x))}{\max(x) - \min(x)})$ 
5:    $c \leftarrow \text{sort}(\text{count}(\text{unique}(z)))$ 
6:    $r_1 \leftarrow$  corresponding elements to  $c(1)$ 
7:    $r_2 \leftarrow$  corresponding elements to  $c(2)$ 
8:   if  $|r_1| \geq \epsilon$  then
9:      $d \leftarrow d + 1$ 
10:     $y(r_1) = d$ 
11:   end if
12:   if  $|r_2| \geq \epsilon$  then
13:      $d \leftarrow d + 1$ 
14:     $y(r_2) = d$ 
15:   end if
16:   Replace elements in  $x$  with NaN for elements  $r_1$  and  $r_2$ 
17: end while
18: for each element  $y_i$  in  $y$  do
19:   if  $y_i \neq y_{i+1}$  then
20:     Append  $y_i$  to  $s$ 
21:   end if
22: end for

```

5.2 Mining Footprints

Converting time series to state sequences allows us to work in discrete space and develop efficient methods for mining footprints. For this purpose, we use a sliding window approach over state sequences and test windows of different sizes to find our footprints of interest.

We form a word from the observed states in each window (e.g., ABCDE or ABABA for window length of 5). Among all words obtained from sliding windows, we look for words that contain more information. For instance, ABCDE is preferred over ABABA because it includes five unique states, while ABABA contains two unique states. Patterns such as ABABA usually cannot be footprints. Instead, they are non-interesting parts of signals (like a fixed line with some little deviations). Good footprints usually are sub-segments of the signal that have more variance.

After removing less informative words (like the ABABA in the above examples), we convert each word to an order-blind pattern. For instance, ABCDE and BCDEA

both will be converted to ABCDE. Similarly, CDEFH and DEFHC are converted to CDEFH (unique letters, without considering the order of states). This step is designed to merge similar patterns that have some shifts. For instance, ABCDE and BCDEA probably refer to the same concept, only with one shift.

Afterward, we count the number of the appearance of the order-blind patterns and pick the most repeated one. For instance, if ABCDE is repeated 150 times and CDEFH is repeated 80 times, we pick ABCDE. This is the point that we realize that our footprints should contain letters of A, B, C, D, and E.

In the next step, we revisit the sliding windows and look for words that contain A, B, C, D, and E. For instance, ABCDE, BEACD both match the condition. So we count the number of occurrences of these two candidates and pick the most frequent one. For example, if ABCDE is repeated 200 times and BEACD occurs 190 times, ABCDE is our footprint. This step is designed to pick the best ordering among similar patterns that differ only in order.

After we found our footprints, we find their positions in the original signal. Then the initial list of cycles is assumed to be those periods between two consecutive footprints. However, from domain knowledge, we might know that a valid cycle should have a specific length. If we apply this condition, we can estimate how many cycles fail to pass this condition. Then, we can obtain the ratio of valid cycles (hereafter we call it cycle correctness ratio) to initial cycles, which is an excellent criterion to verify the correctness of our cycle mining process.

The rationale behind the usefulness of correction ratio is that in the ideal cycle detection scenario, all cycles (accuracy of 100%) should match the length criterion (e.g., in our application context cycles should be greater than 2 minutes and lower than 2 hours). This is against a random cycle detector, which probably generates lots of cycles that are out of length condition. Hence, the ratio of valid cycles is an excellent factor to look at when choosing the right signal or window size.

To find the final cycles, we test different window sizes and signals and compute the cycle correctness factor to pick the one that gives the most accurate cycles.

In the following, we describe the algorithm in more detail.

FootprintMiner Algorithm FootprintMiner Algorithm is presented in Algorithm 3. The algorithm receives four inputs: s : state sequence vector obtained from Timeseries2States process; w_{min} and w_{max} : Minimum and maximum length of sliding windows; a , and b : minimum and maximum length of a valid cycle.

At line 2 we begin by moving a sliding window of varying length L (from w_{min} to w_{max}) over state sequences s . At line 4 we create a word of length L from states observed at window w . At the next line we obtain the unique characters from S_w^L .

At lines 6-11 we check if there is more informative word comparing already observed ones. If the new observed word has more unique characters than previously observed words so far, we forget all words and start recording from the current window.

At lines 14-17 we first obtain the most frequent order-blind patterns and then look for words containing letters of that pattern. Among the matched words, we pick up the most frequent word.

Finally, at lines 14-25, we first obtain the initial cycles and then compute the correctness ratio by applying the expected length condition.

The algorithm ends with returning the obtained cycles (C) and cycle correctness ratio (P) for all window sizes.

Algorithm 3 FootprintMiner

Input $s, w_{min}, w_{max}, a, b$
Output: C, P

```

1:  $mx^L \leftarrow 0, L \in \{w_{min}, \dots, w_{max}\}$ 
2: for each sliding window  $w$  over  $s$  do
3:   for  $L = w_{min}$  to  $w_{max}$  do
4:      $S_w^L \leftarrow s(w : w + L - 1)$ 
5:      $M_w^L \leftarrow \text{Unique}(S_w^L)$ 
6:     if  $\text{count}(M_w^L) > mx^L$  then
7:        $mx^L \leftarrow \text{count}(M_w^L)$ 
8:       Empty  $S_w^L, M_w^L$ 
9:        $S_w^L \leftarrow s(w : w + L - 1)$ 
10:       $M_w^L \leftarrow \text{Unique}(S_w^L)$ 
11:     end if
12:   end for
13: end for
14: for  $L = w_{min}$  to  $w_{max}$  do
15:    $T^L \leftarrow$  words in  $S^L$  that contain unique states corresponding to most frequent  $M^L$ 
16:    $F^L \leftarrow$  Search positions in  $s$  that correspond to most frequent  $T^L$ 
17: end for
18: for  $L = w_{min}$  to  $w_{max}$  do
19:   for each element  $F_i^L$  in  $F^L$  do
20:      $Ri^L \leftarrow x(F_i^L : F_{i+1}^L)$ 
21:     if  $a < |Ri^L| < b$  then
22:       Append  $Ri^L$  to  $C^L$ 
23:     end if
24:    $P^L = |C^L|/|R^L|$ 
25: end for
26: end for

```

6 Experimental Evaluation

6.1 Dataset

The dataset includes historical data of multiple health monitoring sensors installed on a fuel separator in operation on a ship. Fuel separators remove oil and unwanted particles using centrifugal process. The sensors measure parameters such as water transducer, various temperatures, outlet/inlet/drain pressures, bowl speed, etc. The studied data corresponds to almost two months between Feb 2020 to April 2020, sampled at a frequency of 1 second, which makes 17 signals of length 5,324,858.

6.2 Configuration

We set the following parameters for CycleFootprint algorithm. $\epsilon = 5$, $\delta = 100$, $w_{min} = 4$, $w_{max} = 20$, $a = 120$, $b = 8000$

6.3 Results

Fig. 4: Examples of detected cycle via CycleFootprint (left) vs. ground truth (right)

Table 1 shows the output of CycleFootprint algorithm for different sensors on the separator dataset. The first column refers to the extracted footprints. Each state is represented with a number with three digits with leading zeros for lower quantities than 100 and 10. For instance, 013014015017 should be read as [13, 14, 15, 17]. The second column of the table contains the number of total found cycles. The third column presents the number of valid cycles after applying the length condition. L is the size of the window. The fifth column also shows the unique states in the footprint. Finally, P represents the cycle correctness ratio.

The interesting observation is that the algorithm has been able to find relevant signals, which was identified by the domain expert, the most relevant signals for cycle identification. Out of $340=17 \times 20$ tested scenarios (17 sensors by 20 window sizes), we find 24 unique footprints. Among them, "Bowl Speed" with a window length of 6 has produced 236 valid cycles, out of 303 total cycles, which makes the correction ratio of 77.88%. Therefore, the states' segment [8, 10, 12, 14, 15, 16] on bowl speed sensor is the selected footprint. Fig. 5 illustrates some footprints discovered by this pattern. As we can see, our method can detect semantic footprints, in the sense that they can be

Fig. 5: Example of semantic footprints detected by pattern [8, 10, 12, 14, 15, 16] on bowl speed sensor via CycleFootprint algorithm.

Fig. 6: Distribution of length of detected footprints

in different length (See the range of lengths in Fig. 6). If we rely only on same-length footprints, then probably we will lose lots of true cycles.

Fig. 7: Accuracy of cycle detection according to ground truth

Comparing the cycles obtained by this footprint with ground truth, we find that the majority of cycles match with the true cycles. Fig. 7 shows the recovery ratio of extracted cycles comparing ground truth. Almost 86% of cycles have a full or at least 99% match with ground truth. 9% of cycles reconstruct the cycles in ground truth with coverage of 90-99%, and the rest 5% partially cover some cycles in the ground truth. Fig. 4 illustrates some examples from extracted cycles versus their corresponding ones in the ground truth.

7 Conclusion

We introduce a new approximation method based on mining footprints for the detection of operation cycles from the historical time series of multiple sensors. Our experimental evaluation on a real separator in marine shows that our method can detect the majority of right cycles with good coverage, i.e., in 86% of cycles full or at least 99% recovery.

Our method is a general solution that can be used for any kind of machine operation that is composed of a time series with a finite range. The time series of interest can be discrete (e.g., count), continues, and with any distribution. Our method, however, does not apply to time-series that the range of values is infinite or time series with weak periodic behavior.

This is an under development study. We still need to evaluate the proposed method on larger scales, as well as other settings and machines. The next step is to separate regular cycles from abnormal and neutral states using spectral models (e.g., [1]). Our ultimate goal is to learn the normal state of all machines, making a semi-supervised model, and transfer the summarized models/dictionaries to sensor devices for real-time anomaly detection.

Table 1: Output of CycleFootprint Algorithm on Seperator dataset

Footprint	TotalCycles	ValidCycles	L	Unique P
Outlet Pressure				
013014015017	12854	792	4 4	0.061615061
010013014015017	3864	1033	5 5	0.267339545
010013014015017018	1400	688	6 6	0.491428571
010013014015017018019	278	167	7 7	0.600719424
010013014015017018019020	35	7	8 8	0.2
Inlet Pressure				
017018019020	1491	310	4 4	0.207914152
017018019020021	16	5	5 5	0.3125
012013014016017018019020021	2	1	13 9	0.5
012013014016017018019020021	3	1	14 9	0.333333333
012013014016017018019020021	4	1	15 9	0.25
012013014016017018019020021	5	1	16 9	0.2
012013014016017018019020021	6	1	17 9	0.166666667
012013014016017018019020021	7	1	18 9	0.142857143
012013014016017018019020021	8	1	19 9	0.125
012013014016017018019020021	9	1	20 9	0.111111111
Drain Pressure				
013014016019	3396	601	4 4	0.176972909
013014016019020	406	32	5 5	0.078817734
005007009011013015	42	17	6 6	0.404761905
003005007009011013015	37	16	7 7	0.432432432
Water Transducer				
012014015018	6567	1183	4 4	0.18014314
012014015017018	398	283	5 5	0.711055276
012014015017018020	23	13	6 6	0.565217391
Bowl Speed				
013014015016	2407	1796	4 4	0.746157042
011013014015016	593	387	5 5	0.652613828
008010012014015016	303	236	6 6	0.778877888
008010012013014015016	226	127	7 7	0.561946903
008010011012013014015016	133	49	8 8	0.368421053
008009010011012013014015016	16	2	9 9	0.125
001003005006008010012013014015	10	1	10 10	0.1
003005006008009010011012013014015	9	1	12 12	0.111111111
Temperature				
007008009010011012013014015	3	1	11 9	0.333333333

References

1. Fanaee-T, H., Oliveira, M.D., Gama, J., Malinowski, S., Morla, R.: Event and anomaly detection using tucker3 decomposition. arXiv preprint arXiv:1406.3266 (2014)
2. Fu, T.c.: A review on time series data mining. *Engineering Applications of Artificial Intelligence* **24**(1), 164–181 (2011)
3. Jin, X., Wang, Y., Chow, T.W., Sun, Y.: Md-based approaches for system health monitoring: A review. *IET Science, Measurement & Technology* **11**(4), 371–379 (2017)
4. Jin, X., Zhao, M., Chow, T.W., Pecht, M.: Motor bearing fault diagnosis using trace ratio linear discriminant analysis. *IEEE Transactions on Industrial Electronics* **61**(5), 2441–2451 (2013)
5. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems* **3**(3), 263–286 (2001)
6. Laskov, P., Düssel, P., Schäfer, C., Rieck, K.: Learning intrusion detection: supervised or unsupervised? In: *International Conference on Image Analysis and Processing*. pp. 50–57. Springer (2005)
7. LIN, J.: Finding motifs in time series. In: *Proc. of Workshop on Temporal Data Mining*, 2002. pp. 53–68 (2002)
8. Torkamani, S., Lohweg, V.: Survey on time series motif discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **7**(2), e1199 (2017)
9. Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A., Keogh, E.: Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: *2016 IEEE 16th international conference on data mining (ICDM)*. pp. 1317–1322. Ieee (2016)
10. Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Zimmerman, Z., Silva, D.F., Mueen, A., Keogh, E.: Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Mining and Knowledge Discovery* **32**(1), 83–123 (2018)