Graphical Closure Rules for Unsupervised Load Classification in NILM Systems

Joseph Krall, Ph.D.
Chief Data Scientist, LoadIQ
800 Haskell Street, Reno, NV, USA

Sohei Okamoto, Ph.D.
Senior Software Engineer, LoadIQ
800 Haskell Street, Reno, NV, USA

Hampden Kuhns, Ph.D.
CEO/Co-Founder, LoadIQ
800 Haskell Street, Reno, NV, USA

***Abstract*—Unsupervised NILM requires a robust approach to quantify energy usage from electrical loads operating over a broad range of power states and temporal schedules. To be truly unsupervised, the system must automatically scale its detection resolution and find patterns that describe the state of major loads on a circuit. A framework of discretizing an energy time series into clustered transitions and steady states that creates a directed multi-graph with steady states as nodes and transitions as edges is presented. Cycles in the graph represent load behavior patterns where one or more loads change state and return to their original state. The sequences of transitions in these cycles are used to qualify the existence of individual loads. Internal metrics are presented that indicate the self-consistency of the load classification without the presence of ground truth datasets.**

Keywords— NILM, graph theory, unsupervised classification

I. Introduction

One of the primary applications of NILM is to break down and disaggregate power time series into component signals that represent the major energy consuming and high demand power loads on a circuit. With this information, users can target energy conservation measures at loads that have the greatest potential for savings. The value of NILM is maximized when major load disaggregation can be performed with minimal hardware, installation, and human supervision costs.

Great progress in algorithm formulation has been made in recent years with supervised methods that leverage training datasets to recognize individual loads on a circuit [1]. The practicality of supervised algorithms may lose value without the availability of a wide variety of public datasets. To this end, unsupervised algorithms may be more useful as they do not require the training from ground truth to operate.

Toward unsupervised deployments, event based NILM systems as defined by [2] must automatically: (1) choose the resolution for event detection based on the aggregate power time series [3-5]; (2) recognize temporal patterns or features in recurring loads [6-8]; (3) distinguish transition events between single loads and combinations of individual loads [9]; (4) infer the load state of isolated loads throughout time [10]; and (5) estimate the load energy consumption from these results [11]. Optimizations of each step are topics of ongoing research and this paper addresses systematizing steps 2 through 4.

As new NILM approaches are being developed, initial emphasis is rightly placed on evaluating performance with datasets that have ground truth measurements [1,2,12]. Complexity as described by [13] is an independent property of circuits and will affect the performance of load disaggregation. In the primary application of providing a disaggregation of major loads for 'real world' use cases, there will be no ground truth and the quality of results need to be assessed based on other metrics related to how loads fit a predefined model. For users to have confidence in NILM results, self-consistency metrics must indicate the reliability of results without a comparison to ground truth results.

We present a load recognition approach that exploits the Zero Loop Sum Constraint (ZLSC) [14] and a graphical model based on steady state and transition clustering to isolate individual loads, detect combination transitions, and determine if an isolated load may have multiple instances on a circuit.

II. Event Detection and Clustering

Event based NILM systems use a filter to detect when a power signal is in a steady state or transition. Numerous methods of varying complexity have been developed to discretize an aggregate power time series [3,15,16]. In this application, we use a simple windowed rate of change indicator to determine if a power sample is within a transition. A fixed threshold $|dP/dt|$ is applied to the dataset. Periods below the threshold are designated at steady states if they exceed a minimum time duration *t_min*. Transition features are extracted from the difference in power from the first interval of a steady state *dt* and the last interval of the previous steady state. Steady state features are extracted from the entire period.

Features may be any set of discrete measurements derived from the aggregated time series. For this example, the average normalized [14] power profile (32 samples per voltage cycle) is extracted for each steady state and transition. These features are clustered from streaming data using a fixed-radius nearest neighbor method that ensures the intra cluster diameter is less than 2 times a predefined RMS cluster distance in Watts. The results approximate agglomerative clustering with a dendrogram cut at a threshold [17]. Power profile data may be reclustered to change the resolution and number of transition and steady state clusters. The cluster diameter represents a minimum detectable load power that can be resolved with this approach.

III. Graphical Closure Rules

A multidigraph is a graph of nodes and directed edges where edges have their own identity. In other words, there can be many edges between a same pair of nodes. The sequence of steady state and transition cluster labels represent nodes and edges on a multidigraph: every transition is an edge that is directed from a starting steady state to an ending steady state.

Edges thereby become unique sequences which form a clustered power time series with *Start*, *Transition*, *End*, and *Count* fields. These STEC edges then become the basis of a STEC multidigraph that represents the dataset graphically. This approach is similar to the one described in [18].

Each STEC record can be weighted to signify its error residual. Let θ be the maximum allowed cluster radius (RMS distance in Watts). S, T, E are the three sets of clustered profile values for start, transition, and end cluster profiles within a STEC record. C is the number of edges in the record, and $N = 32$ is the number of values within each profile. Then, the edge weighted residual (EWR) value can be calculated as:

$$EWR = \frac{1}{\sqrt{3\theta^2 C^2}} * \sqrt{\frac{1}{N} \sum_{s,t,e \in S,T,E} (s + t - e)^2}$$

The edge weight (EWR) has the property of being lower for edges where the transition profile clusters are approximately equivalent to the difference in steady state period clusters. If a steady state power drifts slowly (with respect to the event threshold) without triggering an event, the edge weight will be larger than a more consistent square wave step.

Directed cycles in the STEC graph represent a path of connected edges that has the same start and end nodes. A cycle is related to the ZLSC [14] and represents a series of transitions that return to the original steady state cluster profile. Self-loops are edges where the source and target node are the same and are cycles with path length of 1. Transitions that are predominantly in self-loops are classified as *trivial* in that they represent events that did not cause change in steady state.

Transitions in longer cycles represent patterns of one or more loads that actuate on and off. These groupings of transitions are called closure rules, and they are the basis for extracting features of major loads on a circuit.

Fast cycle detection in directed graphs is a well-studied and optimized algorithm [19]. We use NetworkX on a Python platform to aid with implementation, which is a graph analysis package for the Python programming language and contains source code for cycle detection [20]. Once cycles have been detected, the STEC edges that compose the cycles are used to form Closure Rules that have unique collections of transitions.

Each closure rule has a weighted residual term (CWR) that signifies its error residual in the same manner that STEC edges are rated. The weights of STEC edges in the closure rule are combined using a series resistance model, where *EWR* represents the edge weighted residual of each edge in the rule:

$$CWR = \frac{1}{\sum \frac{1}{EWR}}$$

IV. Closure Rule Simplification

It is useful for closure rules to be reduced and simplified into a minimal most-informative subset of rules. As a first step, we identify candidate trivial transitions as those closure rules with cycles of length 1, as seen in the Python pseudocode for method `removeTrivials()` in Fig 1. When the closure rules are sorted by weight, only those candidate transitions which do not also appear in stronger closure rules are trivial. The `strongestRule()` method returns the rule which a given transition appears in and has the lowest weight. Since identified trivial transitions do not represent any load changes, they are unneeded and it is thereby a simplification task to remove them entirely from all closure rules.

A further simplification step is a refactoring step which allows a closure rule of lower weight to eliminate its transitions from a longer-path-higher-weight rule as in the pseudo code for method `refactor()`. When smaller patterns of transitioning loads may appear in longer closure rules, the smaller patterns are strengthened in a parallel resistance model manner as they "eat" the pattern from the longer rule. This also simplifies longer rules so that further useable patterns may emerge.

Lastly, all closure rules are then reduced once more so that every rule does not have all of its transitions better explained in stronger rules. In this case, the rule has no value since its transitions appear in stronger rules, and it can therefore be deleted, as shown in pseudocode for `bestExplanation()`.

```
def strongestRule(transition, rules):
   for rule in rules:
      if transition in rule.transitions: return rule

def removeTrivials(rules):
   sort(rules)
   for rule in rules:
      if rule.length == 1:
        if rule == strongestRule(rule.transitions[0],rules):
           rule.transitions[0].mark_trivial()

def refactor(rules):
   sort(rules)
   for i,sub_rule in enumerate(rules):
      for k,source_rule in enumerate(rules):
         if k > i and source_rule.contains(sub_rule):
            source_rule.remove(sub_rule)
            sub_rule.weight = 1/(1/source_rule.weight +
                                 1/sub_rule_weight)

def bestExplanation(rules):
   sort(rules)
   for rule in rules:
      is_useless_rule = True #Assumption
      for transition in rule.transitions:
         if rule ==  strongestRule(transition):
            is_useless_rule = False #Contradiction
            break
      if is_useless_rule: rule.to_delete = True

def RMS(t1, t2):
   return sqrt(avg(sumsq(t1,t2)))

def findMirrors(transitions):
   for t1 in transitions:
      for t2 in transitions:
         if not t1 == t2 and RMS(t1,t2) < 2*cluster_radius:
            addClosureRule(t1, t2)
```

Fig 1. *Python pseudocode for methods of simplifying closure rules.*

Beyond reducing the set of closure rules to a smallest, most-informative subset, sometimes transitions do not appear to be in behavioral patterns as would be found via cycles in the STEC graph. This may happen when there is a very long cycle that was not discovered, due to expenses of computational costs, or when a load behaves unusual to its normative behavior. In these cases, we use a mirror process which finds pairs of opposite signed transitions such that their sum profile has an RMS distance less than the clustering diameter. These mirror rules are not weighted as normal closure rules, but they are sorted separately based on their RMS distance proximity.

The finished closure rules result in a list of relationships between transition clusters that are associated with one or more loads. This approach also has the ability to identify transitions that represent a combination of loads such as when two unique loads turn on independently but both turn off at the same point in time. The next step then, is in using the resulting finalized closure rules as a means of identifying individual loads.

V. Load Detection

Here, we describe the process for using our simplified closure rules as a means of detecting loads. First, closure rules are sorted in ascending order by weight. An automated process builds relationships between transitions and loads based on the following definitions.

Core Rule: a lowest weight closure rule of length 2 (a positive and negative transition, i.e. a 1x1 rule) in which neither transition is included in a lower weight rule.
Cousin Rule: a 1x1 rule of length 2 that best explains one of its transitions, but the other is better explained in a core rule.
Combination Rule: a rule with length greater than 2 that affects multiple loads; the notation MxN indicates that the rule has M on transitions and N off transitions.
Singular Transition: within any rule, the singular transition is the '1' of its type (1xN or Mx1).
Component Transition: the other transitions which lie in the side of the 'M' or 'N'.

The transition-to-load mapping process runs until its methods can no longer find any further useful closure rules. Hence, the output is a dictionary that defines as many transitions as possible. This dictionary is then used in a later step to map the transition load changes to steady states. Since this step is beyond the scope of this paper, we omit its details in favor of submitting more detail on the performance metrics used to evaluate the goodness of our graphical approach.

VI. Performance Metrics for Closure Rule Extraction

Without assumptions to the type of load, useful summary metrics include: *%solved-non-trivial-transitions* and *%solved-sum-abs-transition-power*. Threshold values may be optimized to maximize these terms. We computed these metrics for a BLUED case study, as seen in Table I. In general, reducing event detection and clustering thresholds improves the detection of smaller appliances, however this comes at the expense of increased computational costs and reducing the repeatability of closure rules for less frequent loads.

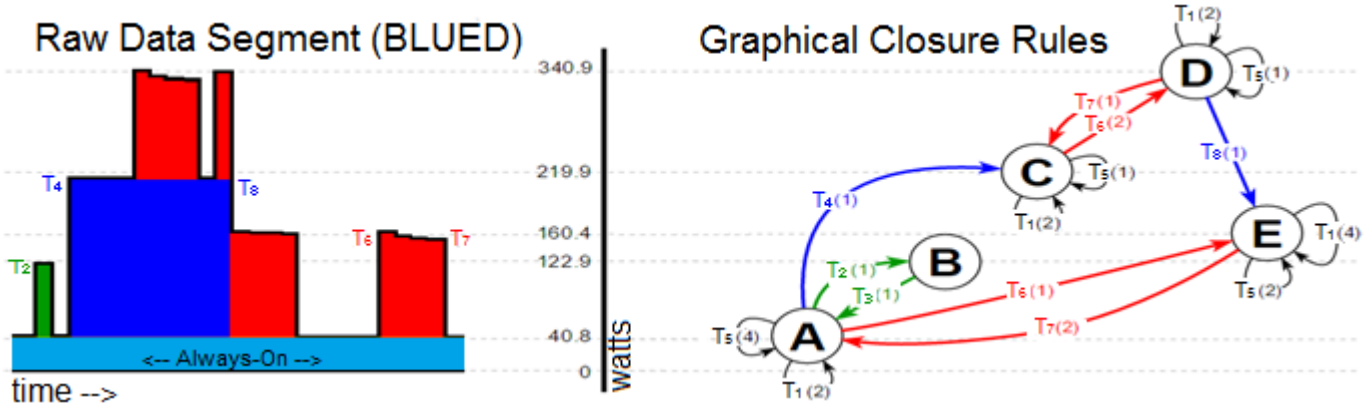

Fig 2. *Power time series (left) transformed to a multidigraph (right). In this diagram, steady state clusters are nodes denoted with letters and transition clusters are edges denoted by T_i. Numbers in parentheses represent the counts profiles contained within the transition cluster, indicating multi-edges. The vertical axis represents watts. Colored edges represent loads.*

The load creation process searches for core rules (see `findCoreLoads()` in Fig 3), in which both transitions are undefined in rules of lesser weight. When we encounter 1x1 rules in which one or more of its transitions were already defined, then this gives us cousin rules that add transitions to existing load definitions. We keep track of how many steps it takes for a transition to reach a core rule. To prevent over-chaining of cousin rules, we only allow a rule to be a cousin to a core if it has not chained more than two steps.

Within any closure rule, it is expected that the sum of both sides (in transition power space) is roughly equal to zero. This allows us to manipulate cases of rules where there exists only one unknown transition in the rule, algebraically.

Consider a 2x1 rule with three transitions (T1, T2 and T3), and that transitions T1 & T3 are known, but not T2. Dictionary notations are used to document the sparse vector state change (value) of each load (key). A load count of {L1: *x*, L2: y} implies that load L1 turns on *x* times and load L2 turns on y times. If T1 = {L1:1} and T3 = {L2:1}, then we compute, algebraically, that T2 = {L1: -1, L2: -1}. Thus, T2 is a combination of two loads turning off simultaneously.

```
loads = []
rules = gatherClosureRules() #See Section III
simplifyClosureRules(rules) #See Section IV

def createLoad(transitions):
  load = len(loads)
  for t in transitions:
    if t.is_positive: t.loadcount = {load: +1}
    else: t.loadcount = {load: -1}
  loads.append(transitions)

def findCoreLoads(rules):
  for rule in rules:
    if rule.length == 2 == countUnknowns(rule.transitions):
      if rule == strongestRule(rule.transitions[0]):
        if rule == strongestRule(rule.transitions[1]):
          createLoad(rule.transitions)

def solveRule(transitions):
  unknown = getUnknownTransition(transitions)
  knowns = getKnownTransitions(transitions)
  unknown.loadcount = negative(dictSum(knowns))

def solveComboRules(rules):
  for rule in rules:
    if rule.length > 2:
      singular = getSingularTransition(rule.transitions)
      components = getComponentTransitions(rule.transitions)
      if countUnknowns(singular, components) == 1:
        solveRule(rule.transitions)
```

Fig 3. *Python pseudocode for methods of mapping transitions to loads.*

VII. Individual Load Consistency Metrics

Load Closure Rule Weight (CRW) for a load is the weight of the core rule defining the load, and should be minimized to indicate that the cycles have been properly detected. Users can assess the consistency of one disaggregated load with respect to another based on these values.

Transition Imbalance (TI) describes the relative difference in the number of defined off and on transitions for a load. If the count of offs and ons do not agree, then there are either false positive or false negative associations between the load and the transitions. This term represents a minimum of the classification error. TI is calculated as a percentage difference: $|num\_on - num\_off| / (num\_on + num\_off)$.

Loads isolated with closure rules are binary in that associated transitions turn the load on or off. In some cases, continuously variable devices may be detectable when the base level on and off transitions form a closure rule. Multistate loads as described by [1] are separated into their individual components (e.g. an air conditioner's fan and compressor). This approach does not assume or require that all loads are single count (i.e. that only one load with the defining transitions exists on each circuit). Banks of identical lights, roof top air conditioners, and computers in offices are examples of loads that may be multicount. A load will have one more state than its number of instances on a circuit.

Single Count Error (SCE) is the fraction that a load sequentially changes state in the same direction to the total number of load transitions. Single count loads should have a minimum of these types of errors. Valid multicount loads will have a small imbalance between the counts of on-on (on transition followed by another on transition) and off-off events. SCE is computed as a percentage difference: $|num\_on\_on - num\_off\_off| / (num\_on\_on + num\_off\_off)$.

VIII. Results

Results are given in Table I and Table II for our case study using the BLUED dataset [16] (only phase A), with an event threshold of 2W/s across 3s and clustering fixed radius of 24 watts. The loads discovered are compared with the ground truth results published by [21-22].

Table I shows some basic summary statistics for our results in working with BLUED. In all, our process found five reliable loads, of which three of them we could confidently match to ground truth data referenced in [21-22]. In Table II, the Event Error column refers to the percent difference in the number of ground truth events and the number of events attributed to a load via our process. For our three loads, the event error was less than 3%, indicating that our results are fairly reliable. For the fridge load, we note that the event error was due to missing events which could be defroster events identified in a separate, less reliable load.

In total, 38 of the 39 transition clusters were defined. The %Solved Non-Trivial Transitions row of Table I is at 98.4%, leaving just a single transition cluster and its profiles as undefined due to a lack of frequency to warrant a satisfying closure rule. When we consider the absolute transition power of these solved transitions, 94.2% of the power was solved, suggesting that the one unsolved transition was also a very high-powered transition. Given that very little energy is consumed by what we were unable to disaggregate, these results indicate very strong performance with detecting loads for BLUED.

In Table II, we note that the 3% single count error associated with the fridge load (which amounts to 9 events that were improperly labeled) could have been due to the presence of a load with similar power. With a visual observation of the raw data, we can indeed see that it would be easy to make the same mistake. On the other hand, the chopper and compressor loads, which are very unique and infrequent loads, had zero errors.

Mapping the transitions to steady state periods to fully disaggregate the energy consumption is beyond the scope of this paper. However, optimization and pattern recognition methods have proved to be effective once load features have been inferred [23-25]. As future work, we are looking to improve upon such techniques while utilizing our graphical closure rule approach.

TABLE I: Summary Statistics for BLUED phase A.

#Non Trivial Events	904
#Steady State Clusters	35
#Transition Clusters	39
#Unique STEC Edges	155
#Unique Cycles	27
#Loads	5
%Solved Non-Trivial Transitions	98.4%
%Solved Sum Abs Transition Power	94.2%

TABLE II: Statistics for top three comparable loads from BLUED phase A. Event Error is the fraction of estimated to actual ground truth number of events. SCE, CRW and TI are defined in Section VI.

Load	*Event Error*	*Single Count Error (SCE)*	*Closure Rule Weight (CRW)*	*Transition Imbalance (TI)*
Fridge	0.024	0.030	0.004	0.008
Chopper	0.000	0.000	0.171	0.000
Compressor	0.000	0.000	0.478	0.000

IX. Conclusions

We present an unsupervised method for NILM load isolation using graph theory cycle detection. The approach extends the ability of matching 'on' transitions with 'off' transitions that are negatively related since transitions are linked through cycles involving multiple loads. An elimination factorization reduces the transitions in the cycles to represent independent sets of loads. Combination transitions are then identified and defined in terms of loads with stronger cycle weight. Consistency metrics are proposed that yield a relative or absolute indication of the quality of fit without the existence of ground truth data. From our results, we show that this process can attain an accuracy with error rate less than 3% on the BLUED dataset when compared to ground truth.

REFERENCES

[1] Zoha, Ahmed, Alexander Gluhak, Muhammad Ali Imran, and Sutharshan Rajasegarar. "Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey." Sensors 12, no. 12 (2012): 16838-16866.

[2] Bonfigli, Roberto, Stefano Squartini, Marco Fagiani, and Francesco Piazza. "Unsupervised algorithms for non-intrusive load monitoring: An up-to-date overview." In Environment and Electrical Engineering (EEEIC), 2015 IEEE 15th International Conference on, pp. 1175-1180. IEEE, 2015.

[3] Jin, Yuanwei, Eniye Tebekaemi, Mario Berges, and Lucio Soibelman. "Robust adaptive event detection in non-intrusive load monitoring for energy aware smart facilities." In ICASSP, pp. 4340-4343. 2011.

[4] Laasch, Frederik, Alain Dieterlen, and Dirk Benyoucef. "Event detection using adaptive thresholds for non-intrusive load monitoring." BW-CAR| SINCOM 50 (2015): 63.

[5] Barsim, Karim Said, Roman Streubel, and Bin Yang. "Unsupervised Adaptive Event Detection for Building-Level Energy Disaggregation." Proceedings of power and energy student summit (PESS), Stuttgart, Germany (2014).

[6] Kolter, J. Zico, and Tommi Jaakkola. "Approximate inference in additive factorial hmms with application to energy disaggregation." In International conference on artificial intelligence and statistics, pp. 1472-1482. 2012.

[7] Elhamifar, Ehsan, and Shankar Sastry. "Energy Disaggregation via Learning Powerlets and Sparse Coding." In AAAI, pp. 629-635. 2015

[8] Guo, Zhenyu, Z. Jane Wang, and Ali Kashani. "Home appliance load modeling from aggregated smart meter data." Power Systems, IEEE Transactions on 30, no. 1 (2015): 254-262.

[9] Pochacker, Manfred, Dominik Egarter, and Wilfried Elmenreich. "Proficiency of Power Values for Load Disaggregation." Instrumentation and Measurement, IEEE Transactions on 65, no. 1 (2016): 46-55.

[10] Figueiredo, Marisa, Bernardete Ribeiro, and Ana de Almeida. "On the optimization of appliance loads inferred by probabilistic models." In Proceedings of the 2nd International Workshop on Non-Intrusive Load Monitoring. 2014.

[11] Egarter, Dominik, and Wilfried Elmenreich. "Autonomous load disaggregation approach based on active power measurements." In Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on, pp. 293-298. IEEE, 2015.

[12] Kolter, J. Zico, and Matthew J. Johnson. "REDD: A public data set for energy disaggregation research." In Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA, vol. 25, pp. 59-62. 2011.

[13] Egarter, Dominik, Manfred Pöchacker, and Wilfried Elmenreich. "Complexity of power draws for load disaggregation." arXiv preprint arXiv:1501.02954 (2015).

[14] G.W Hart, "Nonintrusive appliance load monitoring," Proc. IEEE, vol. 80, no. 12, pp. 1870-1891, Dec. 1992

[15] Leeb, Steven B., Steven R. Shaw, and JJames L. Kirtley Jr. "Transient event detection in spectral envelope estimates for nonintrusive load monitoring." Power Delivery, IEEE Transactions on 10, no. 3 (1995): 1200-1210.

[16] Anderson, Kyle D., Mario E. Bergés, Adrian Ocneanu, Diego Benitez, and José MP Moura. "Event detection for non intrusive load monitoring." In IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society, pp. 3312-3317. IEEE, 2012.

[17] Goncalves, Hugo, Adrian Ocneanu, Mario Berges, and R. H. Fan. "Unsupervised disaggregation of appliances using aggregated consumption data." In The 1st KDD Workshop on Data Mining Applications in Sustainability (SustKDD). 2011.

[18] Streubel, Roman, and Bin Yang. "Identification of electrical appliances via analysis of power consumption." In Universities Power Engineering Conference (UPEC), 2012 47th International, pp. 1-6. IEEE, 2012.

[19] Johnson, Donald B. "Finding all the elementary circuits of a directed graph." SIAM Journal on Computing 4, no. 1 (1975): 77-84.

[20] Hagberg, Aric A., Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

[21] Giri, Suman, and Mario Bergés. "An energy estimation framework for event-based methods in Non-Intrusive Load Monitoring." Energy Conversion and Management 90 (2015): 488-498.

[22] Anderson, Kyle, Adrian Ocneanu, Diego Benitez, Derrick Carlson, Anthony Rowe, and Mario Berges. "BLUED: A fully labeled public dataset for event-based non-intrusive load monitoring research." In Proceedings of the 2nd KDD workshop on data mining applications in sustainability (SustKDD), pp. 1-5. 2012.

[23] Liang, J.; Ng, S.K.K.; Kendall, G.; Cheng, J.W.M. Load signature study Part I: Basic concept, structure, and methodology. IEEE Trans. Power Del. 2010, 25, 551–560.

[24] Suzuki, K.; Inagaki, S.; Suzuki, T.; Nakamura, H.; Ito, K. Nonintrusive Appliance Load Monitoring. Based on Integer Programming. In Proceedings of SICE Annual Conference, Tokyo, Japan, 20–22 August 2008; Volume 174, pp. 2742–2747.

[25] Marchiori, A.; Hakkarinen, D.; Han, Q.; Earle, L. Circuit-level load monitoring for household energy management. IEEE Pervas. Comput. 2011, 10, 40–48.