Non-Intrusive Load Monitoring

Hugo Gonçalves
Electrical and Computer Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213
hgoncalv@andrew.cmu.edu

Adrian Ocneanu
Civil and Environmental Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213
aocneanu@andrew.cmu.edu

Aditi Pandya
Electrical and Computer Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213
apandya@andrew.cmu.edu

Manuel Tragut
Electrical and Computer Engineering
Carnegie-Mellon University
Pittsburgh, PA 15213
mtragut@andrew.cmu.edu

Abstract

Non-Intrusive Load Monitoring (NILM) is a technique that determines the load composition of a household through a single point of measurement at the main power feed. Here we presented an unsupervised approach to determine the number of appliances in the household, their power consumption and the state of each one at any given moment.

1 Introduction

Non-Intrusive Load Monitoring (NILM) is a technique that determines the load composition of a household through a single point of measurement at the main power feed [1]. The current focus of NILM is the disaggregation of load states by means of supervised learning algorithms that use transition signatures. Recorded signatures of appliances are matched with real-time power readings for this purpose [2].

In this project, we attempted to identify an unsupervised learning alternative for load disaggregation, an example of Blind Source Separation (BSS). Given an observation dataset $\boldsymbol{O}$ composed of a variety of overlapping sources $\boldsymbol{S}$, we try to decompose it into the constituent sources by using a weighted sum $\boldsymbol{W}$ of individual observations. Our goal is to iteratively explain $\boldsymbol{O}$ in terms of a 'best-fit' solution from an approximated Weight Matrix $\boldsymbol{W}$ and the Source Matrix $\boldsymbol{S}$. In particular, the source matrix is constituted by the instantaneous power consumption of the appliances (these act as basis that do not have the same norm) and the weight matrix should be filled with binary data, since it is considered that appliances are either ON or OFF.

2 Methods

The power signal is mainly composed of step-up/down events corresponding to the switching on and off of appliances. Due to the sparse variation in the signal to be source separated, the approach taken was to first detect these steps in the power signal, capture their power transition (both active and reactive) and use it to automatic cluster the events. The clustering procedure returns the number of possible clusters (one for each type of appliance) and the center of the clusters, corresponding to the power of the appliances. The centers of the clusters are provided to the source reconstruction algorithm to explain, at each moment, which sources are active.

2.1 Event detector

In order to detect each event, a moving average calculates the average of the power $(P,Q)_{avg}$ of the last n samples. When the sample $n+1$ exceeds the boundaries defined by $\|(P,Q)_{n+1} - (P,Q)_{avg}\| < k \cdot \sqrt{\|(P,Q)_{avg}\|_1}$ where k is a parameter, a new event is registered and the moving average is reinitialized. The number of samples accumulated for an event must be above n_{min} to be accepted, to ensure that the moving average is reliable and avoid spurious events that last only a few samples. All the point that were part of the same moving average window is defined as a *plateau*.

2.2 Clustering procedure

The clustering procedure should determine automatically how many clusters are in the data (corresponding to the number of different appliances) and the center of the clusters (corresponding to the power of those appliances). Two types of clustering methods were investigated: clustering based on genetic algorithms and hierarchical agglomerative clustering.

2.2.1 Genetic and K-means Clustering Algorithm

The genetic algorithm used in this project was based on [3] and works as follows: each chromosome represents a possible clustering solution, indicated by K_{max} numbers N_i between 0 and 1, and K_{max} points $(P_c,Q_c)_i$ in the search space (this case 2-D) representing the center of the clusters C_i. K_{max} is the maximum number of clusters that the algorithm can find. If a number N_i is greater than 0.5, then the cluster C_i is part of the solution. The number of chromosomes is given as a parameter. Initially, N_i and $(P_c,Q_c)_i$ for all $i \in [1,\dots,K_{max}]$ are randomly selected, N_i, and $(P_c,Q_c)_i$ evenly through the search space. K-means is applied to each chromosome, providing an optimal solution given its initial conditions. For each chromosome, an offspring chromosome is created by changing its clusters. That is done by randomly generating $2K_{max}$ numbers between 0 and 1: For any of the first K_{max} below Cr, N_i is changed to a new value between 0 and 1; for any of the last K_{max} below Cr, $(P_c,Q_c)_i$ is changed to the $(P_c,Q_c)_i$ of the *best* chromosome (as defined by the fitness function). This way, both the number and the center of the clusters are changed for an offspring. Cr, the crossover rate, starts as 1 and decreases after each iteration in a linear fashion, until it stops at Cr_{min}. K-means is again applied to each offspring and both solutions (the parent and the offspring) are measured by a fitness function. The better result prevails, while the other is removed. The fitness function is an adaptation to the CS measure:

$$CS = \frac{\sum_{k=1}^{K} \frac{1}{M_k} \sum_{X_i \in C_k} \max_{X_q \in C_k} d(X_i, X_q)}{\sum_{i=1}^{K} \min_{j \neq i} \hat{d}(m_i, m_j)}$$

where K is the number of clusters in the solution, M_k is the number of points in cluster k, X_i is an event, m_i stands for $(P_c,Q_c)_i$, $d(X_i,X_q)$ is the Euclidian distance between X_i and X_q, and $\hat{d}(X_i,X_q) = d(X_i,X_q)/\sqrt{(X_i+X_q)/2}$. This cycle is repeated t_{max} times and at the end the solution with the best fitness function is selected.

2.2.2 Hierarchical Agglomerative Clustering

Hierarchical clustering algorithms can be categorized as either top-down or bottom-up [5]. Bottom-up algorithms treat each data point as a singleton cluster at the outset and then successively merge pairs of clusters until all the clusters have been merged into a single cluster that contains all data points. Bottom-up hierarchical clustering is therefore called hierarchical agglomerative clustering or HAC. Merge operation in HAC is assumed to be monotonic, which means at each step the best merge available is taken.

An important component of a clustering algorithm is the distance measure between data points. If the data points are all in the same physical units then, a simple Euclidean distance metric is sufficient to successfully group similar data points.

Agglomerative clustering algorithms that were tested for this work are:

1) Single Link [6] – two clusters having minimum distance between their closest datapoints are merged together.

$$D_{sl}(C_i, C_j) = \min_{x,y} \{ d(x,y) \mid x \in C_i, y \in C_j \}$$

2) Complete Link [6] – At each step, two clusters having minimum distances between the two farthest points are merged together.

$$D_{cl}(C_i, C_j) = \max_{x,y} \{ d(x,y) \mid x \in C_i, y \in C_j \}$$

Agglomerative clustering results are represented in the form of a dendrogram. Dendrogram is a tree structure, that is formed by decomposing datapoints into several levels of nested partitions. The final clustering of datapoints are obtained by cutting the dendrogram at the desired level (threshold), which gives clusters formed by each the connected component. Setting the threshold level defines the termination condition for the agglomerative clustering.

2.3 Coincidence detection

Some events detected might be the sum of two coincident events and the transition is the combination of the power of both appliances. To detect this type of situations, a post-processing was done to the result of the clustering. Figure 1 depicts this procedure. For each cluster i, it was assumed that it has a 2-dimensional gaussian distribution and the mean $(P_c, Q_c)_i$ and standard deviation (SD) of each cluster was extracted. Additionally, each cluster has an asymmetry factor defined by $af(i) = (n_{i(on)} - n_{i(off)})/(n_{i(on)} + n_{i(off)})$, where $n_{i(on)}$ and $n_{i(off)}$ are the number of ON and OFF events, respectively.

For each cluster i, with its events ordered by time, the time spans between two consecutive ON or OFF events were found (cluster with a possible missing event 1 in Figure 1). Within each time span, the algorithm tries to find an event that could be mixing the missing OFF or ON with other coincident event. To do that, the intra-cluster probabilities (probability of an event belonging to its cluster, given the mean and SD of the 2-D Gaussian defined for the cluster) of the events X_j from all the other clusters, within each time span, is defined as P_j. The other coincident event should be found in the cluster whose center is closest to $X_k = X_j - (P_c, Q_c)_i$ (missing event 2 in Figure 1). The probability that such point belongs to cluster m is defined as $P_k(m)$. A metric that measures how probable X_j is a mixing event is defined as

$$M(i,j,k,m) = \frac{af(i)af(m)P_k(m)}{P_j}$$

,i.e., the value increases as the mixing event gets further away from the center of the cluster it belongs (indicative that it does not belong there), the asymmetry factor of the cluster of the first missing event increases (there should be missing events in the cluster), the asymmetry factor of the cluster of the second missing event increases, and when the estimated missing event is closest to the center of some cluster (the relation between missing and mixing events match). Values of M higher than M_{min} are considered as mixing/missing events and the mixing event is split into the corresponding missing events.

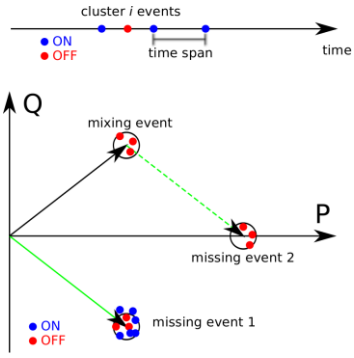


Figure 1 Coincidence detector

2.4 Source reconstruction

2.4.1. Matching pursuit

This greedy algorithm is based on a 'match the most in least components' approach of an observation, when unfolded into individual sources [4]. At each iteration of the algorithm, an observation point will be matched against the closest source, in terms of Euclidian distance, then it will have that value subtracted, in a vector form, and the remaining vector will be matched, in a subsequent iteration, against all other sources (the ones plotted in Figure 2) . For the algorithm to converge, a 'drain' source located at the origin (0,0) was introduced. Put differently, the algorithm stops when a residual vector is closer to 'noise' (the drain) than to any sources present.

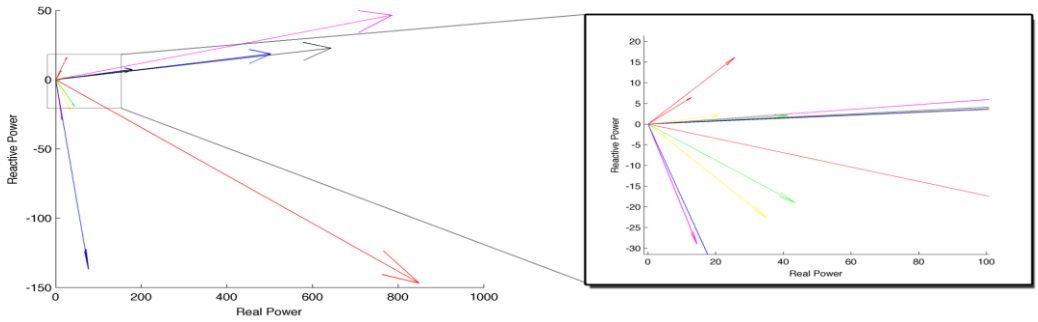


Figure 2 Recovered sources plotted on a PQ plane

There are three main considerations to be made here, representing adaptations made to the original matching pursuit algorithm:
i) the matching is performed in terms of Euclidian distance ($[P_{obs},Q_{obs}]$-$[P_{closest},Q_{closest}]$) and not using a dot product between observation/residual and sources
ii) by performing the operation above, a weight of 1 is assigned by default to each source encountered in an observation, as opposed to a real value that would be the return by a dot product
iii) this operation is performed only on the first element of a plateau, on the assumption that the result obtained on the very first point will always apply to the following ones, throughout the rest of the current plateau (this assumption was made to ease the computational load of the algorithm but nevertheless proves its validity almost intuitively).

2.4.2. Integer programming

In comparison to the iterative greedy approach used in matching pursuit, integer programming tries to solve the following optimization problem:

$$\min_{w} \|w\|_1 \text{ subject to: } s(i) - \Delta s(i) < Sw < s(i) + \Delta s(i)$$

where w is the weight of each source, S is the dictionary (matrix with power of appliances) and $s(i)$ the signal value at time instance i. Furthermore we set the tolerance $\Delta s(i)$ proportional to the standard deviation of the cluster closest in absolute power to $s(i)$. The proportionality constant can be used to tune the algorithm, a higher value will make it more likely that less number of sources can be used to construct the signal, too small values mean that noise would be approximated with a small value power source. Best results were found for a value of 2-4.

3 Results

3.1 Event detector

The data is a 6-day long power reading (active and reactive) with a 0.3s period. With $k = 0.8$, $n_{min} = 10$, and a moving average of 20 samples, the event detector found 1156 events, giving 192 events/day. The power readings are shown in Figure 3, together with the events detected. Due to the lack of ground truth, any manual labeling would still be ambiguous, since many transitions in the power readings are not clear if they are steps. Manual inspection shows that most of the noticeable events are detected, although some false positives and false negatives exist.

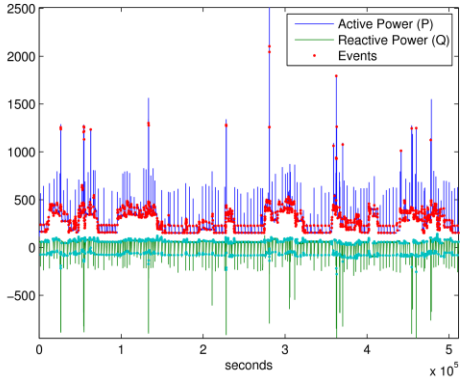


Figure 3 Power readings and events detected

3.2 Clustering

The events were clustered according to two different clustering methods.

The first method, based on genetic algorithms, has a stochastic component and does not produce always the same result. Thus, for 50 trials, the average $\pm$ standard deviation number of clusters was 16.7 ± 2.57 and one example with 16 clusters is plotted in Figure 4. The parameters were set to $K_{max} = 30,\ t_{max} = 50, Cr_{min} = 0.5$ and 20 chromosomes.

The hierarchical clustering algorithms use Euclidean distance as the clustering criteria. Since all the

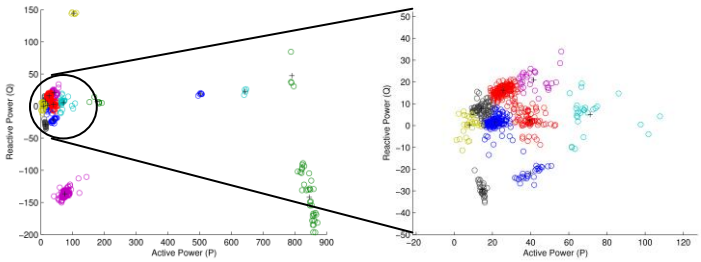
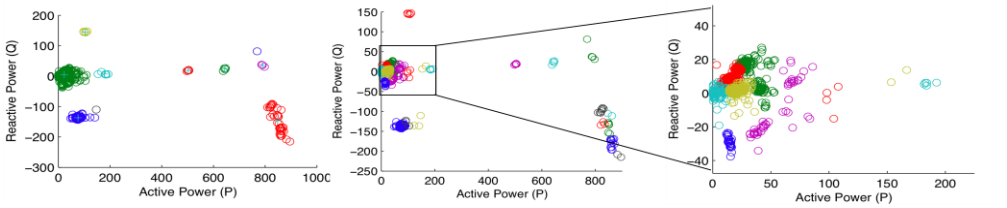


Figure 4. Genetic/K-means clustering example results

data points are successively merged into a single cluster by the algorithm, we need to split this cluster in order to get the final number of clusters. As explained before, we essentially cut the dendrogram at a certain threshold to get the clusters formed by each connected component. We use a threshold of 0.07*max(**d**), where **d** is a vector of distances between all the clusters.



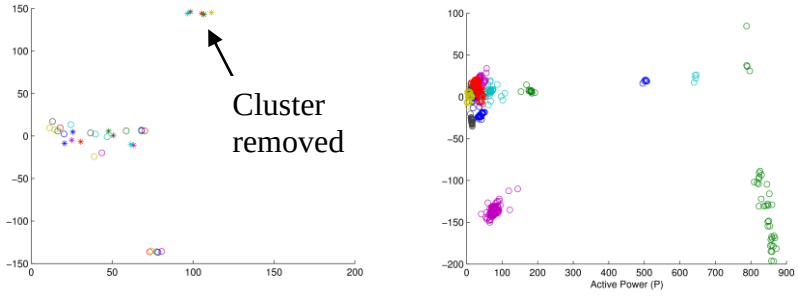
1) Single Link **2) Complete Link**

Figure 5 Single Link vs. Complete Link

We can see from the plots above that clusters formed by single link tend to have a chaining effect as compared to complete link clustering that shows circular clusters. There are 13 clusters formed as a result of the threshold set on the distances between the datapoints.

3.3 Coincidence detection

It is known for a fact that a particular cluster in the results is made of mixing events, because it was found consistently through visual inspection. The coincidence detector is able to detect and split all the events in that cluster. The improved clustering is shown in Figure 6. Some other points were considered mixing events, but visual inspection leaves doubts if they are or not.



a) *b)*

Figure 6 Coincidence detection a) circles: missing events created, stars: mixing events removed b) Clustering after coincidence detection

3.4 Source reconstruction

3.4.1. Matching Pursuit

Based on the sources provided from the clustering (13 individual appliances), the reconstruction (Figure 7) was performed in two steps:

i) explain each first element of a plateau as a sum of appliances, while altering their states;

ii) recompose the signal using these 'presence' values and the (P, Q) value of each source.

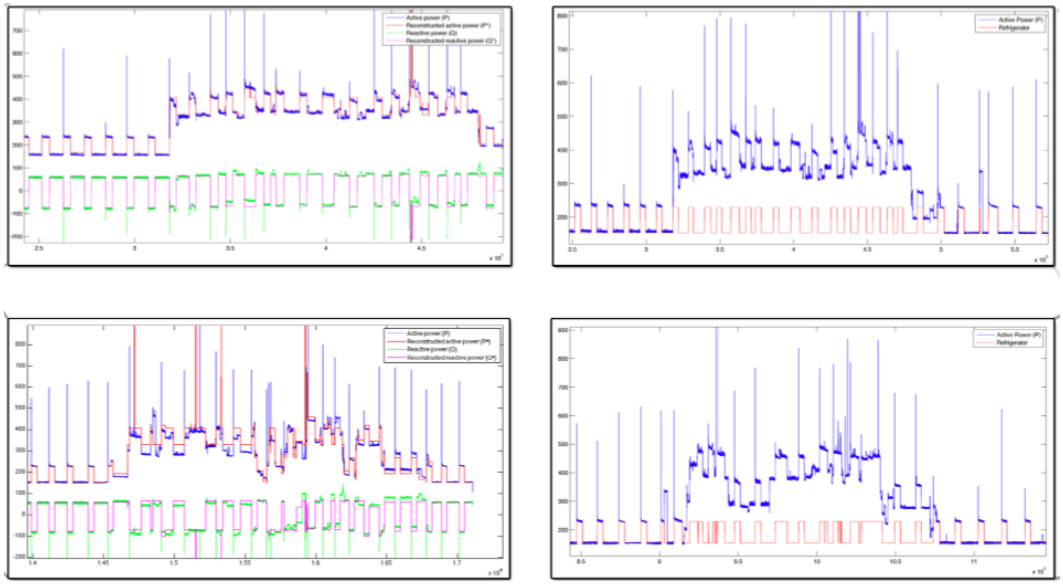


Figure 7. Signal reconstruction (left – reconstructed against observed signal; right – one appliance matched against the observed signal VS. up – a good match; bottom –a bad match)

3.4.2. Integer programming

For computational tractability, we used the binary programming algorithm, as this would give us equally valid results (Except for appliances that have more than one instance running at any point in time, here duplications in the dictionary can help). Results are shown in Figure 8.

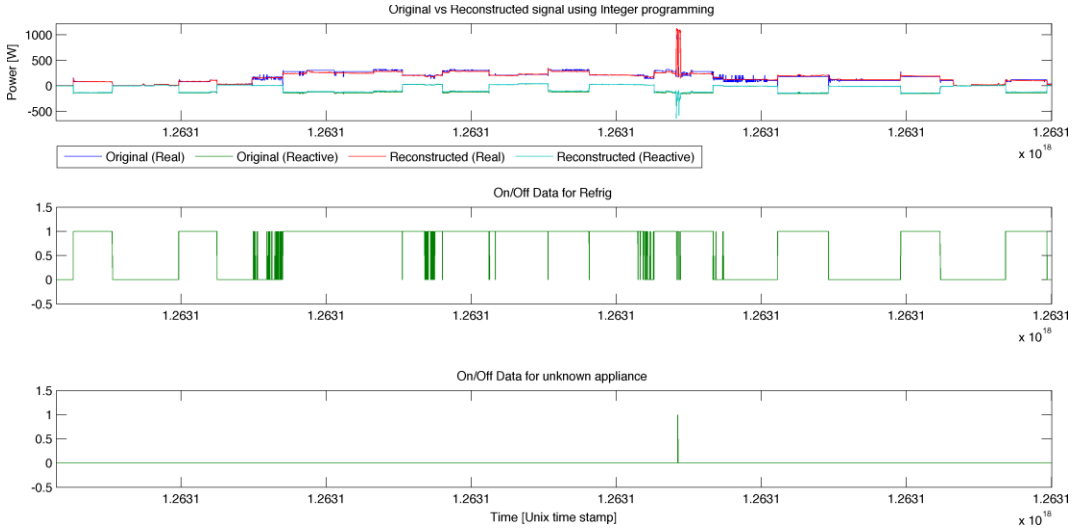


Figure 8. Original and Reconstructed Signal with Integer (Binary) Programming

4 Discussion

This work can be divided in three parts: detection of events, detection of appliances and detection of the states of appliances.

Detection of events was the easiest part and it can be seen by visual inspection that most events were detected correctly. The missing of some events is not relevant for clustering, unless the occurrence of such events is sparse.

Clustering had some challenges, mainly due to the overlap of points in different clusters (especially in low power appliances) and difference in scatter between appliances (high-power appliances had more scatter). Agglomerative algorithms had difficulty in dealing with these properties, while the genetic algorithm, although being more precise on average, had some deviation between solutions.

The source reconstruction algorithms used did a good job in reconstructing signal. However, the real objective was to find the change of states of the sources. Taking the example of the most noticeable and frequent appliance (we believe it to be a refrigerator), it can be seen that sometimes it toggles very frequently, because a change in power may be best explained (in terms of sparsity) by turning it ON or OFF, although it may not be the case. These considerations apply to both methods studied.

5 Conclusion

We distinguish between ON/OFF states of the household appliances, and form clusters based on the active (P) and reactive power (Q) readings of the appliances.

We have tested various clustering techniques based on agglomeration and genetic algorithms for this purpose. We attach a power consumption weight to each of these appliance clusters to ultimately reconstruct the individual energy consumption of each appliance.

We explored the use of a simple Matching Pursuit algorithm and classical integer programming for this purpose. These algorithms try to minimize the signal reconstruction error, while our objective is to find the real profile of the states of each appliance.

We conclude that although they accurately reconstruct the signal, the profiles found are apparently different from the reality. This could have been more frequent for smaller appliances, as they are comparable in magnitude to the noise level, and the matching pursuit algorithm would indubitably go through them.

The lack of ground truth of the data makes it difficult to use meaningful metrics for evaluation of our solution.

References

[1] Hart, G., (1992). Nonintrusive appliance load monitoring. Proceedings of the IEEE, 80(12), p.1870-1891.

[2] Mario Berges et. al (2008) Training Load Monitoring Algorithms on Highly Sub-Metered Home Electricity Consumption Data, Tsinghua Science & Technology, 13(1): pp 406-411

[3] Das, S. & Abraham, A. & Konar, A. (2008) Automatic Clustering Using an Improved Differential Evolution Algorithm. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 38 (1) : 218-237

[4] http://www.scholarpedia.org/article/Matching_pursuit

[5] http://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-agglomerative-clustering-1.html

[6] http://nlp.stanford.edu/IR-book/completelink.html