Exploiting HMM Sparsity to Perform Online Real-Time Nonintrusive Load Monitoring (NILM)

Stephen Makonin, *Senior Member, IEEE,* Fred Popowich, Ivan V. Bajić, *Senior Member, IEEE,* Bob Gill, *Senior Member, IEEE* and Lyn Bartram, *Member, IEEE*

***Abstract*—Understanding how appliances in a house consume power is important when making intelligent and informed decisions about conserving energy. Appliances can turn ON and OFF either by the actions of occupants or by automatic sensing and actuation (e.g., thermostat). It is, also, difficult to understand how much a load consumes at any given operational state. Occupants could buy sensors that would help, but this comes at a high financial cost. Power utility companies around the world are now replacing old electro-mechanical meters with digital meters (smart meters) that have enhanced communication capabilities. These smart meters are essentially free sensors that offer an opportunity to use computation to infer what loads are running and how much each load is consuming (i.e., load disaggregation). We present a new load disaggregation algorithm that uses a super-state hidden Markov model and a new Viterbi algorithm variant which preserves dependencies between loads and can disaggregate multi-state loads, all while performing computationally efficient exact inference. Our sparse Viterbi algorithm can efficiently compute sparse matrices with a large number of super-states. Additionally, our disaggregator can run in real-time on an inexpensive embedded processor using low sampling rates.**

***Index Terms*—load disaggregation, nonintrusive load monitoring, NILM, energy modeling, hidden Markov model, HMM, sparsity, Viterbi algorithm, sustainability.**

I. Introduction

LOAD DISAGGREGATION is a topic studied by researchers who investigate algorithms that try to discern what electrical loads (i.e., appliances) are running within a physical area where power is supplied from the main power meter. Such physical areas can include communities, industrial sites, office towers/buildings, homes, and even within an appliance. When focusing on homes, this area of research is often referred to as nonintrusive (appliance) load monitoring (NILM or NIALM). NILM research was first published by Hart [1] in 1992. By knowing what loads are running, when they are running, and how much power they are consuming, we can begin to make informed choices that can lead to a reduction in power consumption [2]. For load disaggregation to be practical, it will need to run in real-time, use low-frequency sampling data, and have a high degree of accuracy. A good disaggregator helps provide rich information about loads from a given aggregate meter reading.

S. Makonin and F. Popowich are with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada e-mail: smakonin@sfu.ca and popowich@sfu.ca.

I. Bajić is with the School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada e-mail: ibajic@sfu.ca.

B. Gill with the School of Energy, British Columbia Institute of Technology, Burnaby, BC, Canada e-mail: Bob_Gill@bcit.ca.

L. Bartram with the School of Interactive Arts and Technology, Simon Fraser University, Surrey, BC, Canada e-mail: lyn@sfu.ca.

Manuscript accepted October 7, 2015.

A. Our Contributions

Our main contribution is a disaggregation algorithm that: (1) is agnostic of low-frequency sampling rates ($\frac{1}{3}$Hz, per minute, and per hour) and measurement types (A, W, and Wh); (2) is highly accurate at load state classification and load consumption estimation; (3) can disaggregate appliances with complex multi-state power signatures; (4) is the first hidden Markov model (HMM) solution that preserves dependencies between loads; and (5) can perform computationally efficient exact inference, while other methods only use approximate methods that are computationally more complicated.

The extent of matrix sparsity found in HMMs for house loads is an integral issue that motivated our particular disaggregator. Others have examined matrix sparsity in the past [3]. However, their disaggregators implement complex solutions to take advantage of sparsity that is not always efficient. We present an efficient way to take advantage of sparsity in matrix storage and processing and show how to do this by using a super-state hidden Markov model, which was previously dismissed because of state exponentiality. Our disaggregator presents a new Viterbi algorithm variant, called *sparse Viterbi algorithm* that can efficiently process very large sparse matrices (over eight billion states).

Our disaggregator was first presented at the 2nd NILM Workshop [4] and discussed in Makonin's thesis [5]. Our previous workshop presentation showed only preliminary test results. This paper presents a more optimized algorithm than what was discussed in Makonin's thesis. We can now disaggregate HMMs that have billions of states (>18 loads) as opposed to smaller HMMs with of 1–2 million states (11 loads).

B. Super-State Definition

Our super-state HMM is in all respects a basic HMM. However, we have chosen the prefix term *super-state* because each HMM state is the Cartesian product of the different possible states of each appliance/load we want to disaggregate. In other words, the super-state can be viewed as the house's state. At each time interval, the super-state describes each appliance – whether it is ON or OFF – and if ON, its current operational state. There is a unique super-state for each combination of appliance/load states. For example, if we have two appliances each having two appliance-states there

would be a total of four super-states. Super-state 0 would mean both appliances are OFF. Super-state 1 would mean the first appliance is ON and the second appliance is OFF. Super-state 2 is the opposite of super-state 1. Super-state 3 would mean that both appliances are ON.

C. Paper Organization

The remainder of this article is organized as follows. We first review the recent research in the area of load disaggregation (Section II). We continue with an in-depth discussion of our algorithm presenting the methodology (Section III) and an analysis of complexity and efficiency (Section IV). This is followed by details of our experimental setup and many tests and their accuracy results (Section V) and discussion of the results (Section VI). We conclude with a discussion on significance and limitations (Section VII).

II. Background

Many researchers have published vastly different approaches to load disaggregation. Artificial Neural Networks (e.g., [6]), Support Vector Machines (e.g., [7]), and Nearest Neighbour algorithms (e.g., [8]) have been popular methods for disaggregation in the past. Artificial Neural Networks (ANN) are easy to use. However, both the construction and training of ANNs is arbitrary, and tuning can often result in the convergence on local maxima and overfitting. Support Vector Machines (SVM) use *optimal line separation* between classifications and do not suffer from the same problems of ANNs provided the dataset used is not large. Nearest Neighbour (k-NN) is used to classify unlabeled data that is nearest to each other (based on a distance function). However, this method can be memory intensive having large storage requirements.

A. Factorial HMM

Since 2011, methods that use HMMs have become a focal point for most researchers. HMMs are a natural fit for disaggregation because they have the ability to model time series data and represent the unobservable state of each load. To curtail the state exponentiality problem of HMMs, researchers have used factorial HMMs for disaggregation to lower complexity. For instance, 8 two-state loads would have 256 states (2^8) where a factorial HMM (FHMM) would have 8 chains. Each load is a separate multi-state Markov chain that can evolve in parallel to the others. Since each load is a separate chain, load dependency information is lost. Additionally, there is the added complexity of training these chains [9] with approximation methods because exact inference is not possible [10].

Kim et al. [11] used a combination of four different factorial HMM variants to provide an unsupervised learning technique. They achieved classification accuracies of between 69%–98% (for 10 homes) using their M-fscore accuracy measure. Their results seem to suggest the accuracy of the disaggregator quickly decreases (from 98% to 69%) as more appliances were added for disaggregation and required a high degree of computational power to disaggregate. This work has many issues, including their use of 4 FHMMs to disaggregate, a configuration which would not be able to run online in real-time without a large computational cost.

Kolter et al. [10] used a combination of two FHMM variants (additive FHMM and difference FHMM) using high-frequency sampling. Four of seven loads scored with moderate to high accuracy, but loads such as electronics scored very low. This approach would produce even lower accuracy results on low-frequency data due to signal feature loss.

Kolter's factorial models reduce the state exponentiality problem of HMMs but at the expense of exact inference and the loss of load dependencies. In other words, while trying to avoid the complexity problems, these researchers have introduced new inaccuracies. It is difficult to see these factorial algorithms can run in real-time – which our disaggregator is capable of doing.

Our solution preserves load dependence information and a way to perform exact inference in a computationally efficient manner, which is not possible when using factorial HMM or VAST (discussed below). For example, consider the condition when a heat pump turns ON. The HVAC fan (on a separate breaker) increases its rotation speed and does the reverse when the heat pump turns OFF (as we have observed in our dataset is lost [12]).

B. Combining HMMs

Zia et al. [13] built their models using the sub-metering data for each load. They determined load states and created an HMM for each load by hand, then combined two loads to make a larger HMM. This paper is arguably one of the first papers to make use of HMMs for disaggregation. A two load combinatorial search was performed until the total observed load was processed. Combining and testing for only two loads is not a realistic scenario. Accuracy results were not reported, so it is hard to judge how successful their disaggregator was.

Unlike Zia et al., our disaggregator can determine load states automatically during model building, can combine all loads into one super-state HMM, and does not require the added off-line pattern matching step.

C. Viterbi Algorithm with Sparse Transitions

Zeifman [3], [14] proposed *Viterbi Algorithm with Sparse Transitions* (VAST), a modified version of the Viterbi algorithm, on multiple transition matrices (each a triple of loads). However, he limited the number of internal load states to only two (ON or OFF). Zeifman reported the accuracy of VAST to an average of 90.2% on 9 (simple ON/OFF) loads. Such an approach would require the approximation of a many-state load (e.g., the dishwasher) to be a simple ON/OFF load; however, the majority of modern appliances are many-state. Zeifman used triples of loads to avoid one large sparse transitions matrix, but his transitions matrices could still have zero-probability elements and sparsity in the emission matrix was not dealt with.

Like Zeifman, we take advantage of matrix sparsity, but in a different way. Unlike Zeifman, we can disaggregate loads

with complex multi-state power signatures – not just ON/OFF loads. Additionally, we have provided a more efficient and simple method to do so.

D. Semi-Supervised & Unsupervised Learning

Parson et al. [15] used a variant of the difference FHMM from Kolter et al. [10]. They proposed a disaggregator that would train generic appliance models to specific appliance models. This method shows promise for disaggregating a small number of cyclical type appliances such as fridges and freezers. It requires a lengthly training window making it more suited for running off-line.

To have their solution run online, they have focused on using a cloud computing based solution [15] to perform algorithm execution, which could result in data privacy concerns amongst consumers. While they have demonstrated that there is promise in being able to take general appliance models and actively tune them, this method is limited to cyclical appliances – which has practical limits. Additionally, this algorithm still needs a form of priors which is referred to as a general appliance model. Loads that are more complex, such as HVAC systems, operate quite differently based on make/model and efficiency. Such complexity means that it would not be possible to have a general appliance model for all HVAC systems, unlike fridges and freezers that operate very similarly regardless of the make/model and efficiency level. Our solution, which needs priors, can disaggregate multi-state appliances. We believe this is a more practical solution that can be implemented in houses now to assist homeowners in realizing energy savings.

Recently, Johnson et al. [16] considered using the factorial variant of a hidden semi-Markov model (HSMM) because they provided a means of representing state durations in a load model. Although the authors claimed this was unsupervised learning, they were incorrect [15] because they used labeled data to build the appliance models from the same dataset used to test. Rather than having their algorithm run on the entire dataset, they hand-picked a number of specific segments for testing and evaluation – this does not constitute a real-world scenario. Our evaluations test on the entire dataset.

One of our main goals is to achieve high accuracy scores. We believe disaggregators with active tuning (unsupervised learning) may not work best for occupants. It is our opinion the disaggregator would take too long to learn what loads are in a house. Long learning times cause occupants to lose confidence in the disaggregator's ability to work properly because of incorrect results. However, these problems provide direction and motivation for future work.

III. Methodology

Our disaggregator (Figure 1) first analyzes the sub-metered data from load priors and creates a probability mass function for each. Load states are then determined by quantizing the probability mass function (PMF) further. Each of the load's states is then combined to create a super-state HMM. The super-state HMM is very sparse, and matrices are compressed to take advantage of this. Once the super-state HMM is built, there is no need for further sub-metering. The act of model building creates a super-state HMM. We then take the last times observation and the current observation and use our sparse Viterbi algorithm to disaggregate the state of each load and estimate load consumption. The compression technique we use provides a computationally efficient way to perform exact inference in a way that is both space and time optimized to alleviate the state exponentiality problem HMMs have.

A. Nomenclature

- $\mathbf{A}$ - the transition matrix ($K \times K$)
- $\mathbf{B}$ - the emission matrix ($K \times N$)
- K - the number of whole-house states (or super-states)
- $K^{(m)}$ - the number of states for the m-th appliance
- k_t - the super-state at time t
- $k_t^{(m)}$ - the m-th appliance state at time t
- M - the number of loads/appliances
- N - the number of possible observations
- $\mathbf{P}_0$ - initial prior probabilities vector t
- $\mathbf{P}_t$ - the posterior probability vector at time t
- PMF - probability mass function
- $\Pr[\cdot]$ - the probability of
- $p_{Y_m}(\cdot)$ - the PMF of the m-th appliance
- S - the super-state
- T - the length of time
- X - the hidden load/appliance state
- $x_t^{(m)}$ - the m-th appliance state at time t
- $\hat{x}_t^{(m)}$ - the m-th appliance estimated state at time t
- Y - discrete random variable for current draw
- y_t - the aggregate smart meter current draw at time t
- $\hat{y}_t$ - the estimated aggregate current draw at time t
- $y(\cdot)$ - current draw of the m-th appliance
- $\mathbf{y}_{peak}^{(m)}$ - the PMF current peaks for each state

B. Load Consumption Modelling

The Model Builder (second block in Figure 1) uses the following steps to build a super-state HMM:

1) **LOAD DATA:** Prior to calling Model Builder, a dataset is loaded into memory and split into priors data and testing data. Only the priors are given to Model Builder to build our model. The priors contain the aggregate reading and all sub-meter readings of the loads targeted for disaggregation.
2) **CREATE PMF:** Using the priors build a PFM for each load. Discussed in Section III-C.
3) **QUANTIZE:** For each load, the Model Builder quantizes each PMF to find the load's states and the peak value within that state. Discussed in Section III-D.
4) **CREATE HMM:** Using priors and load, state data provided by Model Builder to build the super-state HMM (Section III-D). Each prior observation's sub-meter data is used to find each load's states that are used to calculated the super-state. The previously observed super-state and the currently observed super-state are used to update the HMM data structures. Once all priors are processed the HMM vectors and matrix rows are then normalized to sum to 1.0.

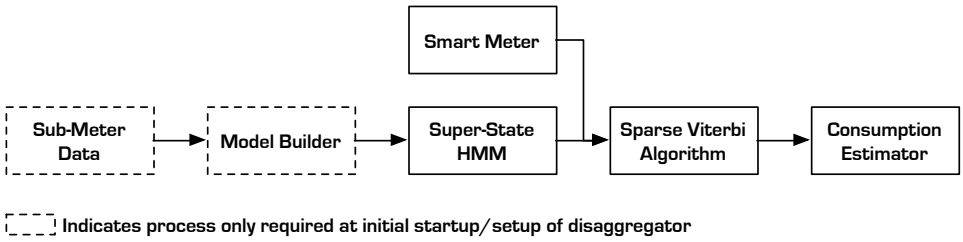

Fig. 1. Block diagram of our disaggregator.

C. Probability Mass Functions

We represent an appliance as a discrete distribution by using a *probability mass function* (PMF). Let there be M independent discrete random variables $Y_1, Y_2, \ldots, Y_M$, corresponding to current draws from M loads. Each Y_m is the current or power measurement of a metered electric load with a PMF of $p_{Y_m}(n)$, where m is the load index $i \in \{1, 2, ..., M\}$, y is a number from a discrete set of possible measurements $y \in \{0, 1, ..., N_m\}$, and N_m is the upper bound imposed by the breaker that the m-th load is connected to. For example, with current measurements (in dA) on a 15A breaker, we would have $N_m = 150$. The PMF $p_{Y_m}(n)$ is defined as follows:

$$p_{Y_m}(n) = \begin{cases} \Pr[Y_m = n], & \text{if } n \in \{0, 1, \ldots, N_m\}, \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

where $\Pr[Y_m = n]$ is the probability that the current/power draw of the m-th load is n. For example, using Table I, for $n = 3$, $\Pr[Y_m = 3] = 0.31$, so the probability of the m-th load drawing 3 dA (i.e., 0.3 A) is 0.31.

D. Super-State HMM

We model a house with M loads as an HMM $\lambda = \{\mathbf{P}_0, \mathbf{A}, \mathbf{B}\}$ having a row-vector of initial prior probabilities $\mathbf{P}_0$ of length K, a $K \times K$ transition matrix $\mathbf{A}$, and a $K \times N$ emission matrix $\mathbf{B}$, where K is the number of whole-house states (or super-states), and N is the number of possible observations. If $t-1$ and t represent the previous and the present time instants, the entries of $\mathbf{A}$ and $\mathbf{B}$ are defined as $\mathbf{A}[i, j] = p(S_t = j | S_{t-1} = i)$, $\mathbf{B}[j, n] = p(y_t = n | S_t = j)$; where S_t is the super-state at time t, and y_t the observation at time t.

The super-state is composed of the states of all M appliances: $S_t = (X_t^{(1)}, X_t^{(2)}, ..., X_t^{(M)})$, where random variable $X_t^{(m)}$ is the internal state of the m-th load at time t. For example, a dishwasher may have 4 internal states that consist of {OFF, WASH, RINSE, DRY}. The total number of super-states is $K = \prod_{m=1}^{M} K^{(m)}$ where $K^{(m)}$ is the number of internal states of the m-th appliance.

As the state of a load changes so too does its power or current draw. In this work, we consider current draw as the observation. Let $y(\cdot)$ be the current draw of the corresponding internal appliance state, so that $y\left(x_t^{(m)}\right)$ is the current draw of the m-th appliance in state $x_t^{(m)}$. For notational convenience, we assume that the current values are non-negative integers, i.e., $y\left(x_t^{(m)}\right) \in \{0, 1, ..., N\}$. In practice, these would not necessarily be integers, but would still be constrained to a discrete set of possible readings of the current meter. The observed measurement at time t from the smart meter is the sum of the current draws of individual appliances:

$$y_t = \sum_{m=1}^{M} y\left(x_t^{(m)}\right) . \quad (2)$$

Model parameters, such as load state probabilities $p(X_t^{(m)} = x_t^{(m)})$ as well as conditional probabilities in $\mathbf{A}$ and $\mathbf{B}$, can be obtained from existing load disaggregation datasets (e.g., AMPds [12], see Section III.A). In general, even though the assumed full set of possible current draws is $\{0, 1, ..., N\}$, most appliances have fewer than N states. To model this for the m-th appliance, we quantize the set $\{0, 1, ..., N\}$ into $K^{(m)}$ bins such that the first bin contains 0 (and corresponds to the OFF state). Other bins are centered around the peaks of the empirical probability mass function of the current draw

$$p_{Y_m}(n) = \Pr\left[y\left(X_t^{(m)}\right) = n\right], \; n \in \{0, 1, ..., N\} , \quad (3)$$

obtained from the dataset. A peak in the PMF is identified when the slope on the left, $p_{Y_m}(n) - p_{Y_m}(n-1)$, is positive, the slope on the right, $p_{Y_m}(n+1) - p_{Y_m}(n)$, is negative, and $p_{Y_m}(n) > \epsilon$, where $\epsilon = 0.00021$ used to ensure that small peaks (noise) are not quantized as states. We chose the value of ϵ by observing the PMFs and finding that any spike that had <=110 out of (524,544) occurrences was not a state.

A simple example of such quantization is given in Table I, where $N = 5$, but only two states are identified after quantization, hence $K^{(m)} = 2$. These states are indexed by $k^{(m)} \in \{0, 1\}$ and are centred around the values $n = 0$ and $n = 3$. The quantized states are denoted $\widehat{X}_t^{(m)}$. The current draws of the quantized states are stored as a $K^{(m)}$-dimensional vector, denoted $\mathbf{y}_{peak}^{(m)}$, whose elements are the locations of the peaks in the original PMF. For the example in Table I, $\mathbf{y}_{peak}^{(m)}[0] = 0$ and $\mathbf{y}_{peak}^{(m)}[1] = 3$. The probability of a given quantized state is simply the sum of probability masses in the corresponding bin, hence for the above example,

$$\Pr\left[y\left(\widehat{X}_t^{(m)}\right) = 0\right] = \Pr\left[k^{(m)} = 0\right] = 0.45$$

and

$$\Pr\left[y\left(\widehat{X}_t^{(m)}\right) = 3\right] = \Pr\left[k^{(m)} = 1\right] = 0.55 .$$

Since $K^{(m)} \leq N$, it can be seen that state quantization will increase the sparsity of $\mathbf{A}$ and $\mathbf{B}$.

TABLE I
AN EXAMPLE OF PMF AND STATE QUANTIZATION

n	0	1	2	3	4	5
count(n)	900	80	100	620	200	100
$p_{Y_m}(n)$	0.45	0.04	0.05	0.31	0.10	0.05
$k^{(m)}$, $\mathbf{y}_{peak}^{(m)}$	0, 0	1, 3				
Pr$[k^{(m)}]$	0.45	0.55				

The super-state corresponding to quantized internal states is $\widehat{S}_t = (\widehat{X}_t^{(1)}, \widehat{X}_t^{(2)}, ..., \widehat{X}_t^{(M)})$. The quantized super-states can be indexed linearly in terms of the indices of the quantized internal states $k^{(1)}, k^{(2)}, ..., k^{(M)}$ as follows:

$$k = k^{(M)} + \sum_{m=1}^{M-1} \left(k^{(m)} \cdot \prod_{i=m+1}^{M} K^{(i)} \right) . \quad (4)$$

Based on (4), one can also extract individual load state indices $k^{(m)}$ from k by iteratively extracting the remainder of division of k by partial products $\prod_{i=1}^{m} K^{(i)}$, starting with $k^{(M)}$.

The sparsity of $\mathbf{A}$ and $\mathbf{B}$ can be used to simplify computations involved in Viterbi-based state decoding, as will be described in the next section. In addition, storage requirements can be significantly reduced. Previously, we employed a modified version of the *Harwell-Boeing sparse matrix format* [17] (or CCS, compressed column storage) to store $\mathbf{A}$ and $\mathbf{B}$ in compressed form. Our modified CCS format uses *hash mapped* data structures (e.g., Python dict datatype) for $\mathbf{P_0}$, $\mathbf{A}$, $\mathbf{B}$, and Viterbi algorithm path vectors $\mathbf{P_t}$ which resulted in the ability to disaggregation greater than 11 loads. We elaborate more on this in the Space Complexity Subsection (IV.A).

E. Sparse Viterbi Algorithm

The standard Viterbi algorithm is well suited for well-populated matrices. However, when used with sparse matrices, there is an extensive amount of naive probability calculations involving zero-probability terms. Matrix compression not only reduces the amount of storage, but it also allows us to avoid calculating zero-probability terms. Taking advantage of matrix sparsity to avoid unnecessary calculations forms the basis of our algorithm called SPARSE-VITERBI$(\cdot)$ (see Algorithm 1), a greedy version of the Viterbi algorithm.

Let y_{t-1} and y_t be the total current measurements at times $t-1$ and t. The goal is to infer the quantized super-state $\widehat{s}_t$ (or, equivalently, its index k_t), from which we will determine the quantized internal states. This will be achieved by decoding the internal states' indices from the super-state index using (4). These posterior probabilities are stored in vector $\mathbf{P}_{t-1}$ as part of *initialization* (Algorithm 1, lines 1–7).

$$\mathbf{P}_{t-1}[j] = \mathbf{P}_0[j] \cdot \mathbf{B}[j, y_{t-1}], j = 1, 2, ..., K . \quad (5)$$

The computation is reduced by only considering non-zero elements of $\mathbf{B}[j, y_{t-1}]$ in (5), to be clarified below. We now calculate the posterior probabilities for the current time period (the *recursion* step, Algorithm 1, lines 9–14)

$$\mathbf{P}_t[j] = \max_{i=1}^{K} (\mathbf{P}_{t-1}[i] \cdot \mathbf{A}[i, j] \cdot \mathbf{B}[j, y_t]), j = 1, 2, ..., K . \quad (6)$$

We *terminate* (Algorithm 1, line 17) to find the most likely current super-state index $k_t = \arg\max(\mathbf{P}_t)$. This algorithm is called each time we need to disaggregate a reading, using a *sliding window* of observations. For example, we disaggregate $t = \{1, 2\}$, then $t = \{2, 3\}$, $t = \{3, 4\}$, and so on. Disaggregation only begins when the first 2 observations are received from the meter. For our purposes we are not interested in the prediction of the super-state from $t-1$ (the *backtracking* step). Once the k_t is determined feedback is sent to the occupant – this makes it final, no turning back time. Again, zero-probability terms are avoided in the calculation. The zero-probability terms are effectively ignored due to matrix compression method, which only stores non-zero elements of the corresponding matrices. Using hash mapped structures (Algorithm 1, lines 5, 10, and 11) only returns non-zero elements, which removes zero probability terms from being calculated.

Algorithm 1 SPARSE-VITERBI$(K, \mathbf{P}_0, \mathbf{A}, \mathbf{B}, y_{t-1}, y_t)$

```
1:  # Set empty posterior path hash mapped structures
2:  P_{t-1} ← {}, P_t ← {}
3:
4:  # Viterbi Step 1: initialization
5:  for (j, p_b) ∈ B[y_{t-1}] do
6:      P_{t-1}[j] ← P_0[j] · p_b
7:  end for
8:
9:  # Viterbi Step 2: recursion
10: for (j, p_b) ∈ B[y_t] do
11:     for (i, p_a) ∈ A[j] do
12:         P_t[j] ← max(P_{t-1}[i] · p_a · p_b)
13:     end for
14: end for
15:
16: # Step 3: terminate
17: return arg max(P_t)
```

F. Load Consumption Estimation

If we know the current/power draws of each appliance $y_t^{(m)}$, we could sum these levels $y_t \equiv \sum_{m=1}^{M} y_t^{(m)}$, and the total should equal the aggregate reading from the smart meter y_t. However, the consumption amount of each load is hidden because the state of each load is hidden. We can estimate the amount of consumption draw for each load by $y(\hat{x}_t^{(m)})$. The current draw of the decoded quantized state $\hat{x}_t^{(m)}$ is assumed to be the location of the corresponding PMF peak, i.e., $y(\hat{x}_t^{(m)}) = \mathbf{y}_{peak}^{(m)}[k_t^{(m)}]$, where $k_t^{(m)}$ is the index of the decoded quantized internal state of appliance m at time t. To find the estimate of the whole-house we simply sum the load estimates

$$\hat{y}_t = \sum_{m=1}^{M} y(\hat{x}_t^{(m)}) = \sum_{m=1}^{M} \mathbf{y}_{peak}^{(m)}[k_t^{(m)}] . \quad (7)$$

IV. Algorithm Complexity & Efficiency

Table II and Figure 2 summarizes the discussion below. We did not test more then 19 loads as AMPds [12] only has 19 sub-meters.

A. Space Complexity

As the number of loads to disaggregate increases, the number of super-states K grows exponentially, and so too do the dimensions of $\mathbf{P_0}$, $\mathbf{A}$, $\mathbf{B}$, and Viterbi algorithm path vectors $\mathbf{P_t}$. However, in the case of load disaggregation these vectors and matrices are very sparse. In fact, so sparse that using an HMM with full super-state space is a practical option. The theoretical sparsity of $\mathbf{B}$ is clear from its definition. Since for each specific super-state $s_t = (x_t^{(1)}, x_t^{(2)}, ..., x_t^{(M)})$ there is exactly one output

$$y_t = y\left(x_t^{(1)}\right) + y\left(x_t^{(2)}\right) + \cdots + y\left(x_t^{(M)}\right) \ , \qquad (8)$$

each row of $\mathbf{B}$ should contain exactly one non-zero element (and that element is equal to 1). However, with quantization, this might not hold exactly, as the current draw may fluctuate slightly within a quantized state even if the super-state does not change. So we may observe several non-zero values in a given row of $\mathbf{B}$, which of course should still sum up to 1. Thus the best-case sparsity of $\mathbf{B}$, defined as the fraction of zero entries, can be calculated as

$$1 - \frac{K}{K \cdot N} = 1 - \frac{1}{N} \ . \qquad (9)$$

The sparsity in $\mathbf{A}$ reflects the fact there are relatively few possible transitions from any given super-state. While this is not as obvious as in the case of $\mathbf{B}$, it can be appreciated by realizing that multi-state appliances usually operate in *cycles* that determine the sequence of their states. For example, a possible state sequence for a dishwasher could be OFF $\rightarrow$ WASH $\rightarrow$ RINSE $\rightarrow$ DRY $\rightarrow$ OFF. Meanwhile, DRY $\rightarrow$ WASH $\rightarrow$ OFF would not make much sense.

Originally we used CCS [17] for matrix compression but found that this was not sufficient. The *column pointer* vector requires space of $\mathcal{O}(K)$ and grows exponentially with the number of super-states. Not compressing the column pointer vector prevented us from disaggregating more than 11 loads. We can dramatically reduce the space requirements by either moving to a hash mapped data structure or a *run length encoding* (RLE)[1] of the column pointer vector. We chose to use a hash mapped (dict in Python) mainly due to our tests being run in Python. Figure 2 (left) demonstrates that when using the AMPds dataset [12] the approximate space cost for $\mathbf{A}$ can be given as $\mathcal{O}(k\sqrt{k}\log_2 K^2)$, where k is the sum of the states of all loads $\sum_m K^{(m)}$. Similar decreases are noticed when using the much smaller REDD dataset [18].

[1] see https://en.wikipedia.org/wiki/Run-length_encoding.

B. Time Complexity

The Viterbi algorithm runs in $\mathcal{O}(K^2T)$. Our algorithm runs online where the length of T is always 1 (or $T = 1$) reducing the time complexity to $\mathcal{O}(K^2)$ which is susceptible to exponentiality of K. Using compressed matrices eliminated any zero-probability states from being calculated which further reduces the computation time cost. Although the worst case time cost would be $\mathcal{O}(K^2)$, in practice this is not the case. Figure 2 (right) demonstrates that when using the AMPds dataset the approximate time cost can be given as $\mathcal{O}(k \log_2 K^2)$, where k is the sum of the states of all loads $\sum_m K^{(m)}$. Similar decreases are noticed when using the much smaller REDD dataset.

C. Algorithm Efficiency

To show the benefits of taking advantage of sparsity we ran our disaggregator using the AMPds dataset. Our tests showed that when disaggregating 2 loads (16 super-states), for every 1 section of code execution in the Recursion Step (Algorithm 1, line 12) of our sparse Viterbi algorithm, the basic Viterbi algorithm would, on average, execute the same section 20 times.

Figure 3 shows that amount of time (in milliseconds) it takes to disaggregate one reading for both a basic Viterbi algorithm and our sparse Viterbi algorithm. These tests ran using Python 3.4 on a 64-bit Debian 8 PC with an Intel Core i7-4790 3.6GHz processor and 32GB of memory. All tests ran in a single thread. As expected the time taken to disaggregate using the basic Viterbi algorithm is exponential following K. However, the sparse Viterbi algorithm avoids this having far better execution times. Results seem to suggest that the more loads that are modeled, the more sparse the data structures in the HMM. We can efficiently process an 8.2 billion state sparse HMM in an average of 4.5 milliseconds.

V. Accuracy Experimentation

To support our contribution claims in Section I-A, we chose two low-frequency datasets: AMPds [12] and REDD [18]. These datasets differ in terms of low-frequency sampling rates (per minute *vs* per 3-seconds) and measurement types (current *vs* apparent power). A prototype of our disaggregator was first coded in Python 3.4. We chose Python because of list manipulation capabilities and its ease in creating rapid prototypes that are easily translated into C (at a later point). Our prototype ran as a single threaded process and would disaggregate the specified loads in all the readings of a given dataset. All tests ran on a Mac Pro (Late 2013 model) with a 3.5GHz 6-core Intel Xeon E5 processor and 64GB of memory. Each test ran used 10-fold cross-validation to mitigate testing accuracy bias.

A. Nomenclature

There are a number of standard terms we use when reporting experimental results in our findings tables. We now explain.

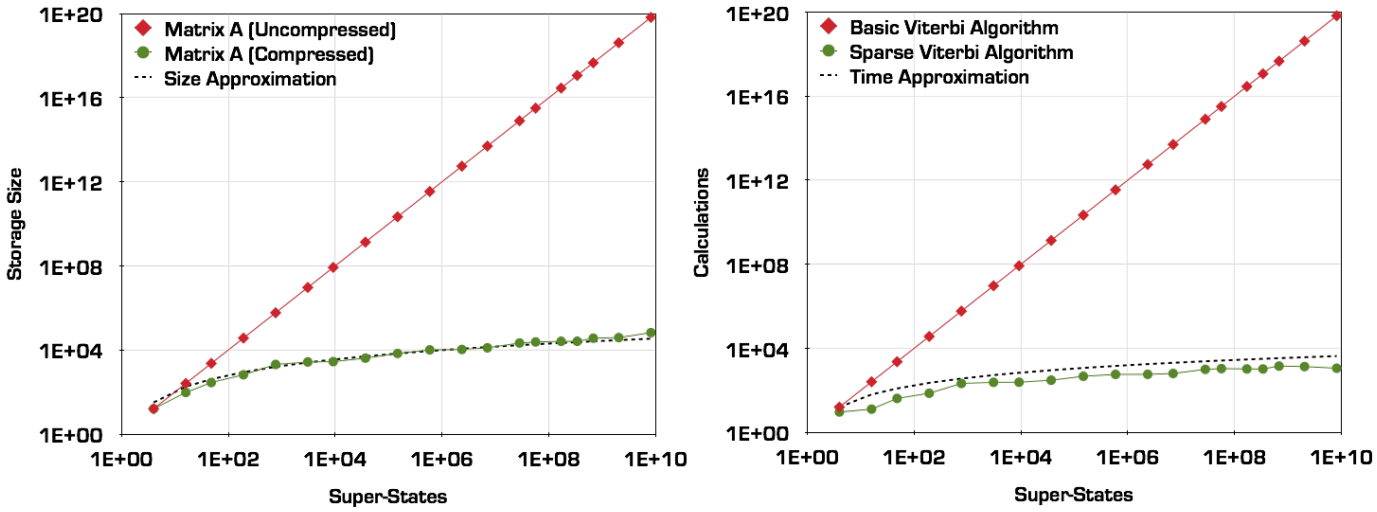

Fig. 2. Space complexities (left) and time (right) complexities for comparing standard HMM space and Viterbi algorithm time with their sparse implementations as the number of super-states increases. Test used AMPds disaggregating from 1 to 19 loads. A graphical depiction of Table II.

TABLE II
TIME AND SPACE COMPLEXITY RESULTS USING DATA FROM AMPDS: UN/COMPRESSED HMMS AND BASIC/SPARSE VITERBI ALGORITHMS

Loads	Super-States, $\mathbf{P}_0(K)$, Init Step	$\mathbf{A}(K\times K)$, Recurs Step	$\mathbf{B}(K\times N)$	Basic Disagg Time (ms)	$\mathbf{P}_0$ (!0)	$\mathbf{A}$ (!0)	$\mathbf{B}$ (!0)	Sparse Init Step	Sparse Recurs Step	Sparse Disagg Time (ms)
1	4	16	8k	0.1	4	16	2.1k	3	9	0.02
2	16	256	32k	1.4	13	95	2.7k	4	13	0.03
3	48	2.3k	96k	13	36	283	4.2k	8	41	0.07
4	192	36.9k	384.2k	198	90	659	6.4k	12	73	0.12
5	768	589.8k	1.5M	3,1501	258	2.1k	11.9k	24	214	0.28
6	3,072	9.4M	6.1M	50,409	386	2.7k	13.7k	28	242	0.34
7	9,216	84.9M	18.4M	451,869	411	2.8k	13.9k	29	244	0.37
8	36,864	1.4G	73.8M		674	4.1k	17.2k	39	304	0.51
9	147,456	21.7G	295.1M		1.2k	6.9k	23.6k	55	467	0.74
10	589,824	347.9G	1.2G		2k	10.2k	31.1k	80	575	1.05
11	2,359,296	5.6T	4.7G		2.1k	10.6k	31.9k	83	578	1.10
12	7,077,888	50.1T	14.2G		2.7k	12.8k	36.1k	97	637	1.28
13	28,311,552	801.5T	56.7G		4.7k	21.4k	50k	151	1,001	2.10
14	56,623,104	3.2P	113.3G		5.5k	24.1k	54.1k	175	1,089	2.38
15	169,869,312	28.9P	339.9G		6.2k	25.8k	57.1k	187	1,046	2.51
16	339,738,624	115.4P	679.8G		6.2k	25.9k	57.1k	187	1,047	2.65
17	679,477,248	461.7P	1.4T		8.9k	36.4k	67.1k	233	1,438	3.33
18	2,038,431,744	4.2E	4.1T		9.8k	38.8k	70.1k	253	1,357	3.78
19	8,153,726,976	66.5E	16.3T		18k	67.1k	79.7k	306	1,133	4.55

1) Test Type: Each test ran on the same dataset and with the same loads, but was configured slightly differently. These different test configurations were: Denoised, Noisy, Modelled. A *Denoised* configuration removes any noise (as defined in [19]) from the aggregate reading so that the aggregate observed value y is equal to the sum of the loads values $y^{(m)}$. A *Noisy* configuration does not remove the noise in the aggregate observed value y, nor does it try to model the noise as a load. To us, this represents a more realistic configuration to test against. A *Modeled* configuration treats the noise in ground truth as a load we label *unmetered* which is then modeled as an additional load to disaggregate.

2) Noise: Here we report the percent-noisy measure (%-NM) [19] defined as

$$\%\text{-NM} = \frac{\sum_{t=1}^{T} |y_t - \sum_{m=1}^{M} y_t^{(m)}|}{\sum_{t=1}^{T} y_t} \times 100 \ , \qquad (10)$$

where y_t is the aggregate observed current/power amount at time t and $y_t^{(m)}$ is the ground truth current/power amount for each appliance m to be disaggregated. For example, a denoised test would result in 0%; whereas, a %-NM of 40% would mean that 40% of the aggregate observed current/power for the whole test was noise.

3) Acc: Here we report the basic accuracy measure as defined $Acc = \frac{correct}{correct+incorrect}$. Although this measure is not the best way to report accuracy, we include it because a majority of disaggregation researchers use this measure.

4) FS-fscore: Here we report our Finite State version of f-score [19] rather than the usual binary measures f-score.

5) Est Acc: Here we report the consumption estimation accuracy measure developed by Kolter and Johnson [18].

B. Testing Deferrable Loads

The use of large appliances, such as a clothes dryer can be deferred until the cost of electricity is cheaper, called off-peak hours. Large load deferral should be the focus of NILM because there is a direct benefit to the occupants (saving money) and may have a direct benefit (brownout avoidance) to the power grid as a whole. Deferring large loads from running during peak times of usage can ensure power grid stability and avoid grid brownouts. This idea is often discussed as *peak shaving* [20]. There are other benefits to having NILM only disaggregate deferrable loads. Firstly, large consumption can be more easily identified resulting in better accuracy. Secondly, disaggregators can run faster as there are fewer loads to disaggregate. However, the disaggregator must be robust enough to handle a large percentage of *noise* – many other loads that would be running in the house.

To test the accuracies for disaggregating deferrable loads we used AMPds [12] choosing the clothes dryer, the dishwasher, the HVAC system, the heat pump, and the kitchen wall oven. We ran all three test types on one year worth of data (524,544 readings). These loads are multi-state loads. The HVAC system has a *continuous variable* fan motor and a *constantly on* thermostat. The readings within APMds do not contain errors, so there was no need to clean or convert the data. We ran tests using per minute sampling (minute tests) and per hour sampling (hour tests). Denoised tests do not represent a realistic scenario; we only report them for comparison as to what the ideal result would be.

All three test types scored very high with the lowest score being 94.1% for the estimation accuracy of the Noisy test (see Table III). Looking at the specific load results for the noisy and model tests, we noticed the accuracy results for the dishwasher scored very low (see Figure 4), while other loads scored quite high. The low results were due to having two load states (at peaks 0.4A and 1.2A) similar to other loads (2 loads and 3 loads respectively). In the model tests, the unmetered load scored low because we restricted a number of states to a maximum of 4. During this experiment, 7,522 of 524,544 observations have multiple loads switch states simultaneously.

We also wanted to test how well our disaggregator would disaggregate very low-frequency sampling data – hourly data. In these tests, we used Watt-hour measurements (at daWh or Wh÷10 precision). We accumulated the Watt-hours for each hour over the entire year for a total of 17,520 readings. Table IV shows the overall results of the hour tests. The Noisy test performed with high classification accuracy (91%) having the estimation accuracy score roughly 4% lower. Examining the load specific scores, we noticed that the dishwasher, again, scored low. However, we also found the oven scored even lower often having an FS-fscore of 0%. There were misclassifications that occurred because of the high level of noise and the infrequent, nonuniform use of the oven. We were surprised our disaggregator scored as well as it did after reading Kolter's [21] published results for hourly disaggregation. His discriminative disaggregator achieved an estimation accuracy of 55%, which is 31% lower than our Noisy test. Kolter used a different dataset (Plugwise) which contained a large number of houses, so it is hard to do a direct comparison of results.

TABLE III
DEFERRABLE LOADS ACCURACY RESULTS (/MINUTE, A)

Test Type	Noise	Acc	FS-fscore	Est Acc
Denoised	0.0%	99.2%	99.2%	99.0%
Noisy	60.6%	98.0%	98.1%	94.1%
Modelled	53.4%	93.6%	94.5%	98.6%

TABLE IV
DEFERRABLE LOADS ACCURACY RESULTS (/HOUR, WH)

Test Type	Noise	Acc	FS-fscore	Est Acc
Denoised	0.0%	96.7%	95.2%	93.7%
Noisy	68.2%	94.6%	90.6%	86.2%
Modelled	51.4%	91.7%	87.0%	88.0%

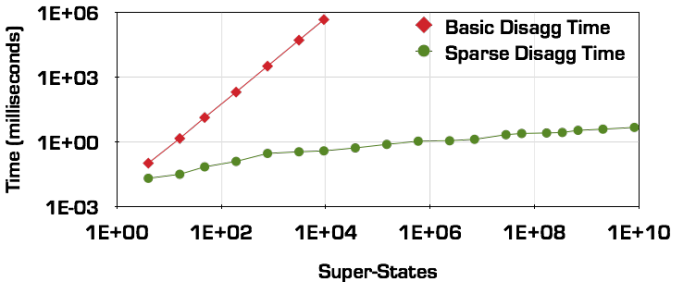

Fig. 3. Runtimes (in milliseconds) of a single disaggregation for both the basic and sparse Viterbi algorithms as the number of super-states increases. Runtime numbers listed in Table II

C. Comparing Disaggregators

To compare the accuracy of our disaggregator to that of others [16], [18] we used the REDD dataset [18] (a smaller dataset compared to AMPds, but data sampled every 3 seconds) and used the same estimation accuracy measure. We wanted to compare our results against the Kolter 2011 [18] results and the Johnson 2013 [16] results. There has been a long-standing issue in the field of NILM where the use of different datasets and accuracy measure makes comparing different algorithms near impossible. However, Kolter 2011 and Johnson 2013 use the same dataset and accuracy measures and publish results that are easy to compare to. The majority of NILM papers do not do this, and we are hoping to continue a trend started by

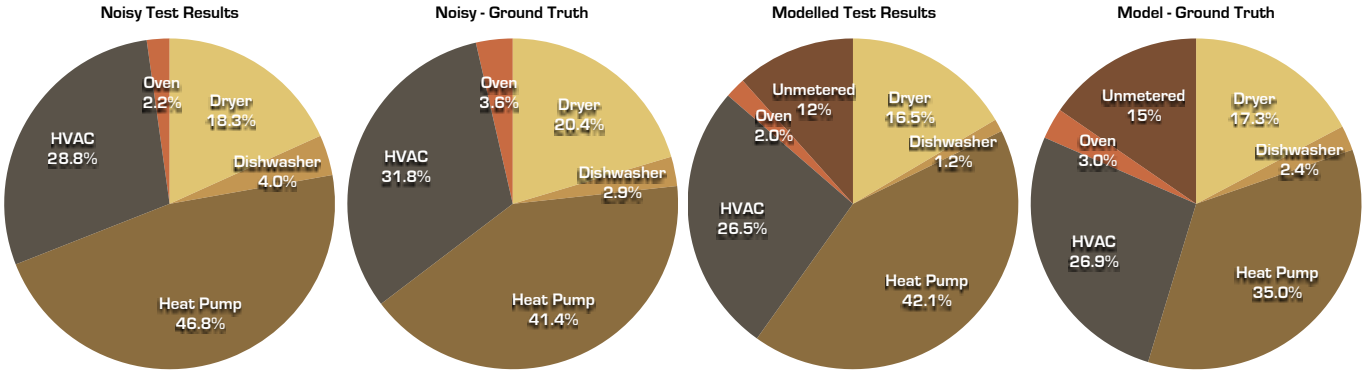

Fig. 4. The resulting estimation accuracy results for Noisy (left) and Modelled (right) tests as compared to their respective ground truth.

the Johnson 2013 paper. Both papers use more complex types of Markov model, and we wanted to show how much more accurate a simple HMM can be. We also wanted to show that our observations about sparsity hold for other datasets.

Kolter and Johnson [18] initially reported accuracies on Houses 1, 2, 3, 4, and 6 using a disaggregator that used supervised learning. Johnson et al. [16] reported accuracies on Houses 1, 2, 3, and 6 using a disaggregator that they claimed used unsupervised learning. However, their claim of unsupervised learning was incorrect. They used a supervised learning solution because labeled data was used to build appliance models (also the opinion of Parson [15]). We chose to use the low-frequency sampling data from REDD that was in apparent power measurements. The whole-house aggregate power meter sampled at 1Hz (per second), opposed to the load sub-meters that sampled at $\frac{1}{3}$Hz (3-second intervals). As a result, we needed to clean and convert the data for our tests. For our tests, we downsampled the aggregate data from 1Hz to $\frac{1}{3}$Hz by discarding the between readings after matching the timestamps. There was the additional problem where the timestamps in some of the readings were out of sync because two different metering systems were used (one for aggregate and one for loads). To correct mismatches we had our conversion program scan the aggregate data records forward and backward until the aggregate reading matched the sum of all the load readings. There were a few instances where we added the difference to the aggregate reading so that the noise cancelation would read zero.

TABLE V
COMPARING ACCURACY RESULTS WITH OTHER PUBLISHED WORK

REDD	Kolter 2011 [18]	Johnson 2013 [16]	Noisy Test	Modelled Test
House 1	46.6%	82.1%	95.6%	99.3%
House 2	50.8%	84.8%	94.8%	99.0%
House 3	33.3%	81.5%	90.6%	97.5%
House 6	55.7%	77.7%	98.4%	99.7%
Average	46.6%	81.5%	94.9%	98.9%
Gain wrt. [18]	—	+34.9%	+48.3%	+52.3%
Gain wrt. [16]	-34.9%	—	+13.3%	+17.3%

We used the same previous three tests, and we selected the same five loads to disaggregate as Johnson did: refrigerator, lighting, dishwasher, microwave, and furnace. We compared our results with the other two in Table V. Note the other two papers did not report load specific accuracies, so we cannot compare those. While the Johnson 2013 results were significantly better than the Kolter 2011 results, our two testing results were significantly better than the Johnson 2013 results. Our Noisy test results were better than Johnson 2013 by 13.3%, and better than Kolter 2011 by 48.3%.

VI. ANALYSIS OF ACCURACY RESULTS

We have shown how our disaggregator achieves a high degree of both load classification and consumption estimation using different low-frequency sampling rates and different measurement. Zeifman [14] previously identified six key requirements he believed needed to be met in order for load disaggregation to be solved. The six requirements are listed below and we review how well we have met these requirements. We have confidently met four of Zeifman's [14] six key requirements. The two we have not confidently met are *no training* and *scalability*.

A. Feature Selection

Feature selection constrains the measurements to those of a typical smart meter: current, apparent power and energy with low-frequency sampling. Our disaggregator has met this requirement. We can disaggregate accurately at even lower sampling frequencies: per minute and per hour.

B. Accuracy

Accuracy requires disaggregators to have a minimal accuracy measure score of 80%. Figure 5 shows our disaggregator has met this requirement.

C. No Training

No training requires disaggregators to *not involve significant occupant efforts* [14] to train. Our disaggregator minimizes the involvement of occupants to just initial model building without pre-tuning.

The idea of no training may be better rephrased as *minimal setup* which could be handled in two ways: the collection of priors or the active tuning of general load models. We

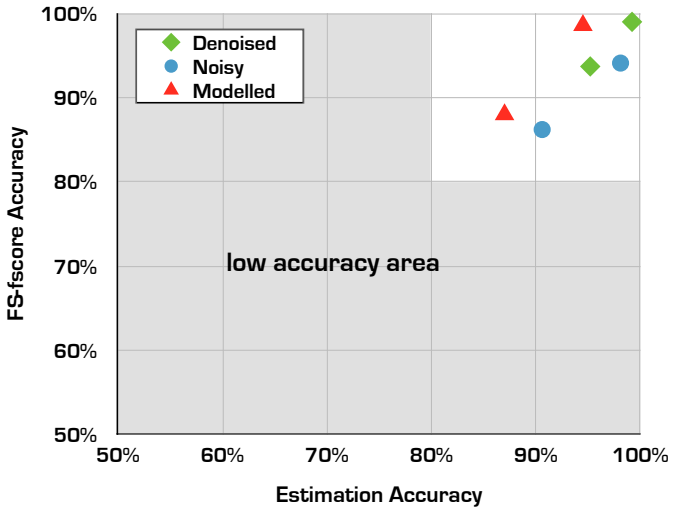

Fig. 5. Compare the deferrable loads test scores (Table III and Table IV) for FS-fscore and estimation accuracies using Zeifman's minimum 80% accuracy requirement.

have chosen the collection of priors which provides more accurate results from the start. Active tuning can only provide accurate results once the general models are tuned properly to the specific loads in the house that could take an arbitrary amount of time. How will the occupants know when this active tuning period has completed and that the disaggregation results can now be trusted as accurate? Low accuracy can cause occupants to lose confidence in the disaggregator's ability to work properly.

D. Near Real-Time

Near real-time capabilities mean disaggregators must run online and respond to events as they happen; i.e., the algorithm must be robust and efficient. Our disaggregator has met this requirement and can disaggregate in under one millisecond.

E. Scalability

Scalability requires disaggregators not to require additional processing time and/or hardware to account for the identification of new appliances. Our disaggregator currently cannot do this; however, it can disaggregate accurately with high levels of noise. One observation is the house used to create AMPds did not have any major deferrable load changes in 10 years.

The idea of scalability is very much still an open research question. It is difficult to identify new loads without devoting some computational time to the act of finding them. This also assumes we want to disaggregate every load a house has. Disaggregation is very much a problem that is computationally exponential in both time and space. This means we can mitigate some effects of exponentiality with different strategies, such as our sparsity optimization or the use of factorial models. We need to disaggregate loads that will allow for the end goal of energy conservation and design robust algorithms that handle large amounts of noise (unmetered loads) like our disaggregator can. If, for instance, an occupant wants to disaggregate lights, then it may be better to use home automation systems to do this. Home automation systems can be used to inform the occupant whether a light is ON or OFF along with the ability to remotely control it.

F. Various Appliance Types

Various appliance types must be handled/detected by the disaggregators. To simplify, our disaggregator only uses one appliance type, multi-state (or finite state). Simple ON/OFF and constantly on are simply special cases of multi-state. For larger appliances such as a front-load washer, with variable speed drum (continuously variable), we create a state for this operation that seems to be sufficient for estimation purposes.

VII. Conclusions

Our disaggregator was designed to run in real-time on an embedded processor using low-frequency sampling data in an effort to show load disaggregation is indeed a viable method for enabling occupants to understand how their home consumes energy. This understanding would allow occupants to make intelligent, informed decisions on how they conserve the use of energy, which by all accounts is a very personal and dynamic decision-making process. It is personal and dynamic because the goals of the occupants involve many factors such as home characteristics, occupant comfort levels, and budgetary constraints (to name a few).

Our disaggregator can disaggregate a model with a large number of super-states. However, there is still the exponential limitation in time and space. We have only pushed back the exponential problem through optimization, but have we pushed it back far enough? No, if we want to try to disaggregate every load in a home. Yes, if we want to disaggregate deferrable loads that affect the amount owing on a power utility bill. We believe the main goal for load disaggregation is to help occupants conserve energy through the smarter use of high energy consumption loads. For example, occupants running large consuming deferrable loads when the per kWh is low, instead of when it is high, would see savings on their power utility bill. The ability to disaggregate 18 loads, as we have shown, goes far beyond the number of loads previously reported in other publications (usually ranging from 6–9 loads).

Despite the proactive discounting of a model/algorithm because of its theoretical limitations, promising models/algorithms often get overlooked or simply argued away. While the theoretical limitation of exponentiality remains, through the deep analysis of data, modifications can be made to these models and algorithms that allow for use in practice. We have demonstrated this through the design of our super-state HMM and sparse Viterbi algorithm. We plan to extend our disaggregator by providing an active tuning module that would allow for a more general super-state HMM to be tuned specifically to a given house.

Source Code

With the publication of the article, we have released our disaggregator as open source for academic use (free of charge). Disaggregator source code is available for download from GitHub at *https://github.com/smakonin/SparseNILM*. In the near future our algorithm will be added into NILMTK [22].

REFERENCES

[1] G. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[2] S. Makonin, L. Guzman Flores, R. Gill, R. A. Clapp, L. Bartram, and B. Gill, "A Consumer Bill of Rights for Energy Conservation," in *Proceedings of the 2014 IEEE Canada International Humanitarian Technology Conference (IHTC)*, 2014.

[3] M. Zeifman and K. Roth, "Viterbi algorithm with sparse transitions (VAST) for nonintrusive load monitoring," in *Proceedings of the IEEE Symposium on Computational Intelligence Applications In Smart Grid (CIASG)*, 2011, pp. 1–8.

[4] S. Makonin, I. V. Bajic, and F. Popowich, "Efficient Sparse Matrix Processing for Nonintrusive Load Monitoring (NILM)," in *Proceedings of the 2nd International Workshop on Non-Intrusive Load Monitoring*, 2014.

[5] S. Makonin, "Real-Time Embedded Low-Frequency Load Disaggregation," Ph.D. dissertation, Simon Fraser University, School of Computing Science, Aug 2014.

[6] M.-S. Tsai and Y.-H. Lin, "Modern development of an adaptive non-intrusive appliance load monitoring system in electricity energy conservation," *Applied Energy*, vol. 96, no. 0, pp. 55–73, 2012.

[7] M. Figueiredo, A. de Almeida, and B. Ribeiro, "Home electrical signal disaggregation for non-intrusive load monitoring (NILM) systems," *Neurocomputing*, vol. 96, no. 0, pp. 66–73, 2012.

[8] M. E. Berges, E. Goldman, H. S. Matthews, and L. Soibelman, "Enhancing electricity audits in residential buildings with nonintrusive load monitoring," *Journal of Industrial Ecology*, vol. 14, no. 5, pp. 844–858, 2010.

[9] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer New York, 2006, vol. 1.

[10] J. Kolter and T. Jaakkola, "Approximate inference in additive factorial hmms with application to energy disaggregation," *Journal of Machine Learning Research - Proceedings Track*, vol. 22, pp. 1472–1482, 2012.

[11] H. Kim, M. Marwah, M. F. Arlitt, G. Lyon, and J. Han, "Unsupervised disaggregation of low frequency power measurements." in *Proceedings of the SIAM Conference on Data Mining*, 2011, pp. 747–758.

[12] S. Makonin, F. Popowich, L. Bartram, B. Gill, and I. V. Bajic, "AMPds: A Public Dataset for Load Disaggregation and Eco-Feedback Research," in *Proceedings of the IEEE Electrical Power and Energy Conference (EPEC)*, 2013.

[13] T. Zia, D. Bruckner, and A. Zaidi, "A hidden Markov model based procedure for identifying household electric loads," in *Proceedings of the 37th Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2011, pp. 3218–3223.

[14] M. Zeifman, "Disaggregation of home energy display data using probabilistic approach," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 1, pp. 23–31, 2012.

[15] O. Parson, "Unsupervised Training Methods for Non-intrusive Appliance Load Monitoring from Smart Meter Data," Ph.D. dissertation, University of Southampton, Electronics and Computer Science, 2014.

[16] M. J. Johnson and A. S. Willsky, "Bayesian nonparametric hidden semi-Markov models," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 673–701, 2013.

[17] I. S. Duff, R. G. Grimes, and J. G. Lewis, "Sparse matrix test problems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 15, no. 1, pp. 1–14, 1989.

[18] J. Kolter and M. Johnson, "REDD: A public data set for energy disaggregation research," in *Proceedings of the Workshop on Data Mining Applications in Sustainability (SIGKDD)*, 2011.

[19] S. Makonin and F. Popowich, "Nonintrusive load monitoring (NILM) performance evaluation," *Energy Efficiency*, vol. 8, no. 4, pp. 809–814, 2014.

[20] B. Roberts, "Shaving load peaks from the substation," *POWER Magazine. [Online]. Available: www.powermag.com*, 2006.

[21] J. Kolter, S. Batra, and A. Ng, "Energy disaggregation via discriminative sparse coding," in *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, 2010.

[22] J. Kelly, N. Batra, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, "Nilmtk v0.2: A non-intrusive load monitoring toolkit for large scale data sets: Demo abstract," in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings (BuildSys)*, 2014.

Stephen Makonin is currently a Postdoctoral Fellow in Engineering Science at Simon Fraser University (SFU) and previously a Postdoctoral Fellow in Computing Science. He received his PhD in Computing Science from SFU in 2014. His research interests are in big data, machine learning, computational sustainability, disaggregation, NILM, and visualization. Previously, he worked in industry as a software developer for over 12 years. He is an active member of the IEEE, ACM, and AAAI and volunteers as the IEEE Vancouver Joint Computing Chapter Chair (includes both the Computer and Computational Intelligence Societies).

Fred Popowich received his PhD in Cognitive Science from the University of Edinburgh in 1989, and since then has been a faculty member in the School of Computing Science at Simon Fraser University. He is currently the President of the Canadian Network for Visual Analytics and Founding Director of the Vancouver Institute for Visual Analytics. Much of his research is concerned with how computers can be used to process human language, either to make it easier for human beings to interact with computers, or to make it easier for human beings to interact with each other. As such, he has been concerned with how knowledge about language and the world can be represented, maintained, and even learned by computers. Typical real world applications of this research include *smart homes*, the automatic translation of language, tools to assist people in learning language, and technology to help people search and manage the information contained on computer systems and networks.

Ivan V. Bajić is Associate Professor of Engineering Science at Simon Fraser University, Burnaby, BC, Canada. His research interests include signal, image, and video processing and compression, multimedia ergonomics, and communications. He has authored about a dozen and co-authored another eight dozen publications in these fields. He has served on the organizing and/or program committees of various conferences in the field, including GLOBECOM, ICC, ICME, and ICIP, was the Chair of the Media Streaming Interest Group of the IEEE Multimedia Communications Technical Committee from 2010 to 2012, and is currently an elected member of the IEEE Multimedia Systems and Applications Technical Committee, Associate Editor of the IEEE Signal Processing Magazine, and Chair of the Vancouver Chapter of the IEEE Signal Processing Society.

Bob Gill is a registered professional engineer in the province of BC. He started his career with IBM in Toronto designing computer systems, moved on to British Columbia Hydro and power authority and as part of a multi-skilled team worked in telecommunication and powers systems technologies for approximately 7 years. He completed his grad studies at the Simon Fraser University in 1996. He then moved on the TELUS and worked as a Technology Manager. Since 1999, Bob has been a faculty member at British Columbia Institute of Technology and is currently the Program Head for the Telecommunication and Networks Option. He is currently the IEEE Vancouver Section Chair and is a Fellow of Engineers Canada.

Lyn Bartram is Associate Professor in the School of Interactive Art and Technology at Simon Fraser University (SFU), where she is Director of the Human-Centred Systems for Sustainable Living research group. She holds a PhD in Computer Science from SFU. Her work explores the intersecting potential of information technologies, ubiquitous computing, computational media and sustainable building design in encouraging conservation and reducing our ecological footprint in our homes and personal activities. She is a member of the ACM.