iProgram: Inferring Smart Schedules for Dumb Thermostats

Sandeep Kalra, Srini Iyengar, David Irwin, Prashant Shenoy, and Benjamin Marlin
University of Massachusetts Amherst

ABSTRACT

Heating and cooling accounts for over 50% of a typical home's energy usage. While programmable thermostats that control residential HVAC are cumbersome to program, next-generation smart thermostats have not seen wide adoption due to their high costs and complex installation. Thus, we focus on making existing “dumb” programmable thermostats smart by leveraging energy analytics on smart meter data to infer home occupancy patterns and compute an optimized thermostat schedule. Unlike smart thermostats, our system, called iProgram, is immediately applicable to a broad array of homes at nearly no cost. iProgram uses an approach based on a Hidden Markov Model (HMM) to derive a custom thermostat schedule for each home using only its smart meter data. We implement iProgram as an open web service and show that, in a representative home, it reduces the miss time by 127 minutes from a default 8am-6pm schedule, and is only 64 minutes off a perfect schedule that precisely tracks occupancy.

1. INTRODUCTION

Buildings account for over 40% of energy and 75% of electricity usage in most developed countries, including United States [13]. Heating, cooling and ventilation (HVAC) accounts for nearly half of the energy usage within a typical building—with HVAC loads consuming 54% of the energy usage within residential buildings. HVAC loads, such as air conditioners and heaters, are typically controlled by a thermostat, such that the user specifies a desired temperature setpoint and the thermostat cycles the HVAC system on and off to maintain the temperature at the desired set point.

The simplest type of thermostat is a manual one, where a user must manually switch a heater or air conditioner on and off and manually specify the desired temperature setpoint when the load is on. Manual thermostats, while simple and inexpensive, suffer from several disadvantages. Their manual nature makes them prone to human errors, since a user may forget to turn off the HVAC system when leaving the house (or leaving a room for an extended period) causing energy wastage. Such thermostats also impact user comfort, since their manual nature implies that the HVAC system cannot be turned on until after the user has arrived home, causing the house to be uncomfortable—too hot or cold—until the HVAC system has fully cooled or heated the house.

To address these drawbacks, most modern thermostats are programmable. A programmable thermostat allows a user to program a *daily schedule*, which specifies the times of the day the HVAC system should be turned on and the corresponding setpoint temperature when on. A programmable thermostat automates when the heating or cooling system comes on based on when the user expects to be home or away. A programmable thermostat can reduce human errors—e.g., by automatically turning off heating or cooling when the user departs for work each weekday as specified in the programmed schedule. Such thermostats can also pre-cool or pre-heat the home on a fixed schedule, which allows the home to be comfortable when the user returns home.

Programmable thermostats have been marketed as energy-saving convenience devices and tens of millions of such thermostats have been sold and installed in homes over many years. Despite their advantages, programmable thermostats suffer from two key drawbacks—complex interfaces and their static nature. The main disadvantage of programmable thermostats is that users find them complex and tedious to program. The interface these thermostats expose is not particularly user-friendly, and programming a schedule requires a user to carefully derive the repeating patterns in their daily schedules to determine an "optimal" schedule to program in. Users often find the task of determining the best schedule for each day of the week and each weekend day to be a cumbersome task.

The second main disadvantage of these thermostats is that the programmed schedule is static in nature and does not adjust to small or large changes in occupants' daily activities. For instance, if a user stays home on a weekday due to illness, the thermostat's schedule must be manually overridden. In addition, long-term changes in daily activities such as a summer schedule being different from a winter schedule requires periodic re-programming. Due to the need to manually program and adjust the schedule, many users simply avoid using these programming features altogether and instead use the thermostat in manual mode, which defeats their purpose.

Recently, a new generation of smart thermostats have been developed to address the key drawbacks of programmable thermostats. Since the main drawback of programmable thermostats is the need to manually determine and program a schedule for HVAC loads, smart thermostats automate this task. In particular, they use onboard or external occupancy sensors to determine when users are home or away and learn repeating patterns of occupancy of the sensor data to derive a schedule. Occupancy sensor data can also be used to make short-term or longer term dynamic adjustments to the schedule. There has been a wide range of prior work on designing such designing "smart" thermostats that automatically adjust a home's temperature setpoint by monitoring and learning home occupancy patterns using various sensors, e.g., motion or GPS [1, 7, 8, 15, 20], and numerous commercial smart thermostat products are now available, including NEST [17], Lyric [16], Ecobee [4], etc.

However, despite their benefits, smart thermostats pose problems that continue to position them as niche devices for energy-efficiency and technology enthusiasts, and prevent them from being deployed in a large number of homes. For example, these thermostat rely on additional occupancy sensors to learn schedules of when users are home or away. While some thermostats have on-board occupancy sensors, others rely of external sensors deployed in a home. The need to install additional sensors significantly drives up the cost of these devices and also increases installation cost for deploying additional sensors. A typical smart thermostat costs *ten* times that of an entry-level programmable thermostat. For example, an entry-level Honeywell programmable thermostat costs \$22, while the Nest and the Lyric cost ∼\$250. The high costs lengthen the return on investment due to the energy-efficiency savings. In addition, smart thermostats do not address issues with the millions of "dumb" programmable thermostats already installed in homes that are being used in a sub-optimal fashion.

Thus, in this paper, we focus on the problem of making existing "dumb" programmable thermostats smart via the use of learning-based techniques and without requiring any additional sensors or new investment. Our goal is to allow homeowners to make more effective use of their existing programable thermostats and extract their full energy-efficiency potential while reducing the pain of determining and adjusting schedules. Our key insight is that electricity usage data from smart meters, which are being deployed by utility companies in larger numbers as part of their smart grid deployment, already reveal occupancy patterns (without requiring additional sensors) and these occupancy patterns can be used to automatically derive a custom thermostat schedule for each home. Thus, it is feasible to develop inexpensive and scalable techniques that use smart meter data to optimize the wide range of "dumb" thermostats already installed in most homes. Our system, called iProgram, analyzes data from a home's smart electricity meter over a long period to infer occupancy patterns and derive an optimal thermostat schedule. The schedule is home-specific, taking into account both its number of zones and type of thermostat, e.g., 1-day, 7-day, or 5-2-day programmable. iProgram's goal is to address current problems that limit smart thermostat adoption by being immediately applicable to the broadest array of homes at no cost.

Our work builds on recent research in energy data analytics, which analyzes smart meter data to learn new insights into home behavior. We focus on smart meter data, since smart meters are one of the most widely deployed sensors deployed in homes: in 2011, an estimated 493 utilities in the U.S. had already installed more than 37 million smart meters with additional deployments continuing to come online [6]. Thus, smart meters provide a readily-available sensor that mitigates the need for a homeowner to install and maintain their own sensors inside the home. In addition, the utilities that collect smart meter data have both an interest in energy-efficiency and direct access to a large set of customers. As a result, utilities are in a position to deliver iProgram's results to users, provide incentives for users to adopt them, and verify their adoption. For example, utilities could include a custom thermostat schedule computed by iProgram for each consumer as part of their energy bill. Utilities already include similar, albeit more rudimentary, data-driven energy-efficiency information in customer bills, e.g., comparing a home's energy usage to their neighbor's usage.

Our hypothesis is that iProgram can improve HVAC efficiency *en masse* by analyzing smart meter data to infer long-term patterns of occupancy, and then compute an optimal thermostat-specific schedule based on those patterns. In evaluating our hypothesis, this paper makes the following contributions.

- **Large-scale Data Analysis**. We analyze smart meter data recorded at five minute intervals over a 6-month period for a small town consisting of 16,800 homes. We show that most homes have highly regular and predictable power usage patterns both across weekends and weekdays and within each day. In particular, we show that most home usage patterns fall into one of three coarse-grained periods: occupied and asleep, occupied and awake, and unoccupied.
- **Deriving Thermostat Schedules**. Based on the observations above, we develop a technique based on Hidden Markov Models (HMMs) to identify and label the three coarse-grained periods above in each day's smart meter data. We then derive an optimal thermostat-specific schedule based on a home's long-term occupancy pattern. Since we identify periods of sleep, our approach is capable of supporting multi-zone systems with one primary zone and bedroom zones, e.g., by setting bedroom zones to a shallower setback than the primary at night.
- **Implementation and Evaluation**. We implement iProgram as an open web service where users may upload their smart meter data and receive suggested thermostat-specific schedules. We evaluate iProgram on data from the Pecan St. dataset. We infer ground truth occupancy in the Pecan St. data using interactive circuit data, and quantify the "miss time" and "waste time," i.e., the amount of time the home is unoccupied and the HVAC system is on, from our inferred thermostat schedules. We show that iProgram is capable of reducing the miss time by 127 minutes from a default thermostat schedule of 8am-6pm, and is only 64 minutes off a perfect schedule.

2. BACKGROUND AND DATA ANALYSIS

iProgram assumes a home is equipped with a networked power meter—a *smart meter*—that reports aggregate electricity usage at fine-grained intervals, e.g., every five minutes. While smart gas meters also exist (our town dataset includes hourly gas data for each home), we focus solely on smart electric meters. Most residential HVAC is electric: all space cooling, i.e., air conditioning, is electric and 38.1% of U.S. homes use some form of electric space heating [5]. While our techniques should also be applicable to smart gas meters, extending iProgram to support them is future work.

We also assume each home has at least one thermostat that regulates HVAC operation. Importantly, though, iProgram does not dictate a specific type of thermostat or configuration. There are a wide range of thermostats available, ranging from simple manual thermostats to programmable thermostats to WiFi-enabled thermostats. Likewise, homes may employ more than one thermostat to support multiple heating and cooling zones. As we discuss, to support the widest range of homes, iProgram is capable of generating multiple possible thermostat schedules optimized for different thermostat types and configurations.

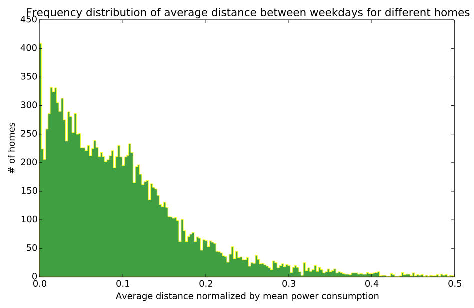


Figure 1: Histogram of distance in power data time-series between weekdays in our town dataset. The graph shows most homes are highly regular with a heavy tail of irregular homes.

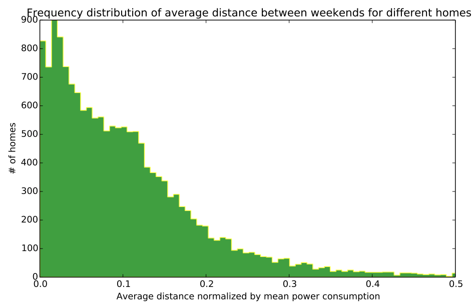


Figure 2: Histogram of distance in power data time-series between weekends in our town dataset. The graph shows most homes are highly regular with a heavy tail of irregular homes.

2.1 Problem Statement

Our goal is to analyze smart meter data to generate a thermostat schedule that best matches a home's pattern of occupancy. Formally, we represent a schedule as a function $S(t)$ that returns a thermostat temperature setting for time t. In essence, the thermostat schedule defined by $S(t)$ consists of a series of variable-length intervals that specify different *setpoint* and *setback* temperatures. The setpoint temperature denotes the comfortable temperature set by the occupants, while the setback temperature denotes the desired thermostat temperature when occupants are not present. A thermostat schedule must specify either a setpoint or setback temperature for each interval. For programmable thermostats, the thermostat schedule must repeat at regular intervals, which limits the range of t. For example, for a 1-day programmable thermostat, t is in the range $0 \leq t \leq 24$ hours, while, for a 7-day programmable thermostat, t is in the range $0 \leq t \leq 168$ hours.

iProgram derives the schedule $S(t)$ by analyzing the time-series $P(t)$ of power readings generated by a smart meter to infer when a home is occupied. As in prior work, we represent occupancy as a binary function $O(t)$, where zero is an unoccupied home and one is an occupied home. Occupancy detection then requires inferring $O(t)$ from $P(t)$. Intuitively, power usage that is high and variable correlates with occupants' use of interactive devices while home. While a variety of background devices that operate autonomously may also cause high power usage, their operation generally follows a regular pattern of usage that is distinct from interactive devices. Prior work evaluates many approaches for detecting occupancy based on this intuition, ranging from employing simple thresholds on power's mean and variance [3] to advanced techniques using Hidden Markov Models (HMMs), Support Vector Machines (SVMs), and k-Nearest Neighbor classifiers [14]. While accuracy depends on the correlation between a home's occupancy and electricity usage, it generally ranges between 75% and 95%.

While prior work quantifies the accuracy of occupancy detection in isolation, iProgram applies the technique to the problem of thermostat scheduling. As we discuss, we adapt and extend an existing technique based on HMMs to identify long occupied and unoccupied periods (suitable for slow adjustments in temperature) and to account for occupancy in multiple zones (a primary zone and one or more bedroom zones). Since our focus is on thermostat scheduling, our performance metric is not the accuracy of occupancy detection, but rather the *MissTime* in the thermostat schedule.

We define two variants of *MissTime*, which have been used in prior work. Our first variant, which we denote as $MissTime_{smart}$, is defined as the total time a home is occupied where its temperature deviates by more than X deg from its occupied setpoint [15]. This metric is typically used to quantify the performance of smart thermostats [15, 20], which are capable of tracking real-time occupancy and adjusting the thermostat temperature in real time. $MissTime_{smart}$ is useful for comparing against the ideal thermostat schedule that perfectly tracks occupancy. Our second variant of MissTime, which we denote as $MissTime_{best}$, represents the best thermostat schedule possible using a conventional programmable thermostat. Note that $MissTime_{best}$ is thermostat-specific, since the best schedule possible depends on the thermostat type, whether it be 1-day, 7-day, or 5-2-day. $MissTime_{best}$ is useful for comparing with the best schedule possible for a given thermostat.

As we discuss below, in most cases, $MissTime_{best}$ effectively divides the thermostat schedule into two highly regular periods: occupied and unoccupied. In these cases, our $MissTime_{best}$ variant is similar to one defined in prior work [10], and written below, which defines average $MissTime_{best}$ over n days assuming that occupied interval is $[T_{arrive}, T_{leave}$, the unoccupied interval is $[T_{leave}, T_{arrive}]$, the conditioned time as $[T_{on}, T_{off}]$, and the unconditioned time as $[T_{off}, T_{on}]$.

$$\frac{\sum_1^n max(0, T_{leave} - T_{off}) + max(0, T_{on} - T_{arrive})}{n}$$

Here, the conditioned time represents the time that the thermostat is at the setpoint temperature, and the unconditioned time represents the time the thermostat is at the setback temperature. Of course, a simple way to achieve a MissTime of zero (in either case) is to never alter the thermostat setpoint, even when a home is not occupied. Thus, for both MissTime variants, there is a tradeoff between HVAC energy usage and MissTime: the lower the HVAC energy usage, the harder it is to achieve a low MissTime, while the higher the HVAC energy usage, the easier it is to achieve a low MissTime. We use the term *WasteTime* to quantify wasted HVAC energy, which represents the time a home is unoccupied but the HVAC system is on. We define two variants of WasteTime similarly to MissTime. Thus, as above, $WasteTime_{best}$ is calculated as:

$$-\frac{\sum_1^n min(0, T_{leave} - T_{off}) + min(0, T_{on} - T_{arrive})}{n}$$

iProgram computes thermostat schedules for homes that specify when the thermostat should be at a setpoint or a setback temperature. However, it does not specify the exact setpoint and setback temperatures for the user. The setpoint temperature at different

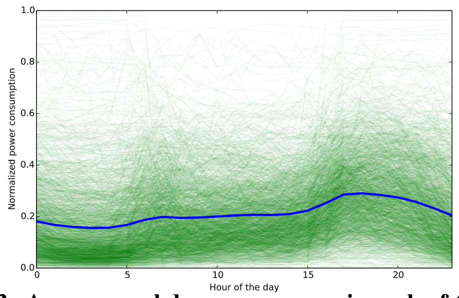


Figure 3: Average weekday power usage in each of the 16,800 homes in our town dataset.

times should be defined by a home's occupants, since it depends on their subjective notion of comfort, i.e., some people may like their house warmer or colder than others. iProgram could possibly aid in setting setback temperatures based on the regularity and length of a home's occupancy pattern. For example, the more regular and predictable a home's schedule, the deeper the setback that is possible without causing discomfort. In addition, setting the depth of the setback also depends on a home's size and insulation, since these attributes affect the time it takes to change the temperature back to the setpoint temperature. Thus, we leave setting of the setback temperature's depth as future work: iProgram computes the schedule, while the user sets their desired setpoint and setback temperature.

2.2 Data Analysis and Observations

Before describing our technique for inferring thermostat schedules from smart meter data, we first analyze smart meter data from a small town. Our dataset includes a year's worth of power data at five minute granularity for 16,800 residential homes in a small town, which gets electric service from a small municipal utility owned by the town. Thus, our dataset represents the actual power usage patterns from every household managed by a single utility.

Figure 1 shows a histogram of the average distance between consecutive weekdays for each home in our dataset. The y-axis is the number of homes with the corresponding value on the x-axis, which represents the normalized average distance in the hourly power data time-series between all consecutive weekdays. Here, we normalize by dividing the distance by the home's mean power consumption. While there are many possible distance functions for two time-series, we use Dynamic Time Warping (DTW). As opposed to Euclidean distance, where small differences or offsets in two time-series can result in large distances, DTW adjusts for slight offsets in time and is thus more robust to irregular data.

We use the DTW distance to get a sense of the regularity (or predictability) of each home's pattern of power usage. Since prior work shows that power usage tracks occupancy, regularity in power usage indicates regularity in occupancy and a conventional programmable thermostat is only useful if there is some regularity in a home's pattern of occupancy. Figure 1 shows that most homes are highly regular with many homes having a distance of near zero. While there is a heavy tail of less regular homes, they represent a small fraction of the 16,800 homes in our dataset. The graph demonstrates that the vast majority of homes exhibit similar levels of high regularity, which indicates that they can benefit from correctly scheduling a programmable thermostat.

Figure 1 shows that a large fraction of homes exhibit similar levels of regularity in their power data, which based on prior work also implies regularity in their occupancy patterns. We are also interested in the average pattern of power usage each day because that pattern ultimately dictates our schedule. Figures 3 and **??** plots the average daily power usage for weekdays and weekends, respectively, at five minute intervals for all 16,800 homes in our town dataset. In addition, Figure 4 shows the average pattern of usage for both weekends and weekdays across all homes. Weekends are similar to weekdays except for slightly higher average power usage that is slightly more irregular across homes.

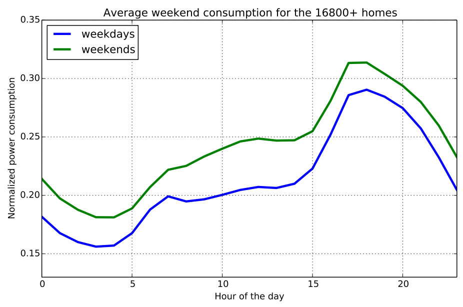


Figure 4: Average weekend and weekday power usage.

As expected, both graphs generally show two peaks during the morning (6am-8am) and evening (4pm-6pm) aligning with breakfast and dinner with a dip in power usage overnight (11pm-5am). The overnight usage is generally lower than the mid-day usage. In addition, the nighttime usage (6pm-11pm), while lower than the peak evening usage, is greater than the mid-day and overnight usage. While the weekend graph shows these same trends, they are generally less well-defined (each period is "fuzzier"), which indicates that occupancy patterns are more variable over the weekend. This is intuitive, since weekends are not defined by the work and school. The weekend graph also shows slightly higher power usage than the weekday, which correlates with higher levels of occupancy.

Ultimately, based on the graphs, we can divide an average home into one of three periods: occupied and awake, unoccupied, and occupied and sleeping. When occupied and awake (roughly from 6am-8am and 4pm-11pm) power usage is high, when unoccupied (8am-4pm) power usage is lower, and when occupied and sleeping (11pm-6am) power usage is lowest. Given these results, the goal of our technique in the next section is to identify these three periods from smart meter data. While these graphs represent general trends across all homes, their "fuzziness" shows that each home exhibits its own unique trend that diverges from this general trend. Thus, iProgram finds per-home schedule based on each homes' trend.

3. INFERRING SCHEDULES

A baseline approach for determining thermostat schedules from smart meter data involves combining two distinct, but well-known, approaches: (i) using an occupancy detection technique to infer occupancy from smart meter data [3, 14], and then (ii) use an approach that takes the occupancy data to determine a schedule that minimizes miss times [10,15]. However, both approaches have limitations, and we found that naïvely combining them results in poor thermostat schedules in practice.

Prior occupancy detection techniques are i) not perfectly accurate and ii) detect fine-grained occupancy, e.g., unoccupied periods as small as a few minutes. We found that even slight inaccuracies in occupancy, when occurring over a large number of days, reduce the accuracy of a static thermostat schedule, as the schedule is based on inaccurate information. In addition, the small periods of oc-

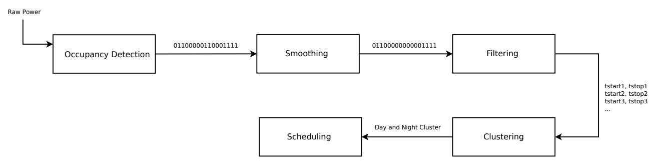


Figure 5: iProgram's pipeline for generating a schedule.

cupancy or non-occupancy existing techniques detect are typically not highly regular across days, and, thus, are not useful in deriving a static schedule for a conventional programmable thermostat. Similarly, the approach taken prior work on self-programming thermostats [10, 15] only infers static schedules for a single zone home and does not i) handle homes with multiple zones or i) make dynamic adjustments to static schedules based on short-term deviations in occupancy patterns. Our proposed approach is designed to handle these drawbacks of both approaches.

Our approach involves: (i) determining a static schedule for a home based on long-term repeating patterns of usage and occupancy, (ii) dynamic techniques to adjust the inferred static schedule based on short-term and gradual long-term changes in daily patterns, and (iii) extensions to handle multiple zones that separate bedroom zones from the home's primary zone.

3.1 Deriving A Static Schedule

Our approach first determines a static thermostat schedule for a home by distilling periods of occupancy and non-occupancy. Our approach ensures that inferred occupancy periods are long and more likely to follow regular daily (or weekly) patterns and where we have high confidence in their accuracy. The process of inferring a static thermostat schedule involves a series of pipelined steps depicted in Figure 5: occupancy detection, smoothing, filtering, clustering, and scheduling. The pipeline defines a general and intuitive sequence of steps to transform raw smart meter data into a thermostat schedule. We discuss each step in-turn below, including how we implement it, potentially among various options, and why we chose our specific implementation.

Step 1: Determine Occupancy. There are a variety of previously proposed methods to detect occupancy from smart meter data using various classifiers, including simple thresholding, k-Nearest Neighbor, Support Vector Machines, and Hidden Markov Models [3, 14]. Each technique trains a classifier based on features of the smart meter data, typically the power value, variance, and range, that correlate with binary occupancy. The classifier effectively discretizes smart meter data at each time period t to be one of two states: occupied or non-occupied. Prior work shows that no one classifier works best across all homes.

In this paper, we chose to use a classifier based on a simple Hidden Markov Model (HMM) [14], which associates hidden, i.e., unknown, states with different power levels, e.g., P_0 for lowest power level state, P_1 for next lowest powered state, etc., and visible states with power consumption. HMM's adhere to the memory-less property, such that a visible change in power at time t_i only depends on the hidden power state at time t_i, which captures the influence occupancy has over the home's power consumption.

The HMM uses two sets of probabilities learned during a training phase: transition and emission probabilities. Transition probabilities capture the probability of a home being occupied at time t_{i+1} given that a home is occupied at time t_i. Intuitively, the transition probability captures the inertia in a home's occupancy status where an occupied home tends to remain occupied for a long duration. Likewise, if a home is in a lower power state because it is unoccupied, then it has a high probability of continuing to remain in that state. The inertial property of the HMM is the primary reason we chose it for iProgram, as the regular patterns of occupancy tend to be long, rather than short, periods, i.e., people generally do not leave home for only a few minutes (or even an hour) at regular intervals every day or week.

Emission probabilities indicate the probability of emitting a particular power level given a particular power level state (P_0, P_1, P_2, etc.). During classification, the HMM uses the transition and emission probabilities to assign values to the hidden states based on the power readings. We associate the lowest power state (P_0) with the unoccupied state and the remaining states with the occupied state.

Step 2: Smoothing and Filtering. Any raw occupancy detection technique from above is sensitive to changes in power, since these changes act as the classifier's features. Our HMM is no different and also construes any significant rise in power with the occupied state. Such fluctuations in power are not always the result of occupants' use of interactive devices, but may also result from background devices, such as refrigerators, air conditioners, heaters, dehumidifiers, etc., regulated by environmental sensors or timers. In some cases, these devices my activate concurrently and generate large increases in power. Thus, we post-process the discretized series of hidden states our HMM produces by smoothing out spikes in the discretized power levels that indicate brief periods of occupancy or non-occupancy that are unlikely to be repeated.

Our iterative smoothing algorithm looks for series of power levels that are less than a threshold K. If the algorithm finds such an interval, it does not simply smooth it, but examines the surrounding intervals for context: the algorithm only smooths an interval, i.e., eliminates it, if the previous interval has a length $> K$ or the successive interval has a length $> K$. Once the algorithm smooths an interval, it starts again from the beginning to determine if the consolidation leads to additional smoothing of past intervals. In our experiments, we have found that $K = 30$ minutes works well.

Finally, after smoothing, we filter the resulting trace of smoothed discretized power levels to extract intervals of length greater than H hours. Our intuition is that longer periods of occupancy and non-occupancy are more likely to repeat and, thus, be useful in scheduling a thermostat. Our filter converts each day's trace into a set of zero or more intervals of the form $\{(start_1, stop_1), (start_2, stop_2), \ldots\}$. Figure 6 and Figure 7 provides examples of these intervals in two homes. As expected, the figures show that most low power intervals occur at night or during the work day.

Step 3: Cluster Unoccupied or Low Activity Periods. After post-processing a home's discretized power data by smoothing and filtering it, we apply a clustering algorithm to determine reasonable clusters for generating schedules. While we could use any clustering algorithm, we chose DBSCAN [9], since it robust to outliers. Outliers are likely to occur in our data, since even people that follow a regular pattern of occupancy may diverge significantly on occasion, e.g., due to inclement weather, illness, etc.

One challenge with clustering is that we do not know *a priori* how many regular intervals any particular home might exhibit. In a large

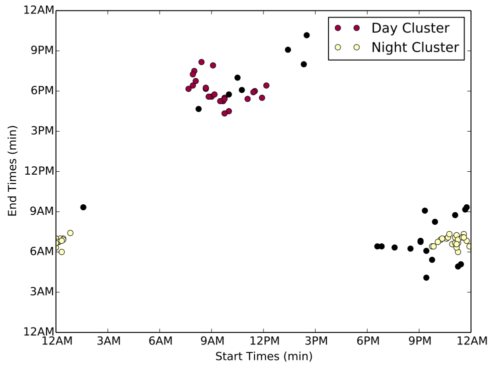


Figure 6: Day and night clusters for Home A.

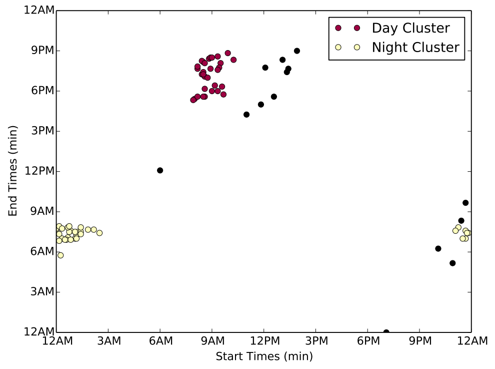


Figure 7: Day and night clusters for Home B.

dataset, such as our town dataset that includes over 16,000 homes, it is difficult to manually determine a customer number of intervals that works best for each home. However, as our figures above show, we found that two clusters are evident in most homes' data: a daytime cluster that signifies the start and end of the unoccupied period and a nighttime cluster that signifies when occupants went to sleep. We categorize the home as being occupied and awake during the time outside these clusters. Thus, we use DBSCAN to find a dense cluster of unoccupied daytime intervals, which we define as occurring between 6am and 10pm. Figures 6 and 7 show the unoccupied daytime cluster in red. To find nighttime intervals, we use the latest arrival and earliest departure from the daytime clusters to define the nighttime period for a home. We then run DBSCAN to find a dense clusters of points that defines the nighttime.

Step 4: Compute a Repeating Schedule. The clusters from the previous step can then be employed to compute a static thermostat schedule. One approach to computing a schedule is to use the centroid of each cluster representing repeating occupied periods to determine when the HVAC system should be turn down. Another approach is to use the mean start times and end times of the daytime and nighttime intervals above to determine the setback period each day. The approach proposed by Gao and Whitehouse [10], is to determine a maximum length setback period for a particular MissTime given arrival and departure times over a long period. A slight variant of the algorithm can also determine the minimum MissTime for a specified duration of the setback period. We can apply these algorithms directly using the collection of arrival and departure times we infer above.

3.2 Thermostat-specific Scheduling

The approach presented above determines a single static daily schedule for a home. However, real homes and actual programmable thermostats are more complex than "one schedule fits all" approach. For example, a home may have multiple thermostats to independently control different rooms of the house. Thus, our approach is designed to handle the most common case of a two-zone house with a thermostat is the living area and one or more thermostats in the bedrooms. Similarly, programmable thermostats come in many flavors: a 7-day thermostat supports seven different schedules (one for each day of the week), while a "5-2" thermostat supports two schedules (one for weekdays and one for weekends), and a "5-1-1" thermostat supports three schedules (one for weekdays, one for Saturday, and one for Sunday).

iProgram handles these real-world factors by using the clusters to determine a custom schedule for each scenario. Deriving a schedule for a multi-zone house treats the primary zone and the bedroom zones differently based on the observed daytime and nighttime clusters. Table 1 summarizes how iProgram determines the schedule for each zone: when the house is unoccupied, both the primary and bedroom zones are turned off (or equivalently, set to a pre-set "away" temperature); when the house is occupied during the daytime, the primary zone is on and the bedroom zones are set to either on or away depending on user preferences; finally, when the house is occupied at night, the primary zone is turned off (or set to away) while the bedroom zone is turned on.

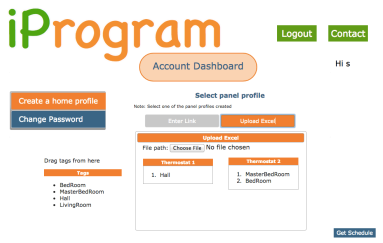
	Living	Bedroom
Unoccupied	off	off
Daytime occupied	on	on or off
Nighttime occupied	off	on

Table 1: Determining schedules for a two-zone home

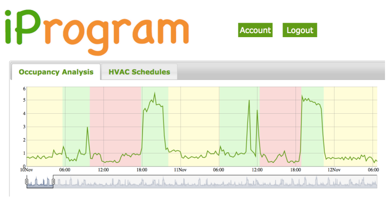
3.3 Dynamic Scheduling

While iProgram is capable of determining a static schedule for multiple zones, in practice, daily occupancy patterns exhibit both short-term and long-term deviations from the inferred repeating patterns. iProgram uses continuous online learning over smart meter data to adapt to such dynamics. Short one-time deviations occur when a home is occupied when it is expected to be unoccupied or unoccupied when it is normally occupied, e.g., when a user stays home due to an illness or is away for a one-time event. To handle such deviations, iProgram runs the occupancy detector presented earlier on live smart meter data and if it determines that the home is occupied, it can set the thermostat schedule to "away" mode. The thermostat goes back the normal mode when occupancy is detected again. Similarly, if a house is occupied during a normally unoccupied period, the thermostat may be temporarily set to on. We note that such dynamic scheduling is feasible only for programmable thermostats that are remotely programmable—since expecting users to manually reprogram thermostats on a frequent basis in infeasible.

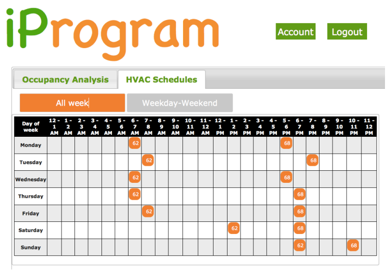
Our system also periodically runs the four step pipeline on new data to detect and handle long-term changes in activity patterns and occupancy. The period can be once a day or once a week as desired; the algorithm can also be invoked dynamically if frequent deviations from the computed schedule are observed. In this case, a static schedule is recomputed on more recent smart meter data to account for persistent changes in occupancy patterns. Since running the entire four step pipeline can be computationally expensive, an incremental approach can be used where occupancy patterns are determined in the first step and compared to the previously computed occupancy. The remaining step are run to compute a new



(a)



(b)



(c)

Figure 8: Screenshots of iProgram's web service, including user profile and thermostat configuration (a), inferred occupancy visualization(b), and generated thermostat-specific schedule(c).

schedule only if more than a threshold amount of deviation is observed in the occupancy patterns.

4. IMPLEMENTATION

We implement iProgram as an open web service that enables consumers to upload their power data and select their type of thermostat and then visualize their occupancy patterns and generate a custom thermostat schedule based on them. The architecture consists of six modules: a profile manager, storage engine, occupancy analyzer, schedule generator, visualization engine, and API manager.

The profile manager handles account creation, user authentication, and user meta-data, such as their username, email address, power consumption data, and details of their home's thermostat. The profile manager interacts with the storage engine to store and retrieve information user profiles, as well as power consumption data. Users may upload power usage data directly as CSV files or provide a URL for a third-party meter storing the data. We currently support eGauge power meters, but intend to add additional third-party meters in the future, such as TED. Figure 8(a) shows a screenshot of iProgram's account dashboard and profile manager.

The occupancy analyzer module applies the pipeline, including the HMM occupancy detection, smoothing, and filtering, from the previous section to derive occupancy from the uploaded power data. The module stores the discretized occupancy information in the storage engine. The occupancy information is then read by the scheduler generator, which is capable of generating schedules for 1-day, 5-2-day, and 7-day programmable thermostats. The schedule generator supports multiple types of scheduling algorithms, including a conservative one that minimizes MissTime and a configurable one that permits users to set a threshold on the MissTime.

The visualization engine displays the occupancy information and the generated schedules to the user. Figure 8(b) shows a sample of the occupancy information for a home, where green indicates the home is occupied and awake, red indicates it is unoccupied, and yellow indicates it is occupied and sleeping, and Figure 8(c) shows a sample thermostat schedule for a home. Finally, iProgram exposes an external REST API via the API manager, which provides a programmatic interface for networked thermostats to auto-program themselves using iProgram's schedules. The API exposes information in JSON format, and could be extended to offer If-This-Then-That (IFTTT) recipes to trigger automated emails to users based on certain events, e.g., if the schedule changes.

iProgram's web service is built using Django, a popular Python-based web application framework. We use SciPy stack, which includes assortment of scientific computing libraries for Python, to process, store, and analyze power data. We also use an R library [18] to implement our HMM for learning and inference. We use dygraph, a Javascript graphing library for displaying occupancy data, and a sqlite3 database to store each user's profile, power data, occupancy information, and thermostat schedules.

5. EVALUATION

One challenge in evaluating iProgram using our town dataset is that we do not have ground truth occupancy for the 16800 homes. As a result, while our data analysis from Section 2 indicates that iProgram would prove useful in these homes, we cannot use them in evaluating how well it schedules thermostats based on ground truth occupancy. Instead, we evaluate iProgram on a set of 20 homes where we either directly monitor ground truth occupancy or can derive it from fine-grained power monitoring of individual appliances and circuits. The first two homes in our evaluation are homes A and B from the Smart* dataset [2]. We directly monitored ground truth occupancy in these homes by instrumenting the occupants' smartphones to record their GPS location.

Since the logistics of directly monitoring both power and occupancy in a large number of homes is challenging, we also take 18 homes from the Pecan Street dataset [19], which includes minute-level power data for entire homes and individual circuits. To infer ground truth occupancy in these homes, we approximate it by only looking at the periods where circuits that power interactive devices use power. The circuit-level data enables us to directly filter out any background devices that use power, rather than infer whether a change in power is due to an interactive device. By looking only at circuits with interactive devices, we know that any change in power correlates with occupancy. While this technique may result in some false negatives during when someone is home but not interacting with any devices, we have found it to be accurate on homes where we directly monitor ground truth occupancy. Figure 9 shows one example from Home B of the Smart* dataset, where we directly monitor ground truth occupancy and overlay power data and "on" events from interactive circuits. The graph shows that anytime power rises above a minimal level the home is occupied.

5.1 Inference Accuracy

We first evaluate how well iProgram's inferred unoccupied daytime and nighttime intervals match ground truth, since iProgram uses these intervals to define the thermostat schedule. Figure 10 shows the percentage of our inferred intervals that overlap with the corresponding ground truth intervals over a three month period. The result shows that for 15 of the 20 homes in our dataset, iProgram finds 60% or more of these intervals, while it finds >80%

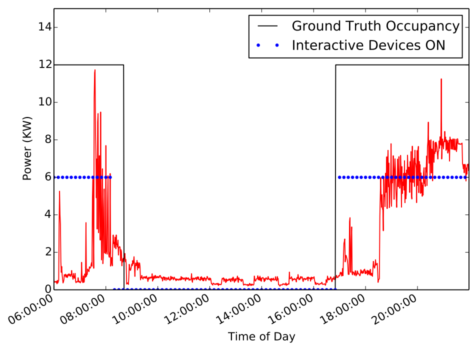


Figure 9: Power usage from interactive circuits is an accurate proxy for ground truth occupancy.

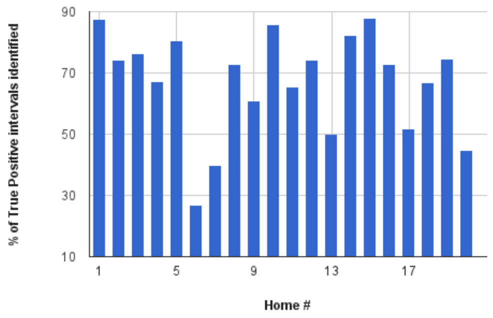


Figure 10: Percentage of unoccupied intervals detected.

of them intervals in five homes. Of course, the results vary by home, since the mix of devices and their behavior is different in each home. However, the results indicate that iProgram is able to detect a large percentage of these unoccupied daytime and nighttime periods. Figure 11 then quantifies the amount of overlap our inferred intervals have relative to the ground truth. For example, an interval we detect may be five hours, while the actual interval in the ground truth data may be six hours. We plot the inaccuracy of our detected intervals as a percentage of the average inferred interval length. We see that for all but two homes, our inferred unoccupied intervals are less than 30% of the length of the ground truth interval. Ultimately, our results indicate that over the 20 homes we i) find a large percentage of the unoccupied daytime and nighttime intervals in a large fraction of the homes and ii) these detected intervals are close in the length to the actual periods.

5.2 Scheduling Accuracy

After detecting the intervals above, we then define our thermostat schedules by clustering the start and end times of the intervals. As we discuss in Section 3, a variety of schedules are possible based on a user's preference for balancing either variant of MissTime and WasteTime. Of course, the accuracy of a schedule is dependent on the regularity of a particular home's occupancy pattern. For example, highly regular home will result in tight clusters for the start times and end times of intervals each day, which enables the schedule to simultaneously minimize both MissTime and WasteTime. Likewise, irregular homes will result in loose clusters that present a trade-off between minimizing MissTime and minimizing WasteTime, as increasing one causes the other to decrease.

We first evaluate the performance of a conservative schedule

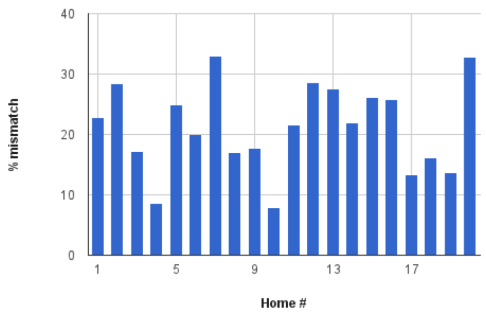


Figure 11: The inaccuracy of detected intervals as a percentage of the average length of the ground truth interval.

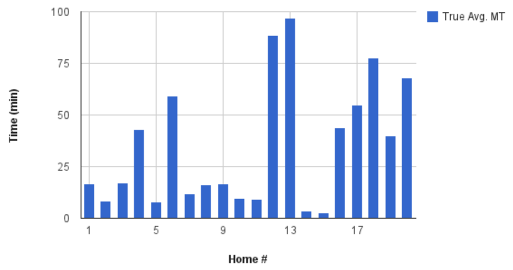


Figure 12: Average $MissTime_{best}$ for each of our 20 homes.

that has zero $MissTime_{best}$ based on our inferred periods of occupancy, which represents the best miss time a particular type of programmable thermostat can achieve when statically dividing the day into a discrete number of fixed periods. To ensure$MissTime_{best} = 0$, we compute the schedule for the unoccupied and sleeping periods based on the earliest point in the cluster representing the start times and the latest point in the cluster indicating the departure times. Figure 12 shows the average $MissTime_{best}$ for ground truth occupancy. For 80% of homes using this conservative schedule, the average $MissTime_{best}$ per day is under 60 minutes. More importantly, for about half of the homes, the average $MissTime_{best}$ is under 20 minutes. While such a schedule may be wasteful for homes with irregular occupancy patterns, it illustrates that iProgram is capable of setting a schedule that does not significantly impact occupant comfort.

Figure 13 shows the $MissTime_{best}$ from Figure 12 along with the $MissTime_{best}$ for a baseline weekday thermostat schedule of 8am to 6pm, which all EnergyStar-compliant thermostats are preprogrammed with by default. As the figure shows, for 19 out of 20 homes, iProgram's schedule does significantly better than the default schedule. $MissTime_{best}$ represents the deviation from the best schedule possible with a 5-2-day programmable thermostat with a static schedule. We also compare with $MissTime_{smart}$ which represents the best schedule possible with a smart thermostat that dynamically adjust the thermostat based on real-time occupancy patterns. Figure 14 shows the $MissTime_{smart}$ for each home using a schedule based on the intervals iProgram infers and a schedule based on intervals derived from ground truth occupancy, which represents the best iProgram could do. The average $MissTime_{smart}$ across all homes when using iProgram is about ≈ 64 minutes, while the average $MissTime_{smart}$ using a schedule based on ground truth occupancy is ≈ 25 minutes. Thus, the result shows that iProgram's $MissTime_{smart}$ is only an hour off the best possible time using a smart thermostat without requiring an upgrade to a programmable thermostat or the installation of differ-

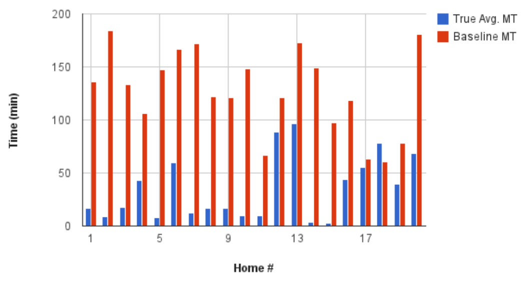


Figure 13: Average $MissTime_{best}$ for both our technique and a default baseline thermostat schedule of 8am to 6pm.

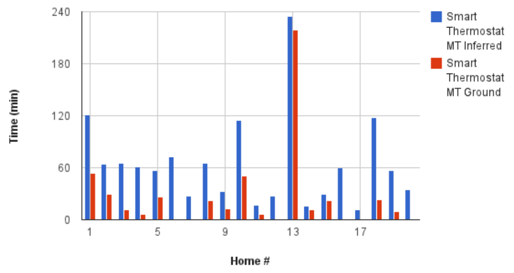


Figure 14: Average $MissTime_{smart}$ for a iProgram's schedule using inferred occupancy patterns and using ground truth.

ent sensors. Note also that $MissTime_{smart}$ represents an upper bound on smart thermostat performance.

Finally, we also show that even with a conservative approach to thermostat scheduling that focuses on minimizing $MissTime_{best}$, iProgram's thermostat schedules result in significant decreases in $WasteTime_{best}$. Figure 15 shows cumulative time that the thermostat is at the setback temperature, either due to occupants sleeping or due to a home being unoccupied. The graph shows that for 80% of the homes in our 20 home dataset, iProgram reduces the conditioned time by over 10 hours each day on average when compared with a thermostat that maintains its setpoint all day.

5.3 Balancing MissTime and WasteTime

We also apply the algorithm by Gao and Whitehouse [10] to enable users to define a tradeoff between comfort and energy-efficiency. While iProgram conservatively determines the schedules above based on $MissTime_{best}$ of zero, relaxing the $MissTime_{best}$ enables reductions in $WasteTime_{best}$ and more energy savings. Figure 16 shows how the actual average $MissTime_{best}$ and $WasteTime_{best}$ varies as iProgram changes the threshold $MissTime_{best}$ it uses to compute its thermostat schedule for Home A from the Smart* dataset. For an inferred $MissTime$ of zero minutes, the average $MissTime_{best}$ is $\approx$ 8.44 minutes. Thus, if a user were use this conservative schedule, she would experience discomfort for $\approx$ 8.44 minutes. However, the corresponding average $WasteTime$ is high at $\approx$ 160 minutes. In this case, if a user were to choose a higher $MissTime_{best}$ target of, say, 30 minutes, their actual average $MissTime_{best}$ would be higher at $\approx$ 65 minutes, but their $WasteTime_{best}$ would decrease significantly from $\approx$ 160 to $\approx$ 35 minutes. The graph also shows how unconditioned time (or setback duration) increases as the schedule goes from conservative to liberal. In the extreme conservative case, the home has an unconditioned time of $\approx$4.5 hours, while for a less conservative one, the unconditioned time increases to $>$ 7.5 hours.

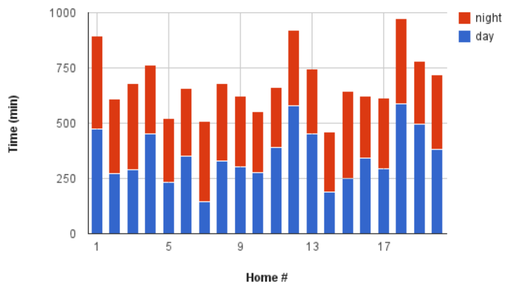


Figure 15: Total amount of time each home's thermostat is at the setback temperature due to it being unoccupied during the day or the occupants being asleep at night.

Figure 17 shows the same experiment as above for Home B in the Smart* dataset. Home B differs from Home A in that the occupants of Home B have a highly regular schedule, which results in tight clusters for unoccupied periods during the day and sleeping periods during the nighttime. As a result, setting a very conservative schedule does not adversely affect the $WasteTime_{best}$. For an inferred $MissTime_{best}$ of zero minutes, the actual average $MissTime_{best}$ for the home is $\approx$16.5 minutes, while the $WasteTime_{best}$ is only about $\approx$7.5 minutes. In addition, for a $MissTime_{best}$ threshold of only five minutes, the home's $WasteTime$ drops to near zero, while the actual $MissTime_{best}$ increases to roughly 45 minutes, which suggests this home has a narrow window in which occupants arrive and leave.

Finally, Tables 2 and 3 show the unoccupied daytime and nighttime setback periods for Home A for different types of thermostats computed using the conservative schedule with $MissTime_{best} = 0$.

6. RELATED WORK

Accurately detecting occupancy from just power consumption data is an active area of research. Chen et al. [3] present an unsupervised threshold-based approach to occupancy detection using various statistical features of the power trace, such as the average power level, standard deviation, and power range. Likewise, Kleiminger et al. [14] present and evaluate multiple supervised approaches to occupancy detection, including techniques based on SVM, KNN and HMM, which first train a model using ground truth occupancy and then use it to predict future occupancy.

There is a large body of work on home heating control and HVAC scheduling. Gao et a. [10] design and evaluate a self-programming thermostat using historical occupancy data to choose optimal setback schedules for homes. Similar to iProgram, their system produces static thermostat schedules. However, iProgram does not assume availability of ground truth occupancy data via sensors. Scott et al. [20] develop an online algorithm for occupancy prediction using historically available data for past days. At any point in a day, they match a day starting from midnight until that point with historical days to see which days are most similar in terms of power consumption patterns. They then use those days to compute an occupancy probability for the rest of the day and use a threshold on the occupancy prediction probability to determine when to schedule the thermostat. Lu et al. [15] use a Hidden Markov Model on historical motion sensor data to decide when to turn the system off.

Yet another approach to HVAC control is to determine occupancy based on GPS data. Gupta et al. [11] use real time GPS data to predict travel-to-home times and use those estimates to preheat or

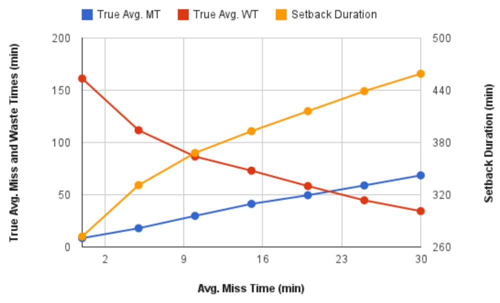


Figure 16: Tradeoff between $MissTime_{best}$ and $WasteTime_{best}$ in Home A.

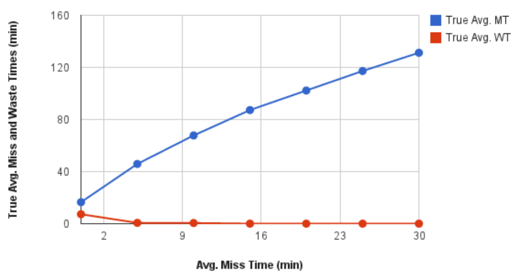


Figure 17: Tradeoff between $MissTime_{best}$ and $WasteTime_{best}$ in Home B.

	1 Day Programmable	5-2 Programmable	7 Day Programmable
Monday	11:54AM - 16:29PM	11:55AM - 16:27PM	11:28AM - 17:25PM
Tuesday	11:54AM - 16:29PM	11:55AM - 16:27PM	11:21AM - 16:30PM
Wednesday	11:54AM - 16:29PM	11:55AM - 16:27PM	11:51AM - 17:15PM
Thursday	11:54AM - 16:29PM	11:55AM - 16:27PM	9:41AM - 16:20PM
Friday	11:54AM - 16:29PM	11:55AM - 16:27PM	12:06PM - 17:35PM
Saturday	11:54AM - 16:29PM	-	-
Sunday	11:54AM - 16:29PM	-	-

Table 2: Day Schedules for Home A

	1 Day Programmable	5-2 Programmable	7 Day Programmable
Sunday	0:25AM - 06:04AM	00:25AM - 06:00AM	00:46AM - 06:00AM
Monday	0:25AM - 06:04AM	00:25AM - 06:00AM	00:20AM - 06:25AM
Tuesday	0:25AM - 06:04AM	00:25AM - 06:00AM	00:00AM - 06:39AM
Wednesday	0:25AM - 06:04AM	00:25AM - 06:00AM	00:16AM - 06:00AM
Thursday	0:25AM - 06:04AM	00:25AM - 06:00AM	23:51PM - 06:25AM
Friday	0:25AM - 06:04AM	00:13AM - 06:20AM	23:57PM - 06:20AM
Saturday	0:25AM - 06:04AM	00:13AM - 06:20AM	00:17AM - 06:35AM

Table 3: Night Schedules for Home A

precool a home. Similarly, Hong et al. [12] use GPS traces to model when an occupant will return home to drive thermostat preheat decisions. They look for conditionally similar days in the past, compute a distribution of arrival times on those days, and then use it to estimate when an occupant will arrive back home. Unlike this prior work, iProgram computes static schedules based on data from a readily available sensor—a utility smart meter—without requiring the installation of a new thermostat or additional sensors.

7. CONCLUSION

This paper presents iProgram: a system for deriving schedules for programmable thermostats by analyzing smart meter data. We implement iProgram as an open web service that is accessible to anyone with smart meter data and a programmable thermostat. We show that the thermostat schedules iProgram derives schedules that are significantly better than the default 8am-6pm schedules in programmable thermostats, and come close to the best possible schedules achievable by smart thermostats.

8. REFERENCES

[1] Y. Agarwal, B. Balaji, S. Dutta, R. Gupta, and T. Weng. Duty-cycling Buildings Aggressively: The Next Frontier in HVAC Control. In *IPSN*, April 2011.

[2] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht. Smart*: An Open Data Set and Tools for Enabling Research in Sustainable Homes. In *SustKDD*, August 2012.

[3] D. Chen, S. Barker, A. Subbaswamy, D. Irwin, and P. Shenoy. Non-Intrusive Occupancy Monitoring using Smart Meters. In *BuildSys*, November 2013.

[4] Ecobee. `http://ecobee.com`, January 2015.

[5] U.S. Energy Information Administration, Frequently Asked Questions, Residential Energy Consumption Survey (RECS). `http://www.eia.gov/consumption/residential/data/2009/`, 2009.

[6] U.S. Energy Information Administration, Frequently Asked Questions, How Many Smart Meters are Installed in the U.S. and who has them? `http://www.eia.gov/tools/faqs/faq.cfm?id=108&t=3`, 2011.

[7] V. Erickson, M. Carreira-Perpinan, and A. Cerpa. OBSERVE: Occupancy-based System for Efficient Reduction of HVAC Energy. In *IPSN*, April 2011.

[8] V. Erickson and A. Cerpa. Occupancy Based Demand Response HVAC Control Strategy. In *BuildSys*, 2010.

[9] M. Ester, H. Kriegel, J. Sander, and X. Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*, August 1996.

[10] G. Gao and K. Whitehouse. The Self-programming Thermostat: Optimizing Setback Schedules based on Home Occupancy Patterns. In *BuildSys*, November 2009.

[11] M. Gupta, S. Intille, and K. Larson. Adding GPS-Control to Traditional Thermostats: An Exploration of Potential Energy Savings and Design Challenges. In H. Tokuda, M. Beigl, A. Friday, A. Brush, and Y. Tobe, editors, *Pervasive Computing*, volume 5538 of *Lecture Notes in Computer Science*. 2009.

[12] D. Hong and K. Whitehouse. A Feasibility Study: Mining Daily Traces for Home Heating Control. In *MS*, April 2013.

[13] J. Kelso, editor. *2011 Buildings Energy Data Book*. Department of Energy, March 2012.

[14] W. Kleiminger, C. Beckel, T. Staake, and S. Santini. Occupancy Detection from Electricity Consumption Data. In *BuildSys*, November 2013.

[15] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse. The Smart Thermostat: Using Occupancy Sensors to Save Energy in Homes. In *SenSys*, November 2010.

[16] Honeywell Lyric. `http://lyric.honeywell.com`, January 2015.

[17] Nest. `http://nest.com`, January 2015.

[18] J. O'Connell and S. Hojsgaard. Hidden Semi Markov Models for Multiple Observation Sequences: The mhsmm Package for R. *Journal of Statistical Software*, 39(4), 2011.

[19] Pecan Street. `http://www.pecanstreet.org/`.

[20] J. Scott, A. Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. PreHeat: Controlling Home Heating Using Occupancy Prediction. In *Ubicomp*, September 2011.