Home Temperature Sensor Network

Carnegie Mellon University
18799M Advanced Machine Learning Class
Spring 2013

Vadim Zaliva, vzaliva@cmu.edu

1 Problem Statement

The motivation for this project was an observation of the variation in temperature among the rooms in the author's home. The practical issue at hand was that the home HVAC system tries to control the temperature based on only the measurements at a single location (the temperature sensor in the thermostat). For example, if the thermostat is set to 20 celsius[1] the actual temperature in other rooms can vary by as much as 5°C. It could be as low as 15°C in one room and as high as 25°C in another. This is a well-known problem with single-zone HVAC systems.

Apparently there is a complex, non-linear dependency among the temperatures in various rooms. Besides the configuration of the building, the temperature of a given room depends on many factors, such as the outdoor temperature, whether the HVAC is running, and whether the doors and windows are open. It would be difficult to build an analytical model based on the laws of physics taking all these factors into account. However, this seems to be a good machine learning problem to learn the temperature distribution. If successful, there are many practical applications, such as:

1. Being able to control the temperature in any room indirectly by adjusting the setpoint of the thermostat.

2. Being able to discover which factors affect temperatures in individual rooms, so those factors could be manipulated to better control the temperature or to save energy. For example, some doors or windows might need to be opened or closed during different times of the day to take better advantage of natural heating or cooling using outside air.

3. Looking at room temperatures alone, the system may infer the state of the doors, the windows, or the HVAC system and warn the user if, for example, a window was accidentally left open.

To be able to analyze room temperatures, the data must be collected first. We looked first at available datasets, including CMU *Sensor Andrew* project [7]. While some data was available, it covered a larger area, and the configuration of the environment was

[1] all temperatures in this report are reported in degrees Celsius, denoted as °C.

not fully understood. Therefore, we have chosen to collect a smaller dataset in a well controlled environment where we were very familiar with the configuration.

The goals we set for this project were:

1. Collect temperature data from one house.
2. Using machine learning, build a model of the temperature distribution across various locations within the house.
3. Use this model to predict temperatures in various rooms based on temperature measurements from other rooms.
4. Try to infer the state of doors and windows (opened or closed) based on temperature measurements. (Stretch goal – time permitting)

2 Data Collection

2.1 Hardware

To collect the data, we needed to outfit each room with a wireless temperature sensor. Looking at available products, we were unable to find an inexpensive wireless temperature sensor, so we decided to build our one ourselves.

Our hardware design was based on similar wireless sensor designs using *XBee* wireless modules [5, 2, 3]. Our design is shown in Figure 1. It uses low-power XBee [1] radio modules which were programmed to wake up from a sleep mode, record, average, and send a batch of data samples approximately every six seconds.

An XBee module has a built-in multi-channel DAC which measures an input voltage comparing it to the reference voltage. We used two inputs to monitor the output of a temperature sensor (temperature) and the power supply voltage (battery status). Each sensor was individually calibrated to take into account variations in component characteristics. 3.3V reference voltage was provided by an *LM3940* micropower, two-terminal, band-gap voltage reference chip.

Because we planned to make just a handful of such sensors, it did not make economic sense to order a custom PCB board, so we just soldered them onto prototyping boards. The assembled sensor is shown in Figure 2.

The house was already equipped with a *Filtrete*™ Wi-Fi Remote Programmable Thermostat, which is a brand name for a *3M CT-50* thermostat with a WiFi *USNAP* module installed as shown in Figure 3. It could be controlled via API [6], but in this project, we only needed to query whether it was currently heating, cooling, or idle. Additionally, we queried the thermostat's built-in temperature sensor, which was used as an additional temperature sensor.

2.2 Experimental Environment

The data was collected in a residential, two-story, hillside townhouse with a living area of approximately 2,000 square feet. It had a single zone HVAC system with the thermostat located in the living room on the second floor. Figure 4 shows a map of the

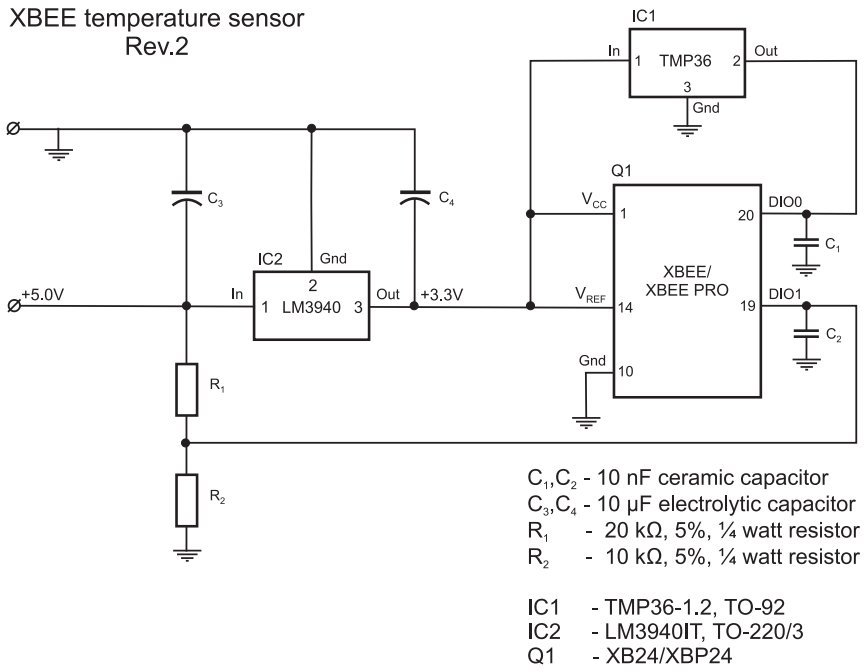

Fig. 1: Temperature Sensor Circuit Diagram

house with the locations of temperature sensors marked with numbered, yellow circles. The yellow circle with number *T* shows the location of the thermostat, which was also used as a sensor.

A PC with Linux using a USB XBee Pro adapter was used to collect data from all sensors. Rooms located closer to the PC used sensors with a regular XBee 1mW module. In rooms located farther away, the XBee Pro module was used, providing 63mW in transmission power.

2.3 Software

Our custom data collection software runs under Linux OS and consists of a collection of Python scripts. The main script reads measurements from the XBee interface and writes them as a CSV file. It has a config file with calibration coefficients for each sensor and applies the appropriate scaling factor to each sample, based on the sensor ID.

To take the outdoor temperature into account, we needed to collect data about the temperature in the vicinity of the house. However, installing additional sensors outdoors was problematic because of the lack of a nearby power outlet and the risk of exposing the electronics to the elements. Instead, we decided to record data from an existing weather station in the vicinity. *Weather Underground* `www.wunderground.com`

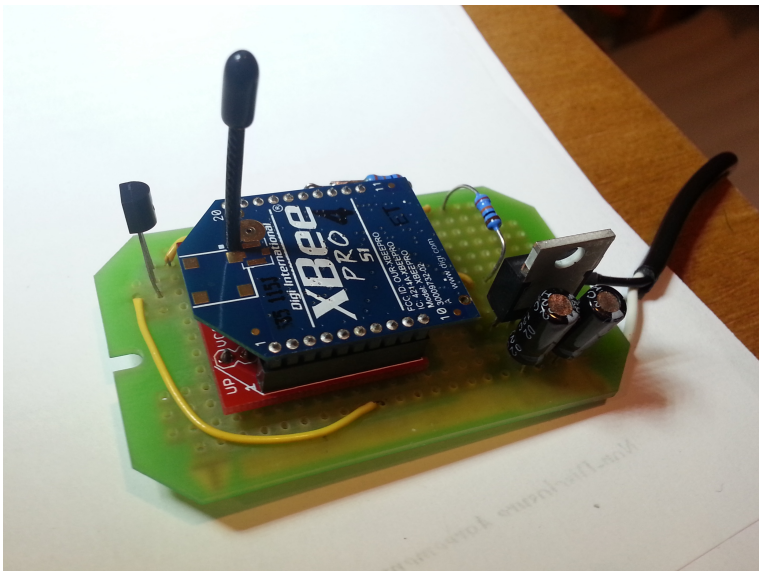

Fig. 2: Sensor

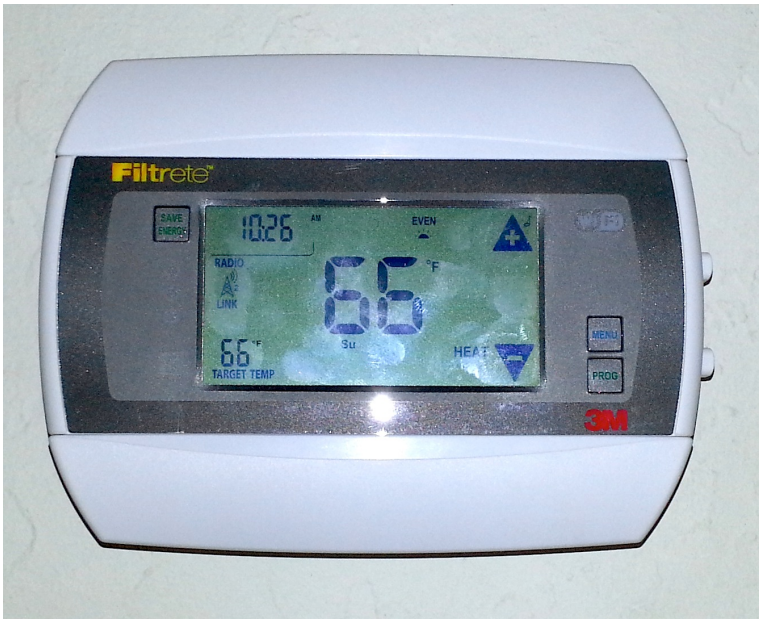

Fig. 3: CT-50 Thermostat

provides API feeds for many amateur and professional weather stations. We were able to find a station located just 2 miles from the house. A python script was written to do

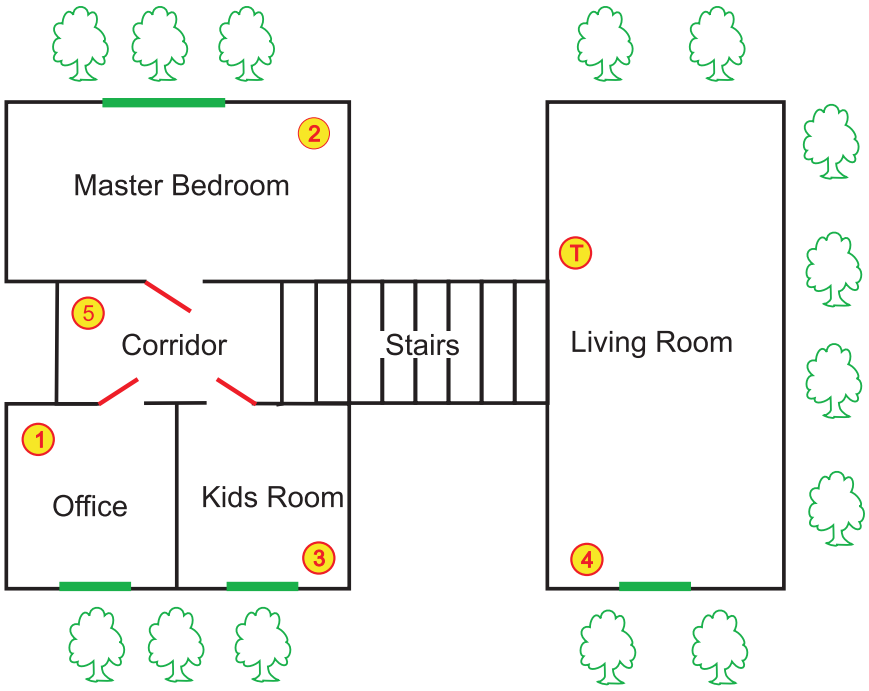

Fig. 4: House map

periodic API calls to *Weather Underground* API, decode the JSON response, and write the current reported temperature to another CSV file.

Yet another script made continuous API calls to the thermostat over the home WiFi network and recorded the status of the fan (on/off) and HVAC (cooling/heating/idle) into a CSV file.

Since the data collection for the experiment was supposed to run for an extended period of time, we needed an easy way to monitor it without watching the CSV files directly. For that purpose, we used COSM web service `https://cosm.com/`. This site allows users to inject data using API coming from various sensors and to monitor it through the web browser. Using an additional Python script, we periodically injected new data from all CSV files into COSM. The results can be viewed at our "feed" page: `https://cosm.com/feeds/118451`. Additionally COSM provides notification services. We have set up Twitter notification for the following events:

1. Battery voltage in a sensor goes below 4.5V.
2. Temperature in any of the rooms falls below 15°C.

All source code was published at GitHub: `https://github.com/vzaliva/xbee_temp_sensor`.

3 Data Analysis and Modelling

3.1 Data Normalization

Raw data from the sensors needed to be pre-processed. This was done using R script, with raw CSV data files as input. The output used CSV files with normalized data.

First, we observed that the data contained some outliers. For example, during sensor battery replacement, there were momentary temperature measurement spikes of up to more than 100°C. To handle the outlier samples, we applied *Chauvenet's criterion* to identify and remove them.

Some of the sensors were inoperable for various periods of time during the experiment. The reasons were various software and hardware malfunctions. Since we were analyzing the data for correlations, we could use only samples which contained measurements for all sensors. If even one of the sensors was malfunctioning, we had to exclude the malfunctioning period of time from the data set. We manually noted periods of time when some sensors were malfunctioning, and the R script automatically excluded all measurements during those periods from the dataset.

Since the times when sensor measurements were taken were not synchronized, we had to resample them. The general idea of what we did was to interpolate each of the variables (sensor measurements) individually and then re-sample interpolated functions at fixed time intervals common to all sensors. For regular sensors, we used *cubic spline* interpolation. The thermostat state variables were boolean and were interpolated using the nearest neighbour value.

Finally, the new sample values were smoothed using *Locally Weighted Scatterplot Smoothing (LOWESS)*. The Weather Underground data seemed to be already smoothed and did not need this step.

3.2 Model Selection

Each sensor could be viewed as a random variable. Moreover, we can introduce additional binary variables reflecting the state of the doors and windows (open or closed). Our working hypothesis was that the joint temperature distribution could be represented by a *Probabilistic Graphical Model (PGM)* [4]. Using our knowledge of the house environment, such as which rooms were connected by doors and whether there were windows or HVAC vents in the rooms, we defined the structure of this PGM as shown in Figure 5.

Nodes named S_n are temperature sensors in the rooms, $D_{m,n}$ are doors, and W_n are windows. Special node *wu* denotes the outside temperature as reported by Weather Underground. *HVAC* node tracks the state of the HVAC system. Normally, it would be split into 2 variables, one responsible for fan status and one for a heating and cooling module. Since the experiment was performed during the fall, the fan was never used, and the HVAC was used only in heating mode. Hence, we modelled the HVAC state as a single boolean variable indicating whenever the heater was on or off.

The arrows here represent directed dependencies. For example, a room temperature is influenced by the HVAC state but not the other way around. Undirected edges show mutual dependencies. Our graph contains both directed and undirected edges, which

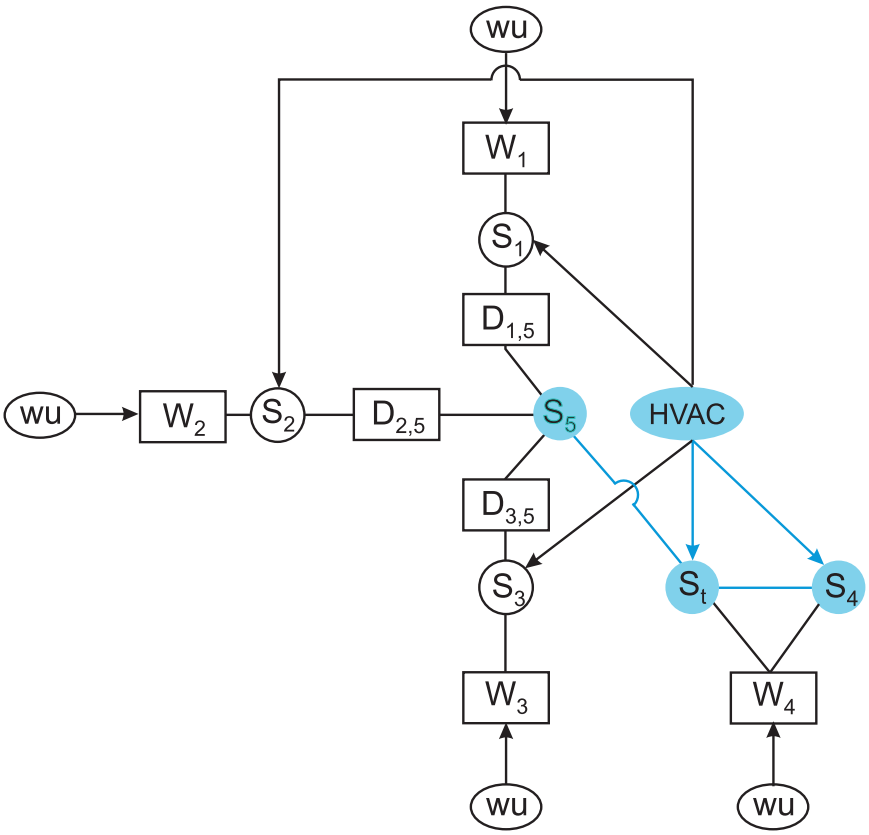

Fig. 5: Full house PGM

means it represents a *partially directed* model. It should also be noted that the graph contains cycles. The *Markov Random Field (MRF)* appears to be the most suitable model for this problem. Since MRF is an undirected model, we will need to treat all edges as undirected.

The second decision we need to make is whether we will use discrete, continuous, or both types of variables in our model. This decision is related to how we deal with the time dimension. The temperature change over time could be treated as a time series, allowing us to apply all relevant time series analysis techniques. However we choose to treat each sample independently from the previous ones. We resample at a reasonably long time interval (1 minute) to give enough time for the temperature to settle after the state changes. Given the rate of temperature change and the sensor noise, we conclude that it should be sufficient to work with temperatures rounded to the whole °C and to treat temperature values as discrete variables.

Before considering the more complex problem of latent variable parameter estima-

tion (doors and windows), we first try to learn the parameters of a subset of the network which does not contain latent variables. This subset is marked in blue in Figure 5 also shown separately in Figure 6.

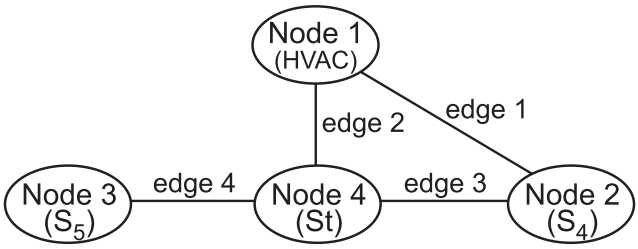

Fig. 6: PGM without latent variables

Our network has 4 nodes and 3 edges. Based on discretized data from our dataset, the variables of this network have 2, 11, 7, and 11 states respectively.

This pairwise undirected graphical model could be parametrized by a set of *potential functions*. We will distinguish *node potential functions*: $\phi_i(x_i)$ (one per node) and *edge potential functions*: $\phi_e(x_{e_j}, x_{e_k})$ (one per edge). The joint probability of an assignment of variables $x_1, x_2, \ldots, x_N$ could be expressed as:

$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \prod_{i=1}^{N} \phi_i(x_i) \prod_{e=1}^{E} \phi_e(x_{e_j}, x_{e_k})$$

The *range* of the potential functions is $\mathbb{R}^+$. The normalization constant Z (historically called *partition function*) is to ensure that this is a proper probability distribution which sums to 1:

$$Z = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_N} \prod_{i=1}^{N} \phi_i(x_i) \prod_{e=1}^{E} \phi_e(x_{e_j}, x_{e_k})$$

A *factor graph*, corresponding to this parametrization is shown in Figure 7.

3.3 Parameter Estimation

The maximum likelihood estimate of model parameters Θ using m data samples could be implemented as a minimization of the negative log-likelihood function:

$$-log(x_1, x_2, \ldots, x_N|\Theta) = -\sum_{m} \sum_{i=1}^{N} \left(log(e^{b_{i,x_i}}) + \sum_{e=1}^{E} log(e^{w_{x_{e_j}, x_{e_k}, e}}) \right) + mlog(Z) \quad (1)$$

where each node i has a bias $b_{i,s}$ per state s, and each edge e has weight $w_{s1,s2,e}$ per state combination $s1, s2$. The parameter vector Θ is a concatenation of b and w.

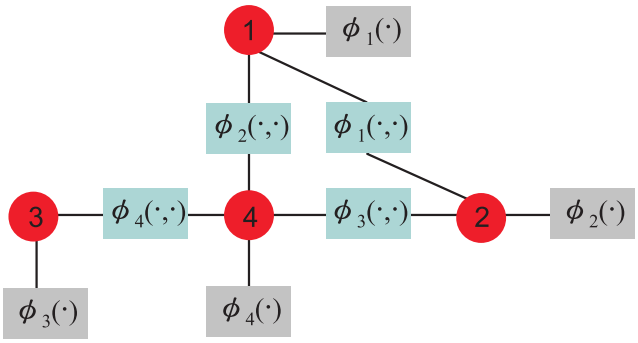

Fig. 7: PGM factor graph

Under this parameterization, the function (1) is convex in b and w [10]. As such, it could be minimized using standard unconstrained optimization methods.

To estimate the parameters of our network, we use Matlab *UGM Toolkit* [9]. Because the model is relatively small, we can use exact inference. To minimize (1), we use the *minFunc* [8] package. The minimization process converges after approximately 300 iterations.

Resulting node (b) and edge (w) potentials:

Node 1:

$5.96 \cdot 10^{5} \quad 1.68 \cdot 10^{-6}$

Node 2:

$1.21 \quad 16.1 \quad 275.0 \quad 79.5 \quad 7.46 \quad 0.00225 \quad 0.0274 \quad 0.0726 \quad 1.22 \quad 0.721 \quad 0.0799$

Node 3:

$0.00149 \quad 2.0 \quad 128.0 \quad 984.0 \quad 1211.0 \quad 2.06 \quad 1.07 \cdot 10^{-6}$

Node 4:

$7.3 \cdot 10^{-4} \quad 112.0 \quad 68.3 \quad 965.0 \quad 0.0503 \quad 0.0983 \quad 323.0 \quad 0.21 \quad 9.12 \quad 0.842 \quad 7.2 \cdot 10^{-5}$

Edge 1:

$$\begin{pmatrix} 3.42 \cdot 10^{-10} & 0.233 & 1.58 & 1.56 & 0.708 & 1.59 \cdot 10^{4} & 304.0 & 289.0 & 269.0 & 546.0 & 20.8 \\ 3.53 \cdot 10^{9} & 69.2 & 174.0 & 51.0 & 10.5 & 1.42 \cdot 10^{-7} & 8.99 \cdot 10^{-5} & 2.51 \cdot 10^{-4} & 0.00454 & 0.00132 & 0.00384 \end{pmatrix}$$

Edge 2:

$$\begin{pmatrix} 1.18 \cdot 10^{-6} & 4.58 \cdot 10^{-4} & 2.31 \cdot 10^{-4} & 0.00188 & 4.17 \cdot 10^{4} & 1.92 \cdot 10^{4} & 2.84 \cdot 10^{6} & 489.0 & 1711.0 & 57.3 & 0.0232 \\ 617.0 & 2.44 \cdot 10^{5} & 2.96 \cdot 10^{5} & 5.13 \cdot 10^{5} & 1.21 \cdot 10^{-6} & 5.13 \cdot 10^{-6} & 1.14 \cdot 10^{-4} & 4.3 \cdot 10^{-4} & 0.00534 & 0.0147 & 0.0031 \end{pmatrix}$$

Edge 3:

$$\begin{pmatrix} 0.0689 & 5.96 \cdot 10^{7} & 6911.0 & 1.94 \cdot 10^{-4} & 0.0129 & 0.00938 & 0.096 & 0.213 & 0.411 & 0.462 & 0.466 \\ 1.08 \cdot 10^{17} & 7.01 \cdot 10^{14} & 5.42 \cdot 10^{11} & 2.63 \cdot 10^{13} & 1.41 \cdot 10^{-11} & 1.55 \cdot 10^{-11} & 5.24 \cdot 10^{-11} & 2.25 \cdot 10^{-8} & 1.22 \cdot 10^{-6} & 4.98 \cdot 10^{-6} & 9.51 \cdot 10^{-6} \\ 3.0 \cdot 10^{14} & 4.1 \cdot 10^{13} & 2.25 \cdot 10^{10} & 1.62 \cdot 10^{12} & 1.35 \cdot 10^{9} & 5.35 \cdot 10^{-14} & 8.38 \cdot 10^{-13} & 2.01 \cdot 10^{-10} & 1.23 \cdot 10^{-8} & 3.65 \cdot 10^{-8} & 1.12 \cdot 10^{-7} \\ 3.77 \cdot 10^{-6} & 1.59 \cdot 10^{13} & 2.78 \cdot 10^{11} & 1.07 \cdot 10^{13} & 2.72 \cdot 10^{10} & 2.49 \cdot 10^{9} & 4.9 \cdot 10^{-14} & 1.29 \cdot 10^{-11} & 7.09 \cdot 10^{-10} & 1.64 \cdot 10^{-9} & 9.01 \cdot 10^{-9} \\ 1.32 \cdot 10^{-6} & 4.54 \cdot 10^{-10} & 2.03 \cdot 10^{12} & 4.12 \cdot 10^{14} & 1.67 \cdot 10^{12} & 8.54 \cdot 10^{11} & 2.43 \cdot 10^{6} & 3.1 \cdot 10^{-13} & 7.47 \cdot 10^{-11} & 3.17 \cdot 10^{-10} & 5.81 \cdot 10^{-10} \\ 2.35 \cdot 10^{-6} & 8.84 \cdot 10^{-10} & 5.1 \cdot 10^{-12} & 9.84 \cdot 10^{12} & 3.76 \cdot 10^{11} & 2.17 \cdot 10^{11} & 2.24 \cdot 10^{6} & 5.08 \cdot 10^{8} & 1.79 \cdot 10^{-10} & 9.19 \cdot 10^{-10} & 1.43 \cdot 10^{-9} \\ 2.51 \cdot 10^{-5} & 1.47 \cdot 10^{-8} & 9.97 \cdot 10^{-11} & 1.92 \cdot 10^{-17} & 2.23 \cdot 10^{11} & 4.74 \cdot 10^{11} & 1.89 \cdot 10^{7} & 2.58 \cdot 10^{10} & 2.61 \cdot 10^{12} & 1.29 \cdot 10^{-8} & 2.26 \cdot 10^{-8} \\ 1.89 \cdot 10^{-4} & 1.89 \cdot 10^{-7} & 3.19 \cdot 10^{-9} & 2.3 \cdot 10^{-15} & 4.0 \cdot 10^{-14} & 3.34 \cdot 10^{10} & 2.85 \cdot 10^{6} & 1.7 \cdot 10^{10} & 6.39 \cdot 10^{12} & 1.46 \cdot 10^{12} & 4.61 \cdot 10^{-7} \\ 2.11 \cdot 10^{-4} & 5.52 \cdot 10^{-7} & 4.74 \cdot 10^{-8} & 7.55 \cdot 10^{-11} & 1.13 \cdot 10^{-11} & 1.02 \cdot 10^{-11} & 2.43 \cdot 10^{4} & 2.54 \cdot 10^{8} & 6.62 \cdot 10^{10} & 6.54 \cdot 10^{10} & 9.46 \cdot 10^{14} \\ 6.04 \cdot 10^{-4} & 2.13 \cdot 10^{-6} & 8.86 \cdot 10^{-8} & 4.25 \cdot 10^{-12} & 4.03 \cdot 10^{-12} & 7.85 \cdot 10^{-12} & 2277.0 & 5.11 \cdot 10^{7} & 2.58 \cdot 10^{11} & 4.77 \cdot 10^{11} & 3.31 \cdot 10^{15} \\ 0.0464 & 0.00314 & 2.11 \cdot 10^{-4} & 1.9 \cdot 10^{-7} & 2.95 \cdot 10^{-8} & 2.16 \cdot 10^{-8} & 9.74 \cdot 10^{4} & 1.89 \cdot 10^{-5} & 5.44 \cdot 10^{-4} & 3.59 \cdot 10^{13} & 5.95 \cdot 10^{17} \end{pmatrix}$$

Edge 4:

$$\begin{pmatrix} 8.48\cdot10^{18} & 7.11\cdot10^{10} & 0.00125 & 2.71\cdot10^{-6} & 2.19\cdot10^{-6} & 7.51\cdot10^{-7} & 3.81\cdot10^{-4} & 0.00134 & 6.41\cdot10^{-4} & 0.0273 & 0.0498 \\ 4.64\cdot10^{15} & 3.05\cdot10^{10} & 4.1\cdot10^{13} & 2.87\cdot10^{9} & 2.18\cdot10^{7} & 6.06\cdot10^{-15} & 2.96\cdot10^{-11} & 4.23\cdot10^{-10} & 3.11\cdot10^{-10} & 2.34\cdot10^{-7} & 1.0\cdot10^{-6} \\ 3.24\cdot10^{-7} & 2.24\cdot10^{7} & 5.62\cdot10^{11} & 3.59\cdot10^{8} & 1.82\cdot10^{7} & 1.87\cdot10^{7} & 7900.0 & 3.23\cdot10^{-12} & 4.44\cdot10^{-11} & 9.42\cdot10^{-9} & 2.4\cdot10^{-8} \\ 1.82\cdot10^{-12} & 8.09\cdot10^{-12} & 2.26\cdot10^{10} & 6.52\cdot10^{7} & 8.07\cdot10^{6} & 1.64\cdot10^{7} & 6.04\cdot10^{5} & 5.34\cdot10^{8} & 155.0 & 1.49\cdot10^{-12} & 4.58\cdot10^{-12} \\ 3.21\cdot10^{-9} & 2.56\cdot10^{-8} & 2.75\cdot10^{-13} & 2.51\cdot10^{5} & 5.81\cdot10^{5} & 2.72\cdot10^{6} & 2.94\cdot10^{5} & 1.27\cdot10^{9} & 5999.0 & 2.52\cdot10^{4} & 2.42\cdot10^{-9} \\ 1.08\cdot10^{-8} & 2.65\cdot10^{-6} & 1.25\cdot10^{-12} & 1.41\cdot10^{-14} & 8.82\cdot10^{-15} & 2.07\cdot10^{8} & 1.97\cdot10^{7} & 1.07\cdot10^{11} & 2.26\cdot10^{6} & 4.78\cdot10^{8} & 9.85\cdot10^{11} \\ 9.08\cdot10^{-4} & 0.0042 & 3.07\cdot10^{-7} & 1.5\cdot10^{-9} & 1.4\cdot10^{-9} & 1.25\cdot10^{-10} & 1.04\cdot10^{-6} & 1.59\cdot10^{-6} & 4.92\cdot10^{11} & 7.78\cdot10^{14} & 5.52\cdot10^{18} \end{pmatrix}$$

It instructive to view the edge potentials as a heat map (in log scale) as shown in Figures 8 and 9. As expected, we can see the higher factor potential values for similar temperatures decreasing away from the diagonal.

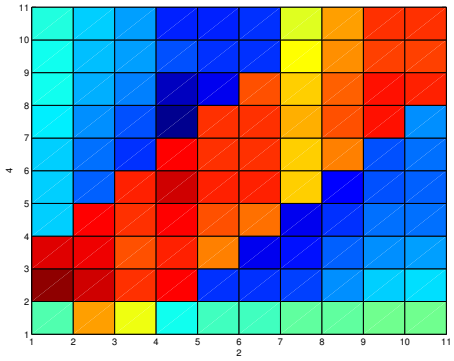

Fig. 8: Edge potentials between nodes 2 and 4

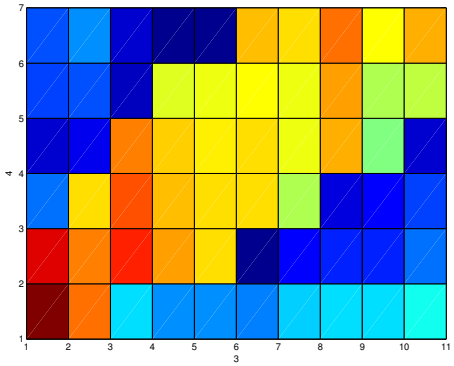

Fig. 9: Edge potentials between nodes 3 and 4

4 Results

Now, using our model of the probability distribution, we can try to answer some queries. For example, we can find the *optimal decoding* using simple *maximum a posteriori probability (MAP) estimate* query over all variables. In our case, the values corresponding to the most likely states are:

Heater: off ($HVAC = 0$)

Temperature in the living room: 18°C ($S_4 = 18$)

Temperature near the thermostat': 17°C ($S_t = 17$)

Temperature in the corridor: 19°C ($S_5 = 19$)

More interesting are the marginal MAP queries, where we fix values of some variables and try to predict the most likely assignment of the remaining ones. For example, knowing whether the heater is on or off and knowing the temperature in one room, we can try to predict the most likely temperature in another room.

Plots shown in Figures 10 and 11 show the results of such queries. The x axis is the measured temperature in one room, and the y axis is the estimated temperature in another room. The red line shows temperature estimates when the heater is running, and the green line shows temperature estimates when the heater is idle.

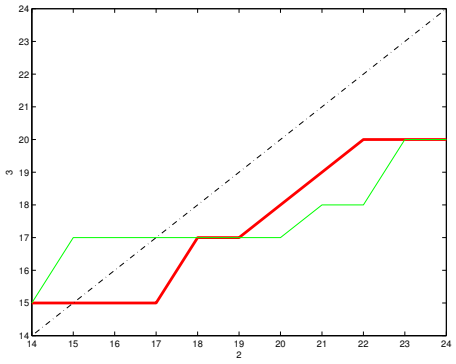

Fig. 10: Marginal MAP query results for $S_t|S_4$

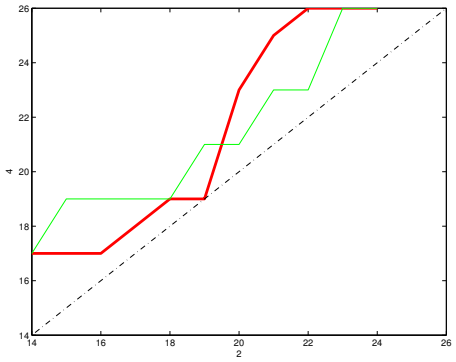

Fig. 11: Marginal MAP query results for $S_5|S_4$

We can observe an interesting inversion of the HVAC effect on both plots. Until a certain point (around 18°C in Figure 10 and 20°C in Figure 11), the temperature estimate with HVAC heating is lower than without it. Above this point, the effect is reversed, and estimates with the HVAC heating are higher than without it.

5 Conclusions

We have been able to design and build a small home sensor network and collect some useful data. This dataset will be made public and available for future experiments.

We successfully modelled temperatures in a few rooms as Markov Random Fields, and estimated the MRF parameters from the collected data.

We have not been able to model the temperatures in all rooms or state of the doors and windows as yet. The main reason is that during the initial planning of this project, we made an assumption that we would be able to do this using only *unsupervised learning* techniques. Thus we have not collected information about the actual state of doors or windows during the experiment. There were also some practical considerations for this decision. Collecting door and window states would require us to design and build additional sensors, which would significantly expand the workload for this project.

6 Future Directions

With our sensor network running, we will keep collecting additional data. As seasons change, we will get more diverse data. For example, we will finally get some data samples in which the HVAC is engaged in cooling mode.

The most interesting future direction would be to try to predict the state of doors and windows. This would require additional data collection with door and window sensors. Once such data is collected, we can use, for example, *Conditional Random Fields (CRF)* to model the conditional probability of doors and windows given temperature sensor values.

It would be interesting to apply the methods we used in this project to different datasets, such as the one from CMU *Sensor Andrew* project [7].

Finally, we could try to use this model in some control algorithms. For example, we can control the setpoint of the thermostat to achieve a desired temperature in a particular room of the house.

References

[1] DIGI INTERNATIONAL INC. XBee®/XBee-PRO®RF Modules. `https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf`.

[2] FALUDI, R. TMP36 instructions: Simple sensor network. `http://www.faludi.com/bwsn/tmp36-instructions-simple-sensor-network/`.

[3] FALUDI, R. *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. O'Reilly Media, Incorporated, 2010.

[4] KOLLER, D., AND FRIEDMAN, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[5] KURLAND, V. Wireless temperature sensor. `https://github.com/vkurland/xbee_temp_sensor`.

[6] RADIO THERMOSTAT COMPANY OF AMERICA. Wi-Fi USNAP module API version 1.3. `http://www.radiothermostat.com/documents/RTCOAWiFIAPIV1_3.pdf`.

[7] ROWE, A., BERGES, M. E., BHATIA, G., GOLDMAN, E., RAJKUMAR, R., GARRETT, J. H., MOURA, J. M., AND SOIBELMAN, L. Sensor andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development 55*, 1.2 (2011), 6–1.

[8] SCHMIDT, M. minFunc: Matlab uunconstrained optimization using line-search methods. `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`, Apr. 2013.

[9] SCHMIDT, M. UGM: Matlab code for undirected graphical models. `http://www.di.ens.fr/~mschmidt/Software/UGM.html`, Apr. 2013.

[10] SCHMIDT, M., AND MURPHY, K. Modeling discrete interventional data using directed cyclic graphical models. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (2009), AUAI Press, pp. 487–495.