# A Learning Agent for Heat-Pump Thermostat Control

Daniel Urieli
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
urieli@cs.utexas.edu

Peter Stone
Dept. of Computer Science
The University of Texas at Austin
Austin, TX 78712 USA
pstone@cs.utexas.edu

## ABSTRACT

Heating, Ventilation and Air Conditioning (HVAC) systems are one of the biggest energy consumers around the world. With the efforts of moving to sustainable energy consumption, heat-pump based HVAC systems have gained popularity due to their high efficiency and due to the fact that they are powered by electricity rather than by gas or oil. One drawback of heat-pump systems is that their efficiency sharply decreases when the outdoor temperature is around or below freezing. Therefore, they are backed up by an auxiliary heating system that is effective in cold whether, but that consumes twice as much energy. A popular way of saving energy in HVAC systems is *setting back* the thermostat, meaning, relaxing the heating/cooling requirements when occupants are not at home. While this practice leads to significant energy savings in many systems, it could in fact increase the energy consumption in a heat-pump based system, using existing control strategies, as it forces an excessive usage of the auxiliary heater. In this paper, we design and implement a complete, adaptive reinforcement learning agent which applies a new control strategy for a heat-pump thermostat. For our experiments, we use a complex, realistic simulator that was developed for the US Department of Energy. Results show that the learned control strategy (1) leads to roughly **7.0%-14.5%** energy savings in typical homes in the New York City, Boston, and Chicago areas; while (2) keeping the occupants' comfort level unchanged when compared to an existing strategy that is deployed in practice.

## Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory, Graph and tree search strategies*

## Keywords

Machine Learning; Control; Energy Savings

## 1. INTRODUCTION

According the United State's Department Of Energy (US DOE), 40% of the energy consumed in the United States is consumed by residential (22%) and commercial (18%) buildings.[1] Furthermore, heating, Ventilation and Air Conditioning (HVAC) systems are responsible for more than 50% of the energy consumed by buildings. According to [3, 7], with the efforts of moving to a sustainable energy consumption, our homes and offices will have to be heated by efficient ground and air source heat pumps powered by electricity rather than existing natural gas and oil fired boilers. In HVAC systems, the thermostat's operation significantly affects energy consumption [4]. Current interest in more efficient thermostat operation is evident from the appearance of recent start-up companies such as Nest[2], and BuildingIQ[3].

This paper focuses on using advanced reinforcement learning (RL) methods to design an adaptive, energy efficient thermostat operation strategy for a heat-pump system. A common energy-saving practice in HVAC systems, termed *setting back* the thermostat, is relaxing the heating/cooling requirements for parts of the day. However, when applied to heat-pump systems with existing thermostat strategies, this practice can in fact increase energy consumption, since recovering the temperature back frequently results in excessive use of an energy expensive, electric-resistance auxiliary heater.[4] One thing that complicates controlling of a heat-pump system is that its heat-energy output significantly varies based on environmental conditions; Since a heat-pump *moves* heat-energy from one place to another, rather then *converts* other energy sources (gas, oil, electricity) into heat, its heat output, or the rate it moves the heat-energy highly depends, for instance, on the temperatures along the heat-energy path. This causes heating or cooling to have a *drifting* rather than approximately-constant effects. In addition, action effects are *noisy* due to environmental factors that are often not easy to measure, for instance the temperature of the house's walls and furniture,

**Appears in:** *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013), Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.*

[1] `http://buildingsdatabook.eren.doe.gov/`
[2] `http://www.nest.com`
[3] `http://buildingiq.com/`
[4] In fact, in the day of writing of this paper, the official website of the US Department of Energy recommends to "avoid setting back heat pump's thermostat manually if it causes the electric-resistance heating to come on". Furthermore, it says that "Programmable thermostats are generally not recommended for heat pumps" since they use setback strategies. It also mentions that some companies have recently started to develop algorithms for setting back while minimizing the usage of the auxiliary heater. See `http://energy.gov/energysaver/articles/tips-heat-pumps` and `http://energy.gov/energysaver/articles/thermostats-and-control-systems`

which serve as heat-capacitors. The presence of such heat-capacitors results in actions that have *delayed* effects. Two additional limitations are related to the class of applicable RL methods for this problem. First, any realistic thermostat control strategy cannot use too aggressive or too long exploration. Second, while RL methods typically optimize the *expected* long-term cost, customer acceptance of any control strategy will probably depend on its *worst-case* behavior.

The main contribution of this paper is designing a complete reinforcement learning agent that learns and applies a new adaptive control strategy for a heat-pump thermostat that (1) leads to roughly **7.0%-14.5%** yearly energy savings in a realistic simulation of different house sizes and weather conditions, while (2) keeping the occupants' comfort level unchanged when compared to an existing strategy that is deployed in practice. Experiments are run using a complex, realistic simulator written for the US DOE. Our strategy simultaneously solves two related, but slightly different problems of heating in the winter and cooling in the summer. Our agent is realistically deployed in a simulated, unknown in advance house, and after 3 days of exploration (during which occupants could be traveling out of home) starts to save energy. Our agent makes decisions in real-time, and keeps learning and improving performance while acting, as it gathers more data. As a side contribution, we extend the use of reinforcement learning to a new relevant domain, that is of current interest.

Technically, in order to apply RL to thermostat control, we carefully define the problem as a continuous state Markov Decision process (MDP). After randomly exploring the effects of its actions during the first 3 days, our agent uses a regression learning algorithm to fit a transition function that models the house in which the agent operates. Using information like weather forecast and history of past-measurements results in a high-dimensional MDP state, and therefore it is impractical to plan, or compute a value function, over the whole state space. Therefore, our agent uses an efficient online lookahead policy, based on a constrained, specialized tree-search.

We note that while the above companies address problems that are closely related to ours, to the best of our knowledge, the details of their algorithms are kept confidential, and therefore unpublished. We are not aware of any published studies that use the methods we use, but there are studies that address related problems. RL was used in an HVAC system [1, 2], but to control the inside operation of the system rather than the higher level actions of heating or cooling that we control here. Adaptive thermostat controller was designed at [8] but used mixed-integer programming instead of RL, in a different simulation environment with a different goal of minimizing the peak demand over the grid.

## 2. BACKGROUND AND PRELIMINARIES

In this section, we start with describing the simulation environment and the default thermostat strategy that is used in realistic deployments, and continue with defining the problem setup and formalizing it as a Markov Decision Process.

### 2.1 Simulation Environment

Since real-world experiments would both be costly and take too much time, in order to achieve our goal, we rely on a complex, realistic HVAC simulation. Specifically, we use GridLAB-D [5], which is an open-source, smart-grid simulator that was developed for the US DOE. Importantly for our purposes, it has a residential building model, that includes heat gains and losses and the effects of thermal mass, as a function of weather (temperature and solar radiation), occupant behavior (thermostat settings and internal heat gains from appliances), and heating/cooling system efficiencies. It models parallel heat flow paths through the envelope of the building (walls, windows, doors, ceilings, floors, and infiltration air flows), considers the mass of the air in the interior volume of the house. It uses meteorological data collected in hundreds of cities across the U.S. by the National Renewable Energy Laboratory[6], recorded in a standard TMY2 (Typical Meteorological Year) file format.

Our simulation uses a heat-pump based HVAC system. At its peak performance, a heat-pump can output heat energy that is 4× higher than the energy it consumes. However, when outdoor temperatures are near or below freezing, its efficiency sharply decreases; therefore it is backed up by an auxiliary heater, in our case a resistive heat coil. On one hand, the auxiliary heater's efficiency is almost unaffected by the outdoor temperature, but on the other hand it consumes about twice the energy consumed by the heat-pump heater. A resistive heat coil is a popular backup system, partly due to the expected entrance of renewable electricity sources to the market. A heat-pump is also used for cooling, and is not backed up by an auxiliary cooler.

Coming from an AI perspective, our focus is on the decision making module of the house's HVAC system, namely the *thermostat.* A widely deployed default thermostat strategy is defined in Algorithm 1, and is illustrated in Figure 1. While this default thermostat strategy is simple and intuitive, from the perspective of energy consumption it has some drawbacks, as seen next.

**Algorithm 1** *DefaultThermostat(upperBound, lowerBound)*

```
1: $buffer \leftarrow 1$, $auxBuffer \leftarrow 2$, $T_{in} \leftarrow$ indoor temperature
2: if $T_{in} > upperBound + buffer$ then
3:     COOL until $T_{in} < upperBound - buffer$
4: else if $T_{in} < lowerBound - (buffer + auxBuffer)$ then
5:     AUX-HEAT until $T_{in} > lowerBound + buffer$
6: else if $T_{in} < lowerBound - buffer$ then
7:     HEAT-PUMP-HEAT until $T_{in} > lowerBound + buffer$
```

### 2.2 Problem Setup and Challenge

In our setup, we simulate a single-family residential home, and assume that occupants are at home between 6pm and 7am of the next day, and that the house is empty between 7am and 6pm (we call it a *don't-care* period). Our goal, illustrated in Figure 2, is to minimize the total energy consumed by the HVAC system, while (1) keeping a desired temperature range of 69-75$^{\circ}F$ whenever the occupants are at home, and (2) being indifferent to the home temperature during the don't-care period.

Under this setup, a straightforward setback strategy would be to turn the system off during the don't-care period, and turn it back on once occupants are at home. However, such setback of a thermostat that uses Algorithm 1 can in fact increase consumption by more than 7% comparing to just

[5]`http://www.gridlabd.org`

[6]`http://www.nrel.gov`

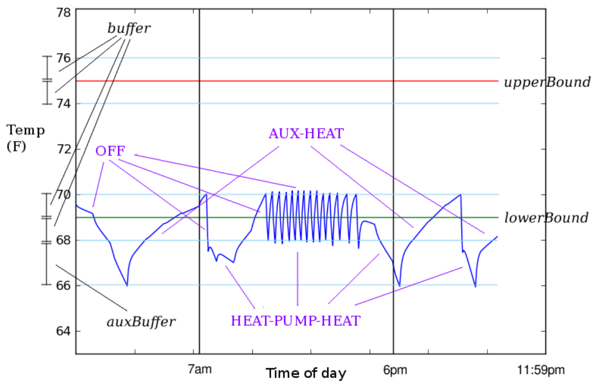

**Figure 1: Indoor Temperature controlled by the default thermostat, over a 24-hour period of a winter day. Quantities of algorithm 1 are illustrated here.**

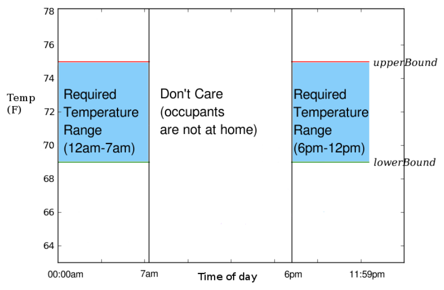

**Figure 2: Temperature Requirements Specification.**

leaving it always on. The main reason for this is that at the end of the don't-care period, the temperature is often significantly out of range, in which case Algorithm 1 forces extended use of the energy-expensive auxiliary heating unit. More regular use of the heat pump throughout the don't-care period ends up consuming less energy, as happens when leaving the thermostat always on. An additional problem with such setback is that requirement (1) is frequently violated, since it might take up to several hours for the HVAC system to bring the temperature back to the desired range.

However, setting back the temperature is still desirable for saving energy, as long as it does not cause unnecessary use of auxiliary heating. This is due to the fact that setting-back gets the indoor temperature closer to the outdoor temperature, which in turn slows the heat dissipation, so that less energy is needed to compensate for heat energy losses. Therefore, an ideal strategy strategy would be able to predict whether it is possible to set back the thermostat for some time, then start heating enough in advance using the heat-pump whenever possible and auxiliary heating when unavoidable, so as to reach the desired temperature by the time the occupants are back, thus allowing the temperature setback to effectively save energy, while leaving the occupants' comfort unchanged. In this paper, we define and test such a strategy.

Figure 3 illustrates a challenge in designing such a strategy. For a *given day*, let the *heating slope* be the (not necessarily straight) line consisting of all $(x, y)$ points, such that if $x$ is a time of day in the don't-care period and $y$ is the indoor temperature at time $x$ then turning on the heat-pump at time $x$ would bring the indoor temperature back into the desired range, exactly at 6pm. We define the *auxiliary slope* similarly for the auxiliary heater. Note that these

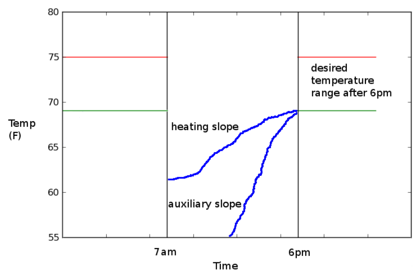

**Figure 3: The heating slope and the auxiliary slope**

slopes are only hypothetical, and cannot exactly be computed in advance, as doing so requires a complete knowledge of the world-states throughout the uncertain future. During a don't care period, as long as a temperature $y$ at time $x$ is above the heating-slope (resp. auxiliary-slope), an agent can reach the desired temperature at 6pm using only the heat-pump (resp. auxiliary heater). A good strategy should handle the trade off of trying to setback the temperatures to as close as possible to the (unknown) heating slope, while keeping a safe distance above it, to avoid the need to use the auxiliary heater in the face of possible outdoor temperature drops. More generally, depending on the specific house properties and weather conditions there exists some path through the time-temperature space (such as the one in Figure 3) that would lead to minimal energy consumption. Our challenge is to design a control strategy that would be able to approximate this path for each house our agent is deployed in, and for any weather conditions.

Note that to this point, we have focused on winter, which is more complicated than summer due to the two different heating actions. In fact, our strategy works in the summer as well, where there is only one cooling action (so no need to avoid a more expensive action), but where there is still the challenge of setting back the thermostat to save energy, and start cooling in advance to bring the temperature back to range on time. In our experiments, we run tests throughout the year, thus testing both conditions simultaneously.

We assume that the default thermostat strategy is used to keep the temperature in range whenever occupants are at home, in order to keep a similar comfort level across all tested strategies (so that only energy usage differs), and due to the lower potential for energy savings at these times. Therefore, we only consider changing the thermostat strategy during the don't-care period.

## 2.3 MDP Model

A thermostat works in the real-time cycle of sensing the world state, for instance the temperature and the time of day; running some computations; and acting by choosing one of four actions: cooling, off, heat-pump heating, or auxiliary heating. A strategy's goal is to minimize a cost function, which is the total energy it uses over some period, while satisfying a desired comfort level. Formally, this problem can be represented as a Markov Decision Process (MDP) [6]. An (episodic) MDP is a tuple $(S, A, P, R, T)$, where $S$ is the set of states; $A$ is a set of actions; $P : S \times A \times S \rightarrow [0, 1]$ is a state transition probability function where $P(s, a, s')$ denotes the probability of transitioning to state $s'$ when taking action $a$ from state $s$; $R : S \rightarrow \mathbb{R}$ is a state-based reward function; and $T \in S$ is a set of terminal states, where entering one of which terminates an *episode*. Our MDP is defined as follows.

- **S**: $\{\langle T_{in}, T_{out}, Time, e_a, prevAction, t_0, \ldots, t_9, weatherForecast\rangle\}$. Here, $T_{in}$ and $T_{out}$ are the indoor and outdoor air temperatures, respectively; $Time$ is the time of day; $e_a$ is the energy consumed by last action; $prevAction$ is the previously taken action; $t_0, \ldots, t_9$ is a history of the last 10 indoor temperatures, and $weatherForecast$ is a noisy weather forecast from the current step until the end of an episode.

- **A**: $\{\text{COOL}, \text{OFF}, \text{HEAT}, \text{AUX}\}$. Namely, there are four possible actions for cooling, off, (heat-pump-)heating and auxiliary heating, respectively.

- **P**: a complex, initially unknown, transition model given by the GridLAB-D simulator, based on the house properties, and the environmental conditions.

- **R**: $-e_a - c_{6pm}$. Here, $c_{6pm} = 100000 \times \Delta^2_{temp}$ is large quadratic cost applied when missing the temperature spec at 6pm by $\Delta_{temp}$, to help enforcing the comfort constraint. The energy consumption proportion is roughly 1:0:2:4 for COOL:OFF:HEAT:AUX respectively, but is specific to each house/weather condition and is unknown in advance.

- **T**: $\{s \in S | s.time == \text{23:59pm}\}$

We discuss our choice of state representation in detail in Section 3. In our MDP, an action is taken every 6 minutes, as the simulator models a realistic lockout of the system, such that every control action is applied for at least 6 minutes. In the context of MDPs, the goal of RL is to learn an optimal *policy*, when the *model* (namely $P$ and/or $R$) is initially unknown. A *policy* is a mapping $\pi : S \rightarrow A$ from states to actions, and an optimal policy is defined as one that maximizes the long-term rewards, or equivalently minimizes the long-term costs, from every state.

## 3. AGENT COMPONENTS

When the agent is deployed in a new house, in order to perform robustly it needs to learn the characteristics of the specific house and heating system it controls, and adapt its control strategy to these characteristics. It does so by exploring and learning the effects of its actions in the house's environment for three simulated days. During this period, the agent selects each of the four possible actions uniformly at random and records their effects. While in practice it might be possible to use more advanced exploration policy, for the purpose of this work we assume that a one-time 3-day random exploration is still a realistic setup, for instance during a weekend where occupants are traveling. Action effects are recorded in the form of $\langle s, a, s'\rangle$ tuples where $s$ is a state, $a$ is an action taken from $s$, and $s'$ is the next state transitioned into after taking action $a$ in $s$. Note that since $e_a$ is part of the state in our definition of the MDP, the reward can be computed exactly by the agent at every given state. One advantage of fully random exploration is a quick coverage of larger portions of the state space and of different action sequences, which facilitates faster learning. A disadvantage of it is an increased energy consumption, but we will see that this is outbalanced by the energy savings starting the fourth day throughout the year.

Starting the end of the third day, the agent plans and executes an energy saving set-back policy. While doing so, the agent keeps recording the effects of its actions, fitting a regression model to the accumulated action-effect tuples once at every midnight. Based on the most recently learned model, the agent keeps executing an efficient lookahead policy to choose the next action. The main routine for action selection, called at every time step with the current state observation, is summarized in Algorithm 2. In the following two sections we describe two main subroutines of this algorithm, namely the agent's model-learning algorithm (LearnHouseModel), and the agent's planning and action selection algorithm (TreeSearch). In the results section, we present an ablation analysis that tests the contribution of each of the main components of these algorithms to the final performance.

**Algorithm 2** [main routine] SelectAction*(currentState)*

```
1: dataSet.add(prevState, prevAction, currentState)
2: t ← currentState.Time
3: if  t ∈ firstThreeDays  then
4:    return randomSelect(COOL, OFF, HEAT, AUX)
5: else
6:    if t = midnight then
7:       model ← LearnHouseModel(dataSet)
8:    if t ∈ don't-care period then
9:       bestAction ← TreeSearch(model)
10:      return bestAction
11:   else
12:      return thermostatAction()
```

### 3.1 Learning the House Characteristics

The agent learns the house characteristics in a routine named LearnHouseModel. LearnHouseModel fits regression models to the collected data-set of $\langle s, a, s'\rangle$ tuples, which are samples from the house's state transition function. The agent uses these tuples as labeled examples $\langle s, a\rangle \rightarrow s'$ for fitting a regression to model the transition function, separately for each of the four actions (a total of four regression runs). A critical part in learning the transition function is selecting what features to include as the regression's independent variables. In turn, this implies the features included in our state representation, based on definition 5.4.1 from [5], used here as a main guideline:

*Definition 1.* A state variable is the minimally dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function, and the contribution (here the reward) function.

In what follows we describe the process by which we select the regression, and therefore the state, features. We start by including in our state the three features that are required for computing the reward function: $T_{in}$, $e_a$, and $Time$. For computing the transition function, we need features that help predicting $T_{in}$ and $e_a$ ($Time$ can be directly computed). Here we only present the process of selecting features for predicting $T_{in}$, but the process for selecting features for predicting $e_a$ is conceptually similar and uses a subset of the state-variables needed for predicting $T_{in}$. For predicting $T_{in}$ at the next time-step, an obvious feature that is included, besides $T_{in}$ itself, is the outdoor temperature at the current time step, $T_{out}$, as it directly affects the heat-pump operation, and is easily measurable, similarly to $T_{in}$. We start by testing a linear regression using only $T_{in}$ and $T_{out}$ for predicting $T_{in}$. Note that during regression runs we always add a constant 1 as a "bias" (regression-only) feature, to enable affine regression. To test the prediction's accuracy, we generate data by simulating one year of uniformly random actions and recording the resulting 87,600 $\langle s, a, s' \rangle$ tuples, one for each 6-minute time-step during one year. We then measure the prediction error in a cross-validation test which repeatedly chooses 70% of the data as a training set and the rest 30% of the data as a validation set and averages the results of multiple runs.[7] The cross-validation's error measure is the mean-squared prediction error over the validation-set, but here we report the related and more intuitive error measure of the standard deviation of the prediction errors, measured in $^\circ F$ (Fahrenheit).[8]

Using only $T_{in}$ and $T_{out}$ the prediction error is unacceptably high: a standard deviation of more than $1^\circ F$ for a 6 minutes time-step. This means that over 1 hour, the standard deviation of the prediction error is $10^\circ F$, which makes it hard to plan actions several hours in advance. A main source of prediction error is a hidden state of the house and the environment, for example the temperatures of the house's walls and furniture, that serve as heat capacitors, and causes actions to have *delayed* effect. While a realistic thermostat generally cannot measure this hidden part of the state, it could use observable quantities that affect or correlated with the hidden part of the state. Specifically, we add as features the previous action taken by the thermostat, and a history of 10 measured indoor-temperatures. Adding the previous action as a feature results in a 4x4=16 combinations for the recent pair of actions, and for each such combination we run a separate regression, in a total of 16, rather than 4, regressions.[9] The 10 historic temperatures are added directly as regression features. Figure 4 shows the cross-validation error achieved by incrementally adding regression features, starting from the two features of $T_{in}$ and bias, then adding $T_{out}$, adding the previous action, and then adding 10 historic temperatures one-by-one, from the most recent to the least recent, to a total of 14 features. Using all the features, the average standard deviation of the errors is less than $0.1^\circ F$ per 6-minute time-step, or less than $1^\circ F$ per hour. As each feature contributes to error reduction, the LearnHouseModel is chosen to fit a linear regression using the above 14 features.[10] Based on Definition 1, we include all these features (except the bias feature) in our state representation. Note that cross-validation test were run on a typical, 2500 square feet with weather conditions recorded at New-York City. In the results section we test the agent under a range of house sizes and different weather conditions recorded in US cities.

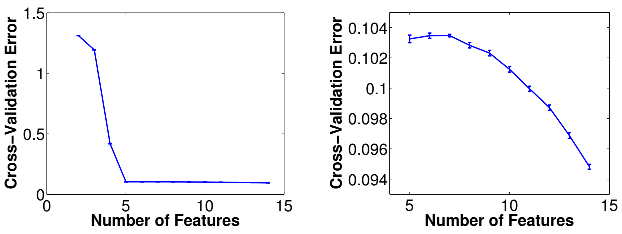

**Figure 4: Cross-validation error (standard deviation, in $^\circ F$) vs. number of features. The right figure is a zoom-in of the left figure using 5-14 features.**

Adding features to the state representation helps us in predicting $T_{in}$ as a part of the transition function. But, being part of the state, these state features now need to be *predicted* as a part of the transition function. All but one of the added features are just forward recordings of past measurements, and can be directly computed from $\langle s, a \rangle$ without the need to predict them. The only one needs to be predicted is $T_{out}$. However, $T_{out}$ is different than $T_{in}$ in that it is independent of the agent's actions and can be considered as an *information state* [5], a term that refers to the part of the state describing random processes external to the agent. The approach we take for predicting $T_{out}$ is using a weather forecast that is assumed to be given by an external source. For instance, the agent can connect to a weather forecast agency using the internet infrastructure in a realistic deployment. As the weather forecast is needed for predicting $T_{out}$ which is already part of our state, based on Definition 1 we add the weather forecast to our state representation as (multidimensional) state feature. As the weather forecast is given from an external source, it does not need to be predicted by itself from $\langle s, a \rangle$, so no further features are needed in our state representation and the resulting state representation is the one defined in Section 2.3. For the purpose of simulation, we generate a noisy weather forecast from the actual future weather data, given in the TMY2 file, using the following rule. At a specific hour $h$, the forecast for $i$ hours into the future, denoted as $f_{h+i}$ is defined (recursively) as:

$$f_{h+i} = \begin{cases} T_{out}(h) & \text{if } i = 0 \\ f_{h+i-1} + N(0, 0.5) & \text{if } i > 0 \end{cases}$$

where $N(0, 0.5)$ is a normal random variable with $\mu = 0$ and $\sigma = 0.5$. Note that the noisy forecast is computed at every time step until the end of the day, and therefore changes as time progresses. A histogram of the resulting forecast errors over one year, summarized together for forecast ranges of 6-17 hours into the future (these are the forecasts ranges needed during the don't-care period) is shown in Figure 5.

## 3.2 Planning and Acting

Recall that our agent uses the default thermostat strategy to keep the temperature in range outside the don't-care period, whenever occupants are at home. During the don't-care period, the agent plans and selects actions using the nightly learned model, with the goal of executing an effective set-back strategy that both saves energy and minimizes violations of the temperature comfort requirements. In MDP

[7]While random action data is not a representative sample of the actual resulting behavior, it still gives a good estimation of the prediction's accuracy using a specific set of features. Indeed, when testing our model on online data resulting from the actual agent's policy predictions, accuracy was similar.

[8]Due to the large sample size, the difference between the standard deviation and the root-mean-squared error is negligible.

[9]While the previous-action could be approximately treated as an "ordinal" continuous feature that receives 4 possible values, the accuracy was better when using it as a feature that separates to different regression runs.

[10]Adding polynomial features did not significantly improve the prediction accuracy.

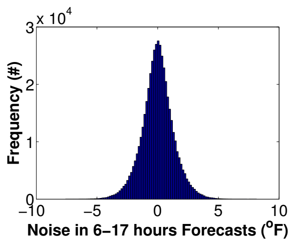

**Figure 5: Histogram of the added noise in all weather forecasts generated over a period of 1 simulated year, where forecasts predict 6-17 hours into the future. Here, the resulting range of noise values is $[-7.6, 8]$.**

terms, the agent's goal is finding a policy that maximizes the long-term reward. In general, once the approximate transition (and/or reward) functions are learned, the agent can use them to approximate the optimal policy using either one of the following three methods, or a combination of them: value-function approximation, policy function approximation, or lookahead methods [5].

Due to the dimensionality of our state representation, it might be computationally intensive, or even impractical to plan or approximate a value function over the whole state space. Assuming the agent has limited on-site computational resources, it needs an efficient way to plan its actions. Therefore, our agent uses an efficient tree-search lookahead that is limited to a specific class of policies. A lookahead search starts at some point during the don't-care period, and ends at the end of an episode, namely at midnight. As a result, the agent makes plans for time-ranges of 6-17 hours, using actions of 6-minute length. As predicted values at time $t$ are used to estimate values at time $t+1$, predictions that are further into the future accumulates uncertainty and become more noisy. Therefore, we take an approach similar to Model-Predictive Control, where our agent runs a lookahead search at a given time-step step, uses the results of the search to determine only the next action to take, then runs a new search at the next time-step, and so on.

Algorithm 3 implements this lookahead search, selecting the next action to be the first action of the most promising path. Specifically, it initializes a priority queue (step1) and retrieves the current weather forecast (step 2). Next, it iterates over every time-step $i$ starting the current time until the end of the don't-care period (step 5). The *simulate*() function (steps 7, 9, 12) uses the model and the weather forecast to simulate a specific set-back policy, which applies one action from the current time-step until time-step $i$, and another action from time-step $i$ until the end of the don't-care period. For instance, step 7 simulates applying OFF and then HEAT. Simulation continues from 6pm until the end of episode at midnight, at this point simulating the default thermostat actions. Each simulation outputs the total accumulated reward along the simulated path, and the first action taken in this path (steps 7, 9, 12) which are then inserted into the priority queue as a key-value pair, where the total reward is the key and the returned action is the value (steps 8, 10, 13). Note that the first action could be either of the two simulated actions as initially $i = m$. The first action of the path that resulted in the highest reward is then selected for execution (step 15). The intuition behind the algorithm is to maximize the set-back time while still returning the temperature back to range by 6pm, through an efficient search within a policy class that does exactly that. The reason we add step 9, in which we simulate HEAT and then AUX is to account for cold days in which the heat-pump is not able to bring the temperature to the desired range.

**Algorithm 3** TreeSearch *(model)*

```
1: $Q \leftarrow priorityQueue()$
2: $f \leftarrow currentWeatherForecast()$
3: $m \leftarrow getTimeOfDayInMinutes(now)$
4: $end \leftarrow getTimeOfDayInMinutes(6pm)$
5: for $i \leftarrow m, m+6, \ldots, end$ do
6:   if $heatingNeeded$ then
7:     $[reward, action] = simulate(\text{OFF}, i, \text{HEAT}, f, model)$
8:     $Q.add(reward, action)$
9:     $[reward, action] = simulate(\text{HEAT}, i, \text{AUX}, f, model)$
10:    $Q.add(reward, action)$
11:  else if $coolingNeeded$ then
12:    $[reward, action] = simulate(\text{OFF}, i, \text{COOL}, f, model)$
13:    $Q.add(reward, action)$
14: $[bestReward, bestAction] = Q.top()$
15: Return $bestAction$
```

An important part of the *simulate()* function is handling the uncertainty in the long-term predictions of $T_{in}$. In general, regression models predict the *expected* transition for a specific $\langle s, a\rangle$ pair. However, actual values can be higher or lower, so that relying on expected transitions can result in overly optimistic behavior that applies a strong set-back, from which the heat-pump is eventually not able to recover the temperature back to range by 6pm, thus violating the comfort requirements. To hedge against that, we augment each prediction with a dynamic safety buffer that encourages risk-taking in safer situations and discourages risk-taking in less safe situations. Specifically, we augment each prediction as follows. Let $\sigma$ be the standard deviation of the regression model measured on the training-set. Let ${T_{in}}'$ be the expected temperature predicted by the regression model. Let $\Delta_{temp}$ be the difference between the current temperature and the required temperature range at 6pm, and let $\Delta_{time}$ be the number of minutes until 6pm. Then *simulate()* uses an augmented prediction $p$ defined as:

$$p = \begin{cases} {T_{in}}' - c \cdot \frac{\Delta_{temp}}{\Delta_{time}} \cdot \sigma & \text{if } currentTemperature < 69 \\ {T_{in}}' + c \cdot \frac{\Delta_{temp}}{\Delta_{time}} \cdot \sigma & \text{if } currentTemperature > 75 \end{cases}$$

where $c$ is a constant and where $\Delta_{temp}$ and $\Delta_{time}$ are normalized by dividing $\Delta_{temp}$ by 15 ($^\circ F$) and $\Delta_{time}$ by $11 \cdot 60 = 660$ (minutes), and trimming their quotient to a $[0, 1]$ range. The constant $c$ determines the maximum number of standard deviations that could possibly augment a prediction, and was determined to be 1, by running a grid search to find the best performing parameter, over a 2500 $ft^2$ house using NYC weather data. The importance of using this dynamic safety buffer is demonstrated in the ablation analysis in Section 4.2.

## 4. RESULTS

We start with testing the agent's performance in a range of different house-sizes and weathers, and continue with ablation analysis, analyzing the contributions of the different agent components to the overall performance.

## 4.1 Agent's Performance in Different Houses and Weathers

To test the agent's performance we use GridLAB-D to simulate different homes at different weather conditions over a period of 1-year, where the the heat-pump system is controlled by our agent. More specifically, we simulate 21 typical residential homes, of sizes ranging from 1000 square feet ($ft^2$) to 4000 $ft^2$. We simulate these homes under different weather conditions using typical weather data that was recorded in different cities in the United States by the US National Renewable Energy Laboratory, given in a TMY2 format. The comfort requirements are as described earlier, requiring an indoor temperature of 69-75$^\circ F$ from 6pm to 7am, with a don't-care period of 7am-6pm. We compare our learned strategies with the default thermostat strategy (Algorithm 1) that is used in real deployments, where the thermostat is always on (recall that setting the thermostat back during the don't-care period when using the default strategy actually increases energy consumption).

Figure 6 shows the energy saved by our agent with respect to the default thermostat strategy, both as a percentage, and in actual savings. We compare the energy savings both when including the exploration period (days 1-365) and when excluding it (days 4-365). Excluding it is reasonable to do given that the increased consumption during days 1-3 is a one-time cost, while the savings starting day 4 could continue for several years. We see that even when including the exploration period, our agent still saves 628-1327 kWh over the course of a year, or 5.7%-12.4%, depending on the house size and weather conditions. When excluding the exploration period the savings are 734-1572 kWh, or about 7.0%-14.5% per year. Figure 7 shows how our agent minimizes violations of the temperature comfort requirements; in each of the above simulations, we recorded the indoor-temperature at 6pm, which is the end of the don't care period. The figure shows a histogram of the 6pm temperatures at more than 22,000 simulated days. In Figure 8 we demonstrate how the RL agent controls the temperature in mild and extreme winter/summer days.

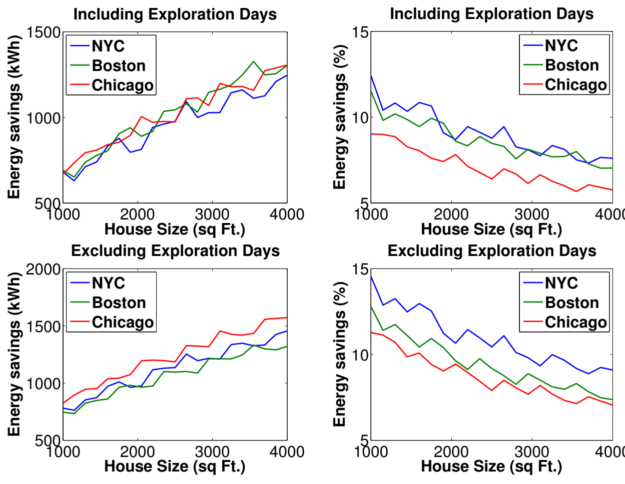

**Figure 6: Energy savings of our agent, in 21 different house sizes, using typical weather conditions recorded in NYC, Boston and Chicago, when including and excluding the 3 exploration days.**

## 4.2 Ablation Analysis

Table 1 shows an ablation analysis that tests the contribution of each of the agent's main components to its overall performance. All simulations are done on a single, typical 2500 $ft^2$ home using weather files recorded at New York City. Performance is summarized with respect to both energy consumption and satisfying comfort requirements. The "Comfort Violations" column displays the number of days in which the temperature was outside 69-75$^\circ F$ by 6pm, and the "Range of 6pm Temp." column displays the range of temperatures measured at 6pm throughout the year.

The bottom line of the table summarizes the final agent's performance. The upper part of the table summarizes the performance of the agent when one or more components are removed. Components are named as follows: 'prevAct' is the previous-action regression feature used by LearnHouseModel; 'hist' is the history of ten indoor temperatures that are used by LearnHouseModel; 'conf' is the dynamic confidence bound, or safety buffer, used inside TreeSearch. We see that removing each of these components by itself does not significantly increase the energy consumption, but removing 'conf' and 'hist' does result in a slightly reduced comfort. When removing the 'prevAct' and 'hist' together, energy consumption increases by 5.4% an comfort is violated more significantly due to the prediction errors in the transition function resulting from the absence of these features which are correlated with the hidden state of the house (Section 3.1). When removing all three features altogether, energy consumption increases by 9.5%, and comfort violations increase to a level in which the agent misses the specification by up to 10$^\circ F$. The bottom part of the table shows the how performance deteriorates when changing the dynamic safety buffer's constant to a value of 2, or making the buffer a fixed value of 2 standard deviations, both results in a good comfort performance, but too conservative behavior that leads to an increased energy consumption.

## 5. CONCLUSIONS

Heat-pump systems are gaining increased popularity as a part of the efforts to move society to a sustainable energy consumption. While setting back the temperature is an effective energy saving strategy in other HVAC systems, the common practice is to avoid setting back heat-pump systems, as when used with existing control strategies it actually increases energy consumption. In this paper, we have designed and implemented a complete reinforcement learning agent that learns an effective set-back strategy, which lead to roughly **7.0%-14.5%** of yearly energy savings in a realistic simulation of different house sizes and weather conditions. Our agent is *adaptive*, in the sense that when deployed in a new house, it learns the house properties and efficiently plans and executes a set-back strategy, which both saves energy starting the fourth day, and minimizes violations of the temperature comfort constraints. We see at least two possible directions for future work. The first one is taking our agent from simulation to reality, deploying it in a real heat-pump system. The second one is extending this strategy in simulation, to optimize more complex objectives, for instance minimizing price-based energy costs, which is a part of the smart-grid vision, in which energy price should encapsulate information such as the availability of renewable energy resources. As a part of this direction, it is possible

**Table 1: Ablation Analysis**

| Analysis Type | | Energy Consumption (kWh) | Comfort Violations (#) | Range of 6pm Temp. |
|---|---|---|---|---|
| **Removed Feature** | prevAct+hist+ conf | 1112(+9.5%) | 232 | 60.1-84.4 |
| | prevAct+hist | 1070(+5.4%) | 193 | 60.8-80.9 |
| | conf | 1024(+0.8%) | 138 | 67.5-78.3 |
| | hist | 1016(+0.0%) | 133 | 67.1-77.7 |
| | prevAct | 1015(+0.0%) | 65 | 67.8-76.5 |
| **Other conf. bounds** | $2\sigma$ | 1090(+7.3%) | 29 | 69.0-78.5 |
| | $c = 2$ | 1039(+2.3%) | 27 | 69.0-77.8 |
| **Final Agent** | | **1015** | **23** | **68.8-76.6** |

to further extend our strategy, and test it in a multi-agent smart grid environment in which many agents optimize costs and by doing that affect the resulting energy prices.

## 6. ACKNOWLEDGMENTS

We would like to thank Shivaram Kalyanakrishnan, Sam Barrett, Todd Hester and the rest of the members in the LARG group at The University of Texas at Austin, for useful discussions and feedback. We would like to thank Ron Eyal and Gil Gelber for fruitful discussions. We would like to thank Jason Fuller and Matt Hauer from the GridLAB-D team for their excellent support. Our numerical computations were done using the Armadillo [9] library.

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-0917122), ONR (N00014-09-1-0658), and the FHWA (DTFH61-07-H-00030).

## 7. REFERENCES

[1] R. Hafner and M. Riedmiller. Reinforcement learning in feedback control. *Machine Learning*, 84:137–169, 2011. 10.1007/s10994-011-5235-x.

[2] R. M. Kretchmar. *A Synthesis Of Reinforcement Learning And Robust Control Theory.* PhD thesis, 2000.

[3] D. J. C. Mackay. *Sustainable Energy – without the hot air.* UIT, Cambridge, MA, 2009.

[4] T. Peffer, M. Pritoni, A. Meier, C. Aragon, and D. Perry. How people use thermostats in homes: A review. *Building and Environment*, 46(12):2529–2541, 2011.

[5] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition.* Wiley, 2011.

[6] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[7] S. Ramchurn, P. Vytelingum, A. Rogers, and N. R. Jennings. Putting the "smarts" into the smart grid: A grand challenge for artificial intelligence. *Communications of the ACM*, 55(4):86–97, 2012.

[8] A. Rogers, S. Maleki, S. Ghosh, and J. Nicholas R. Adaptive home heating control through gaussian process prediction and mathematical programming. In *Second International Workshop on Agent Technology for Energy Systems (ATES 2011)*, pages 71–78, May 2011.

[9] C. Sanderson. Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.

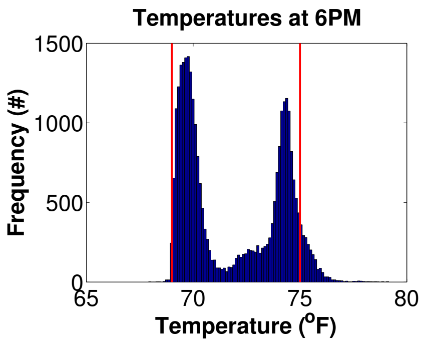

**Figure 7: Histogram showing how the agent minimizes violations of the temperature comfort requirements ($69 - 75°F$, vertical lines). The histogram shows the temperatures at the end of the don't-care period, namely 6pm, in more then 22,000 simulated days, using 21 different house sizes, using weather conditions recorded at the cities from Figure 6.**

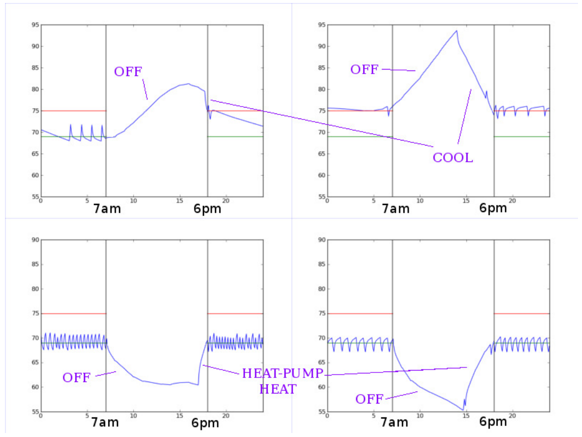

**Figure 8: Our agent controlling the temperature in a house in New York City area, in mild and hot summer days (top-left and top right, respectively), and mild and cold winter days (bottom-left and bottom-right, respectively). In all figures, x-axis is the time of day, y-axis is the temperature, the don't care period is between the two vertical lines, and the desired temperature range is between the red and the green lines. Note how in the top left, the agent waits until temperature drops back instead of starting to cool earlier.**