Model-Selection for Non-Parametric Function Approximation in Continuous Control Problems: A Case Study in a Smart Energy System

Daniel Urieli and Peter Stone

Dept. of Computer Science
The University of Texas at Austin
Austin, TX, 78712 USA
{urieli,pstone}@cs.utexas.edu

Abstract. This paper investigates the application of value-function-based reinforcement learning to a smart energy control system, specifically the task of controlling an HVAC system to minimize energy while satisfying residents' comfort requirements. In theory, value-function-based reinforcement learning methods can solve control problems such as this one optimally. However, since choosing an appropriate parametric representation of the value function turns out to be difficult, we develop an alternative method, which results in a practical algorithm for value function approximation in continuous state-spaces. To avoid the need to carefully design a parametric representation for the value function, we use a smooth non-parametric function approximator, specifically Locally Weighted Linear Regression (LWR). LWR is used within Fitted Value Iteration (FVI), which has met with several practical successes. However, for efficiency reasons, LWR is used with a limited sample-size, which leads to poor performance without careful tuning of LWR's parameters. We therefore develop an efficient meta-learning procedure that performs online model-selection and tunes LWR's parameters based on the Bellman error. Our algorithm is fully implemented and tested in a realistic simulation of the HVAC control domain, and results in significant energy savings.

1 Introduction

This paper is motivated by a real-world *discrete-time continuous control problem* in which the state space is *continuous* and the action space is *discrete*. Specifically, we focus on the task of controlling an HVAC system's thermostat[1] in a house with 'heat', 'cool', 'auxiliary-heat' or 'off' actions, with the goal of reducing yearly energy consumption while satisfying temperature comfort requirements for the occupants. Such discrete-time continuous control problems commonly arise when a digital controller controls a physical system, and when the possible control actions constitute either a finite set, or a low dimensional space that can be discretized without losing much control capability. Other examples of this class of problems are robot control, autonomous helicopter control, and autonomous car control. When the controlled system's dynamics is unknown in advance, *model-based Reinforcement Learning* can be used to efficiently

[1] HVAC: Heating, Ventilation, and Air-conditioning

learn the dynamics first, and then solve the control problem, possibly by computing or approximating a value function[13].

When using such a method to learn fine-grained control actions, one of the most crucial choices is how to represent the value function. One reason that this choice is so crucial is that the cycle time between consecutive control actions is typically short, compared to the time-range, or the *horizon*, over which the overall system behavior is optimized. Therefore, a single action often has a relatively minor effect on both the state of the system and on the immediate cost/reward, so that the overall performance is a sum of large number of minor contributions. Consequently, the value of a state that results from a *suboptimal* action is typically close to the value of the state that results from an *optimal* action. To induce the optimal policy, a value function approximator must be able to capture these fine differences between state values. Note that taking a suboptimal action may not seem like a problem when action effects are minor. However, since the problem's horizon can be orders of magnitude longer than the length of an action, the number of actions taken within the horizon is typically large, and repeatedly choosing suboptimal actions can accumulate to large losses.

Value function approximation is an active area of research: it is often unclear how to approximate the value function well enough so as to distinguish an optimal action from a suboptimal action. The three most common methods to approximate a value-function are lookup-tables, parametric methods, and non-parametric methods [16]. Lookup tables often suffer from Bellman's curse of dimensionality at the resolution levels that are required for continuous control problems. Parametric methods are typically computationally efficient, but assume that the value function takes some global, parametric form. Non-parametric methods make much weaker assumptions about the value-function's form, and therefore can, in principle, approximate any function. However, they typically require more data and computation than parametric methods.

One way to avoid the difficulties in approximating a value function in continuous spaces, is to use *direct policy search* methods, which directly optimize the parameters of some parametrized policy. Policy search methods have recently achieved several notable successes, e.g. [13,9,3], and have been gaining increased popularity for real-world control problems, perhaps due to the difficulties in approximating the value function in continuous spaces. However, if we could address the challenge of approximating the optimal value function well enough, we could gain some of the advantages of value-function-based methods over direct policy search methods, for instance aiming for global rather than local optimum, and requiring less interactions with the real-world due to bootstraping.

To address our HVAC control problem, we develop a general, practical, algorithm for approximating the value function in continuous state spaces. To avoid the need to carefully design a parametric representation for the value function, we use a smooth non-parametric function approximator, specifically Locally Weighted Linear Regression (LWR) (e.g. [2]). To compute the value function we use LWR within Fitted Value Iteration (FVI), an algorithm that has proven convergence properties and often performs well in practice [6,12]. However, being limited by a small sample size due to a run-time efficiency requirement on the system, we must tune LWR's parameters carefully, otherwise the system performs poorly. We therefore develop an efficient meta-learning procedure that performs online model-selection and tunes LWR's parameters. The model selection procedure is based on two main ideas, substantiated empirically through a large number of simulations. The first idea is that minimizing the empirical L_1 or L_∞ Bellman error of the approximate value function is correlated with optimizing performance on our task. It is shown that the same statement is not true for the L_2 Bellman error. The second idea is that minimizing the Bellman error by tuning LWR's parameters can be done efficiently. We note that while the Bellman error was used as a criterion for optimization by algorithms implementing generalized policy iteration using a *fixed* representation for the value function, and for tuning and generating basis functions in linear architectures, to the best of our knowledge it has not been used as an optimization criterion for tuning a non-parametric representation (see Sec. 6).

We apply our algorithm to the realistic control task of controlling a thermostat to optimize energy consumption while satisfying comfort requirements in a realistically simulated home. We build a complete Reinforcement Learning agent that uses our algorithm and show that (1) our agent outperforms the thermostat strategy that is deployed in practice, (2) our function approximation scheme leads to better performance than when using popular methods of value function discretization, linear function approximation with reasonable features, and non-parametric function approximation using equivalent computation with a much denser sample and without model-selection; and (3) our online model selection leads to performance that is close to that of an empirical upper-bound achieved using a state-of-the-art optimization method (CMA-ES [7]) combined with a clairvoyant model evaluator that returns the actual future performance of a model. The result is an adaptive value-function approximation algorithm for continuous state-spaces, which uses a non-parametric representation to minimize the assumptions about representation, and tunes it online to the specific environment in which it is deployed.

2 Preliminaries

2.1 Reinforcement Learning

In this paper we focus on solving control problems through Reinforcement Learning (RL) [19]. Reinforcement learning problems are often modeled as Markov Decision Processes (MDPs). An (episodic) Markov Decision Process (MDP) [18] is a tuple (S, A, P, R, T), where S is the set of states; A is a set of actions; $P : S \times A \times S \to [0, 1]$ 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; $R : S \times A \to \mathbb{R}$ is a reward function; and $T \in S$ is a set of terminal states, where entering one of which terminates an *episode*. 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 $\pi : S \to A$ from states to actions. A policy π induces a *value* for each state $s \in S$, denoted as $V^\pi(s)$, defined as the expected sum of rewards obtained by the agent when starting in state s and following policy π: $V^\pi(s) = E\left[\sum_{t=0}^{N} R(s_t, a_t) | s_0 = s, s_N \in T, \pi\right]$. $V^\pi(s) : S \to \mathbb{R}$ is called a *value function*. For a given MDP, there exists an *optimal* policy π^* such that $V^{\pi^*}(s) \geq V^\pi(s)$ for every s.

While $V^{\pi^*}(s)$ is induced by the policy π^*, it also *induces* π^*. It can be shown that $\pi^*(s) = argmax_{a \in A} \sum_{s' \in S} R(s,a) + P(s,a,s') \cdot V^{\pi^*}(s')$. Therefore, given $V^{\pi^*}(s)$ (and R, P), an agent can act optimally using a one-step look-ahead from any given state. This is the premise of *value function based RL*. For a finite S, there are algorithms that provably find $V^{\pi^*}(s)$ and therefore the optimal policy. When S is infinite, in general we can only compute an approximation $\hat{V}^{\pi^*}(s)$ of the optimal value function, and the best methodology to do so is still an open research problem. As RL does not need to know the system dynamics (model) in advance, it is an appropriate approach to control problems when the system dynamics are either unknown or partially known, and when a system is controlled in an uncertain environment to which it needs to adapt.

2.2 The Challenge of Function Approximation

The choice of function approximator can be crucial to determining an RL algorithm's performance in control problems of the type we consider. We start by defining a sufficient condition for a function approximator to induce the optimal policy. Denote $E_{[s'|sa]}[V^{\pi^*}(s')] := \sum_{s' \in S} P(s,a,s') \cdot V^{\pi^*}(s')$. In a given state s, let ϵ_s be the smallest absolute difference between the expected values of states resulting from two different actions taken from s, where one action is optimal and the other is sub-optimal:

$$\epsilon_s := \min_{\substack{a^*, a \in A \\ a^* = \pi^*(s) \text{ is optimal} \\ a \text{ is sub-optimal}}} \{ |E_{[s'|sa^*]}[V^{\pi^*}(s')] - E_{[s'|sa]}[V^{\pi^*}(s')]| \} \qquad (1)$$

Suppose that the system is in state s_0 and an action needs to be chosen.[2] In general, if the function approximator is able to approximate $V^{\pi^*}(s)$ to within $\frac{\epsilon_{s_0}}{2}$, meaning

$$\max_{s \in S} |\hat{V}^{\pi^*}(s) - V^{\pi^*}(s)| < \frac{\epsilon_{s_0}}{2} \qquad (2)$$

then greedy action selection based on $\hat{V}^{\pi^*}(s)$ is guaranteed to induce the optimal action from s_0, since:[3] $E_{[s'|sa^*]}[\hat{V}^{\pi^*}(s')] - E_{[s'|sa]}[\hat{V}^{\pi^*}(s')] > E_{[s'|sa^*]}[V^{\pi^*}(s') - \frac{\epsilon_{s_0}}{2}] - E_{[s'|sa]}[V^{\pi^*}(s') + \frac{\epsilon_{s_0}}{2}] \geq \epsilon_{s_0} - \frac{\epsilon_{s_0}}{2} - \frac{\epsilon_{s_0}}{2} = 0$. When condition 2 holds for every state $s_0 \in S$, the function approximator induces the optimal action in every state, and therefore the optimal policy. When it does not hold in every state, the function approximator may not induce the optimal policy.

Clearly, the smaller ϵ_s is, the harder it is to achieve the desired $\frac{\epsilon_s}{2}$ function approximation accuracy. Unfortunately, for the type of problems this paper is concerned with, namely real-world, discrete-time continuous control problems, ϵ_s can in fact be small for many states. This happens when the following two properties hold (for $S := R^n$):

– *Actions have "small" effect*: there exists some (relatively small) δ such that for every $s', s'' \in S$ that can result from taking actions at some state $s \in S$, it holds

[2] Note that we assume the existence of a sub-optimal action, since otherwise there is no decision of any consequence to be made, as all actions are optimal. Therefore $\epsilon_s > 0$.

[3] For simplicity of presentation, we neglect the 1-step reward, which could be incorporated into Equation (1) in a straightforward way.

that $||s' - s''|| < \delta$. This can happen, for instance, when the cycle time between subsequent control actions is short compared to the problem's horizon.

– *The optimal value function is Lipschitz continuous*: there exists some $K > 0$ such that $\forall s_1, s_2 \in S : ||V(s_1) - V(s_2)|| < K||s_1 - s_2||$. This holds, for instance, when the reward and the transition functions are Lipschitz continuous.

Combining the two conditions, we get that $\forall s \in S : \epsilon_s < ||E[V(s')] - E[V(s'')]|| \leq E[||V(s') - V(s'')||] < E[K||s' - s''||] < K\delta$, so that the smaller the action effect δ, the smaller ϵ_s is for any state s, and so the harder it is to achieve the desired approximation accuracy. In the results section we demonstrate how this results in repeated suboptimal actions, degrading performance on our thermostat control task.

3 Approximating the Value Function

RL algorithms that are based on value function approximation can roughly be divided into *model-free* algorithms, which are usually more computationally efficient, and *model-based* algorithms, which are usually more data efficient. As we are motivated by real-world problems, where gathering experience is often an expensive operation, we focus here on model-based RL. In model-based RL, an RL agent first explores the environment and learns an approximate model of it (namely P and R). Using this model, the agent simulates experiences and computes $\hat{V}^{\pi^*}(s)$. Since in this paper we focus on value-function approximation, we assume that an approximate model is either given, or was already learned by the agent. For instance, in the results section, our RL agent first learns an approximate model, and then uses it to compute $\hat{V}^{\pi^*}(s)$.

3.1 Approximate Dynamic Programming

For computing $\hat{V}^{\pi^*}(s)$, we start by using sampling-based Fitted Value Iteration (FVI) [6] (a detailed overview of its roots can be found at [12]). FVI is an approximate dynamic programming algorithm that computes $\hat{V}^{\pi^*}(s)$ by repeatedly scanning a finite sample of states $S_{FVI} := s^{(1)}, s^{(2)}, \dots, s^{(m)}$, applying the following two steps:

$$\forall i \in 1, \dots, m \tag{3}$$

$$y^{(i)} := max_a \left(R(s^{(i)}, a) + \gamma E_{[s'|s^{(i)}a]}[\hat{V}^{\pi^*}(s')] \right)$$

$$\hat{V}^{\pi^*}(s) := SL \left(\{ \langle s^{(i)}, y^{(i)} \rangle | i \in 1, \dots, m \} \right) \tag{4}$$

where $\hat{V}^{\pi^*}(s^{(i)})$ is initialized arbitrarily; the expectation over the resulting state is approximated by Monte-Carlo sampling; and after each update scan, a supervised learning algorithm *SL* is used as a function approximator that approximates the value function over the complete state-space, based on the "labeled" examples $\langle s^{(i)}, y^{(i)} \rangle$. While FVI is not guaranteed to converge to $V^{\pi^*}(s)$, it often performs well in practice, and is theoretically well-behaved [12]. In addition, FVI is an *off-policy* algorithm, which means it can approximate an optimal policy before ever executing it.

3.2 Function Approximator

Inside FVI, the function approximator we use as SL is Locally Weighted Linear Regression (LWR). LWR is a non-parametric, smooth function approximator that uses only minimal representation assumptions, and that has been used successfully to model complex real-world dynamics [13]. Given a set of m labeled examples $(x^{(i)}, y^{(i)})$ and a query point x for which we want to predict a value, our version of LWR does the following:

1. $w^{(i)} := exp\left(-\frac{(x^{(i)}-x)^2}{2\tau}\right)$ for $i = 1, \ldots, m$ [compute a weight for each training example]
2. Fit θ that minimizes $\sum_{i=1}^{m} w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$ [use weights for weighted regression]
3. Output $\theta^T x$

Here τ is a "bandwidth" parameter that determines how quickly the weights decay. Small weights are typically truncated for computational efficiency reasons. Since the weights $w^{(i)}$ depend on the specific query point, LWR builds a local model around the query for every prediction it makes. Note that when used with FVI, $x^{(i)} := s^{(i)}$, where $s^{(i)} \in S_{FVI}$. In general, LWR results in smoother function approximation than simpler non-parametric methods such as nearest-neighbors, which is a desirable property for continuous control tasks. However, in general, LWR can extrapolate, and this can prevent FVI from converging [6], so to ensure convergence we trim LWR's predicted value to be within the range defined by its neighbors values.

4 Efficient Model Selection

Like most learning algorithms, LWR usually needs to be tuned to work well for a particular problem. LWR is typically tuned by adjusting the values of the bandwidth parameter τ and of distance-metric-related parameters $c_1, \ldots, c_n$, where c_i is a scalar that scales s_i, the i'th state attribute in a state s. Adjusting the values of $c_1, \ldots, c_n$ effectively changes the distance metric based on the relative importance of state attributes. While it is possible to make c_i a general function $c_i : S \rightarrow \mathbb{R}$ rather than a scalar, we take an approach of *global tuning* [2], in which c_i is a scalar. This keeps the number of LWR parameters at a total of $n + 1$, so that tuning is more computationally efficient and less susceptible to overfitting. A given set of parameter values is said to define a *model* to be used by the LWR function approximator, and the process of tuning these parameters is a form of *model selection*.[4] The goal of our model-selection process is to find a set of parameters, that when used by LWR inside FVI, results in a function approximation that is close to the optimal value function.

When model-selection is done in a supervised learning setup, each candidate model is typically evaluated using *cross-validation*. In our setup, using cross-validation by holding out subsets of S_{FVI} is problematic since (1) we don't have the *actual* values of states $s \in S_{FVI}$ as we have in supervised learning, but only the values that FVI *converges* to, and (2) as S_{FVI} is typically sparse (to keep the run-time of FVI acceptable), having a good cross-validation accuracy on S_{FVI} does not necessarily imply

[4] Note that LWR's model is different than, and should not be confused with, the MDP model.

good prediction accuracy over the rest of the state space $S \setminus S_{FVI}$. Therefore, we seek an alternative model evaluation measure. The ideal way of evaluating a model is by measuring the agent's performance when acting based on a value function that uses this model. However, evaluating the agent in the real-world with different models is often prohibitively time-consuming and expensive. Therefore, we only use it as an empirical upper-bound in our simulated domain.

Instead, we use a theoretically-founded model evaluation measure that is efficiently computed in practice: the value-function's *empirical max Bellman error*. In a given state s, the absolute *Bellman (optimality) error* of a function $\hat{V} : S \rightarrow \mathbb{R}$ is defined as:

$$BE_{\hat{V}}(s) := |\hat{V}(s) - max_a(R(s,a) + \gamma E_{[s'|sa]}[\hat{V}(s')])|$$

Next, for a function $\hat{V} : S \rightarrow \mathbb{R}$ the following holds:

$$\hat{V} \equiv V^{\pi^*} \iff \forall s \in S : BE_{\hat{V}(s)} = 0 \tag{5}$$

Furthermore, [21] (resp. [12]) establishes that for a full (resp. sample) Bellman backup:

$$|V^{\pi^*}(s) - \hat{V}(s)| \propto BE_{\hat{V}}(s) \tag{6}$$

Equations (5), (6) imply that ideally the Bellman error would be 0, or as close as possible to 0, for every state. *Note that while the convergence of FVI means that* $\forall s \in S_{FVI}$: $BE_{\hat{V}(s)} \approx 0$*, the Bellman error might still be large for states* $s \in S \setminus S_{FVI}$. In order to address that, we create a random sample of test states $\mathcal{T} := \{t^{(1)}, ..., t^{(m')}\}$, and define a vector of Bellman errors (overloading notation):

$$BE_{\hat{V}}(\mathcal{T}) := (BE_{\hat{V}}(t^{(1)}), \ldots, BE_{\hat{V}}t^{(m')}) \tag{7}$$

Motivated by Equations (5) (6), we use the max Bellman error $||BE_{\hat{V}}(\mathcal{T})||_\infty$ as a model-evaluation measure when tuning LWR's parameters. This model evaluation measure is computed solely based on a value function computed by FVI, without needing more data or interactions with the environment. Our model selection process then becomes a continuous optimization problem of finding a set of LWR parameters $\psi \in \mathbb{R}^{n+1}$ that minimizes $||BE_{\hat{V}}(\mathcal{T})||_\infty$ where $\hat{V}$ is the resulting value function after running FVI with LWR using ψ. Ideally, the max Bellman error should be computed over all states in the state space, however since the state-space in non-enumerable, we take a practical approach and set $\mathcal{T}$ to be a (as dense as computationally possible) random sample of states. In the results section we show that (a) minimizing $||BE_{\hat{V}}(\mathcal{T})||_\infty$ is correlated with good actual performance, and that (b) minimizing it can be done efficiently.

Putting all of these components together, the main general contribution of the paper beyond the domain-specific results is the MSNP algorithm (Model Selection for Non-Parametric function approximation). As summarized in Algorithm 1, it executes the following steps. The algorithm's input is a learned MDP model, and an iterative continuous optimization algorithm, that finds a minimum of a function $F : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$. Since we generally do not have the gradient of the Bellman error as a function of the LWR parameter set, we use gradient free optimization algorithms in our experiments. An interesting future extension would be comparing them with subgradient methods. MSNP starts by generating a sample $\mathcal{S}$ of states for FVI (step 1) and a sample of test states $\mathcal{T}$ over which it would compute the max Bellman Error (step 2). It then initializes a vector v of Bellman Errors (step 3) and a vector of LWR parameters ψ (step 5). In the main loop, it repeatedly runs the following steps until the max Bellman Error converges: run FVI (step 8), compute the resulting max Bellman Error (step 9-11), send it back to the optimization algorithm as the evaluation of the current parameter set (step 15), and get from the optimization algorithm a new set of LWR parameters to evaluate (steps 16-17).

Algorithm 1 MSNP(MDP-Model, OptimizationAlgorithm)

```
1: $\mathcal{S} \leftarrow \{s^{(1)}, s^{(2)}, \ldots, s^{(m)}\}$ the set of points used by *FittedValueIteration*
2: $\mathcal{T} \leftarrow \{t^{(1)}, s^{(2)}, \ldots, t^{(m')}\}$ test points sampled within the boundaries of $\mathcal{S}$
3: $v \leftarrow \{\infty, \ldots, \infty\} \in R^{m'}$
4: *PreviousError* $\leftarrow \infty$
5: $\psi \leftarrow$ *OptimizationAlgorithm.initializeParameters()*
6: *done* $\leftarrow False$
7: **while** *not done* **do**
8:     $\hat{V} \leftarrow FittedValueIteration($MDP-Model$, \mathcal{S}, \psi)$ // approximate value function
9:     **for** $i = 1 \rightarrow m'$ **do**
10:        $v_i \leftarrow BE_{\hat{V}}(t^{(i)})$ // the Bellman error in $t^{(i)}$
11:    **end for**
12:    *MaxBellmanError* $\leftarrow ||v||_\infty$
13:    **if** |*MaxBellmanError - PreviousError*| $< \epsilon$ **then**
14:        *done* $\leftarrow$ *True*
15:    **else**
16:        *OptimizationAlgorithm.observe(MaxBellmanError)*
17:        $\Delta\psi \leftarrow$ *OptimizationAlgorithm.step()*
18:        $\psi \leftarrow \psi + \Delta\psi$
19:        *PreviousError* $\leftarrow$ *MaxBellmanError*
20:    **end if**
21: **end while**
```

MSNP is an efficient algorithm for approximating the value function in continuous state spaces, that uses our model-selection procedure for tuning a Fitted Value Iteration with Locally Weighted Linear Regression, and which has the following properties:

1. *General*: It uses only minimal assumptions about a value function's representation, by using an non-parametric function approximator (LWR).
2. *Practical*: It tunes the representation of the LWR function approximator without the need to evaluate the agent in the environment but rather based on an internal property of the value function, namely the Max Bellman Error. In that sense, it is data efficient.
3. *Adaptive*: It tunes online the function approximator's representation to the environment it is deployed in.
4. *Effectively use computation*: Its model selection procedure achieves better performance than when using the same amount of computation on a larger S_{FVI} sample without any model-selection.

5 Results

In this section we investigate the application of MSNP to the task of controlling an HVAC's thermostat in a realistically simulated home.

5.1 Experimental Setup

Our experiments are run using GridLAB-D[5], an open-source smart-grid simulator that was developed for the U.S. Dept. of Energy. It models a residential home, including 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 uses meteorological data collected by the National Renewable Energy Laboratory[6] in cities across the USA. In our experiments, GridLAB-D simulates a residential home with a heat-pump based HVAC system, which is widely used due to its high efficiency.

We assume that occupants are at home between 6pm and 7am of the next day, and that the house is empty between 7am and 6pm (referred to as the *don't-care* period). Our goal is to (1) minimize the energy consumed by the HVAC system, while (2) keeping a desired temperature range of 69-75$^{\circ}(F)$ when the occupants are at home, and being indifferent to temperature otherwise (Figure 1). Due to uncertainty in future weather and in the house's environment, simple strategies fail to satisfy at least one of the requirements. For instance, turning the system off at 7am and turning it back on at a fixed time, such as 6pm, or even earlier, can fail to satisfy *both* requirements in cold winter days, since the temperature gets significantly out of range at 6pm and restoring it might take several hours, during which comfort is violated. Moreover, while doing so, an energy expensive auxiliary heater is used (since the heat-pump becomes inefficient), and the resulting energy is higher than when just keeping the temperature between 69-75$^{\circ}(F)$ throughout the day. We model the problem as an episodic MDP[7], as follows:

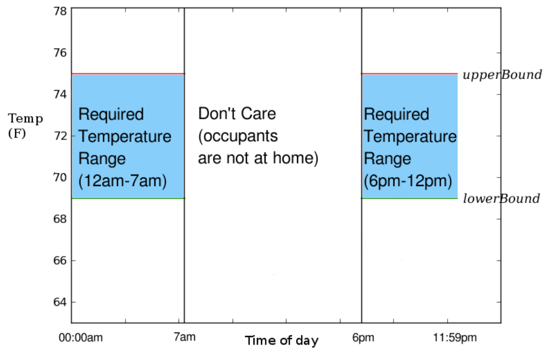

Fig. 1. Temperature Requirements Specification.

[5] http://www.gridlabd.org
[6] http://www.nrel.gov

- $\mathbf{S}$: $\{\langle T_{in}, T_{out}, Time\rangle |$ T_{in} and T_{out} are the indoor and outdoor air temperatures (in Fahrenheit), and $Time$ is a 24-hour clock time (in minutes).$\}$
- $\mathbf{A}$: $\{\text{COOL}, \text{OFF}, \text{HEAT}, \text{AUX}\}$. Namely, there are four possible actions for cooling, off, (heat-pump-)heating and auxiliary heating, respectively.
- $\mathbf{P}$: computed by the GridLAB-D simulator and is initially unknown to the agent.
- $\mathbf{R}$: $-$(the energy consumed by the last action) $-C_{6pm}$. Here, C_{6pm} is a quadratic cost applied when missing the temperature spec at 6pm.
- $\mathbf{T}$: $\{s \in S | s.time ==$ 23:59pm$\}$

For testing the MSNP algorithm, we build a full RL agent that controls the thermostat. Since our focus is on value function approximation, we leave the problem of sample-efficient exploration and model-learning outside the scope of this work. Instead, to cover diverse weather conditions, the agent explores during don't-care periods of one simulated year, where the OFF action is chosen with probability of $(1 - \frac{currentTime - \text{7am}}{\text{6pm} - \text{7am}})$. Otherwise, cooling or heating is chosen, depending on whether the indoor temperature high or low, respectively. If heating is chosen, then HEAT or AUX are chosen with probabilities 0.9 and 0.1 respectively. While exploring, the agent collects tuples of the form $\langle s, a, r, s'\rangle$, which are the current state, action, reward, and next state. The agent uses these tuples as labeled examples $\langle s, a\rangle \rightarrow r$ and $\langle s, a\rangle \rightarrow s'$ for fitting the functions R and P with linear regression using state features and their squares, where s is represented as $\langle 1, T_{in}, T_{out}, Time, {T_{in}}^2, {T_{out}}^2, Time^2\rangle$. This representation was chosen based on a small amount of trial and error. Adding the squares was intuitively aimed at addressing non-linearity in the transition, to some extent. Using P and R, the agent runs MSNP and acts greedily based on $\hat{V}^{\pi^*}$ for an additional year.[8] Inside MSNP, FVI uses a state sample S_{FVI} arranged as a grid inside the three-dimensional state space, of size 20x10x20=4000. For running LWR inside FVI we use the 15-nearest neighbors, and the grid structure allows us to find them in constant time.

5.2 Sensitivity to Errors in Function Approximation

Figure 2 demonstrates the difficulty of value function approximation in continuous domains with short actions, using the thermostat control task. The x-axis is the 24-hour time of day and the y-axis is the indoor temperature controlled by the actions of the agent, who acts greedily based on an approximate value function. The agent turns the system off during the don't-care period, letting the temperature rise, and eventually cools in advance to return the temperature back to range by 6pm. However before starting to cool, there are several heating actions that are physically wrong, chosen due to small approximation errors in the value function, in this case due to using LWR without tuning its parameters. Each suboptimal 2-minute action increases the daily consumption by only about 0.1%, but repeatedly taking them can increase consumption by 10% and more. These suboptimalities and more severe ones happened when using discretized

[7] An action is taken every 2 minutes, as the simulator models a realistic lockout of the system.

[8] In practice, Gridlab-D only has one year of "average" weather data. We therefore used 9 of every 10 days during the training year, and the remaining days during the testing year so as to have separate training and testing data. Our reported results reflect the average of repeating this experiment 10 times with each different possible subset of "held-out" days.

representations as well as linear value-function representations with reasonable features.

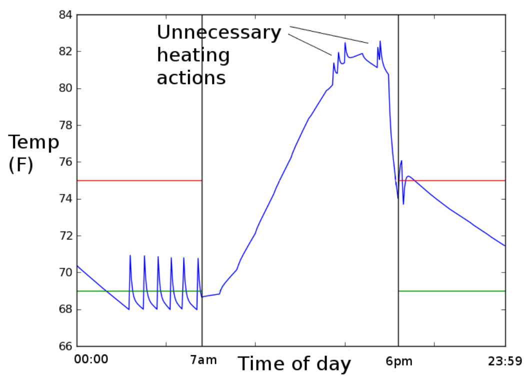

Fig. 2. Suboptimal policy due to func. approximation errors.

5.3 Using the Max Bellman Error for Model-Selection

Next, we investigate using the Bellman error as a criterion for model-selection of our LWR function approximator. The plots in Figure 3 show the correlation between empirical Bellman error in the approximate value function $\hat{V}^{\pi^*}$ and the agent's performance when acting based on $\hat{V}^{\pi^*}$. The plots summarize 10,000 experiments, each represented as a point. Each experiment tests a set of $n+1$ LWR parameters, which defines a *model* used by LWR, as was discussed in Section 4. In the thermostat domain there are three state attributes, and therefore four model parameters. We sweep the parameter space by setting each parameter to one of 10 possible values, and this gives $10 \times 10 \times 10 \times 10 =$ 10,000 possible parameter sets. The empirical Bellman errors were measured as the L_1, L_2 or L_∞ norms of the vector of Bellman errors in $\hat{V}^{\pi^*}$ over a uniformly random sample of $|\mathcal{T}| = 256,000$ states. It can be seen that when the L_1 and L_∞ errors are smallest, performance is expected to be close to the best possible (lowest energy consumption). The same does not hold for the L_2 error, as minimizing the L_2 error results in consuming 4% more energy than the best result. Note that in general these plots clearly highlight the need for model-parameters tuning, as untuned parameters can consume about 25% more energy than the best possible parameters.

5.4 Efficiently Optimizing the Bellman Error

The previous section tested the first of two steps for creating an efficient model-selection algorithm. We saw that model-selection, or representation tuning, of the LWR function approximator can be done without the need to evaluate an agent in the environment, but

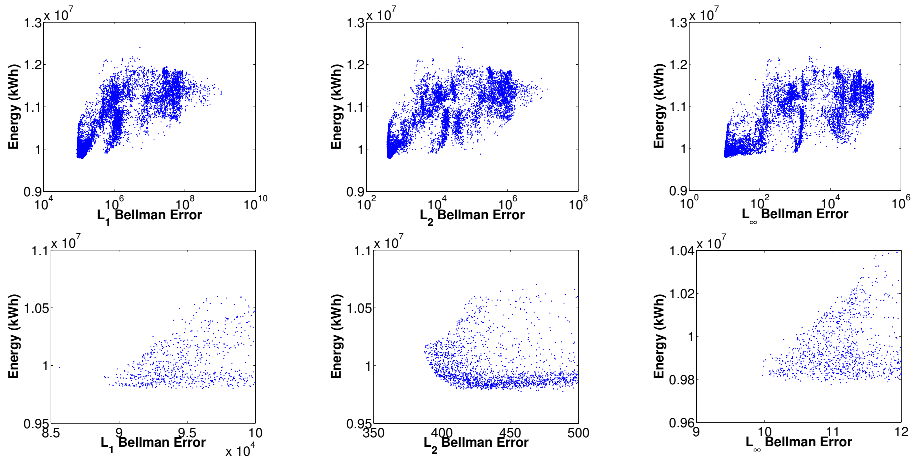

Fig. 3. Bellman errors (x-axes) vs. actual performance (y-axes, lower is better). Top row: full plots. Bottom row: zoom into the bottom-left corner (best performance) of each top row plot.

rather based on an internal property of the value function, namely the L_∞ or L_1 Bellman errors, so in that sense it is data efficient. Next, we show that tuning the model parameters based on the max Bellman error can be done computationally efficiently. Note that once using the Bellman error for model evaluation, our model-selection problem becomes minimizing an objective function that maps an LWR parameter set to the max Bellman error in $\hat{V}^{\pi^*}$ computed using this parameter set by FVI with LWR. We compare several efficient local-search derivative-free algorithms for finding the minimum of a continuous function.[9] The algorithms we compare are Powell's method, Nelder–Mead method, also known as Amoeba, and a coordinate-descent algorithm in which we hold all parameters fixed and optimize one parameter at a time using Brent's method, using implementations from [17]. Results are shown in Figure 4. The horizontal gray line in the figure is the best max Bellman error that was achieved when using an offline, state-of-the-art, parallel optimization algorithm CMA-ES [7], when running it with 10,000 function evaluations (100 generations with a population-size of 100). It can be seen that after about 30 function evaluations all three methods get close to CMA-ES's value, and that the Brent's method-based coordinate-descent reaches there after about 15 function evaluations. Our FVI implementation converges in less than 2-minutes on a standard desktop machine, so that 15-30 function evaluations takes 30-60 minutes.

How robust is running local optimization for finding a global minimum of the max Bellman Error? To try to answer this question, we fixed the parameter values at $c_1 = c_2 = c_3 = 0.5$, $\tau = 0.0005$, and then changed one parameter at a time across its range ($c_i \in [0.05, 1], \tau \in [0.00005, 0.0010]$) measuring the max Bellman error as a function of this parameter, where as usual, the max Bellman error was computed over the value function computed using FVI with LWR. Results are in Figure 5, and show that while the max Bellman error is not a convex function of representation parameters, it still has a relatively large basin of convergence.

[9] In general we do not have derivative information for the optimized function

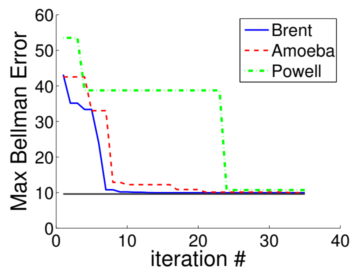

Fig. 4. Comparing different optimization algorithms on the task of finding a parameter set that minimizes the max Bellman error.

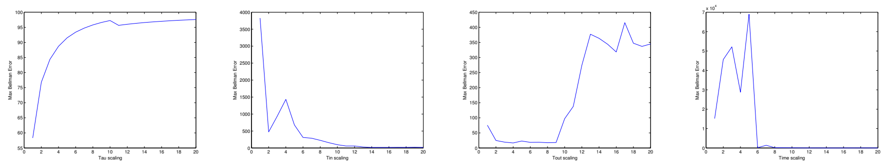

Fig. 5. Bellman Error basin of convergence: Bellman Error (y-axis) as a function of each LWR parameter, holding all the other parameters fixed. The x-axes from left to right: τ, $c_{T_{in}}$, $c_{T_{out}}$, c_{Time}

5.5 Performance of MSNP

Finally, Table 1 demonstrates the advantage of using MSNP, by testing it on our thermostat control problem. In these experiments, we use our RL agent, changing only the way the value function is approximated. The table shows the energy consumed by the HVAC system over one simulated year, in which the agent acts greedily based on each of the different approximate value functions. Simulations were run using real weather files from three different cities in the US. As a reference, the "Default" column shows the results of using a default heat-pump thermostat strategy that is used in real-world deployments, which just keeps the temperature between 69-75$^\circ(F)$ throughout the day. This strategy does not shut down the system during the don't-care period, since doing so without knowing how long in advance to turn the system back on can result in violating either or both requirements (1) and (2), as discussed above. The "LargeSample" column was generated when using FVI with LWR to approximate the value function, but instead of using the model-selection like MSNP does, the agent spends the same amount of computation on just running FVI with LWR on a larger state sample S_{FVI} of 160x80x160=2048000 states without any model selection, and using default values for the LWR parameters, similar to the values used in Section 5.4: $c_i = 0.5$ for $i = 1, ..., n$ and $\tau = 0.0005$.[10] MSNP was run using a state sample S_{FVI} of 20x10x20 states and used $|\mathcal{T}| = 256,000$ states for computing the Bellman error, so that its

[10] The "LargeSample" results are actually slightly better than they should be because they did not use the 9 days of 10 methodology referenced in footnote[8]. It was trained on the full year of data.

computation time, dominated by the number of LWR predictions was no larger than that of LargeSample's. The "CMA-ES" column serves as an empirical upper-bound on performance in our simulated domain. It was generated by running the state-of-the-art CMA-ES optimization method to perform model selection on top of FVI with LWR, using (1) 10,000 model evaluations (100 generations, each with a population size of 100), and (2) a clairvoyant model-evaluator that returns the *agent's actual future performance using a given model*, by running a one-year simulation using this model. It can be seen that MSNP performs better than "LargeSample", which demonstrates that online model-selection has an advantage over just increasing the density of the sample size. MSNP's performance is close to that of CMA-ES's, despite the fact that it uses only 40 function evaluations (instead of 10,000) and doesn't have access to the "real" model evaluation measure of the unknown future performance, that CMA-ES has. Note that while the MSNP and CMA-ES agents satisfied the temperature comfort requirements, the LargeSample agent frequently did not satisfy them. In Figure 6 we demonstrate how the RL agent controls the temperature in mild and extreme winter/summer days.

Table 1. Performance of an agent using MSNP

City	Default (kWh)	LargeSample (kWh)	MSNP **(kWh)**	CMA-ES (kWh)	% Energy-Savings
New York City	11084.8	10923.5	**9859.3**	9816.3	**11.0%**
Boston	12277.1	12480.7	**11433.6**	11052.8	**6.9%**
Chicago	15172.5	14778.2	**14186**	13778.4	**6.5%**

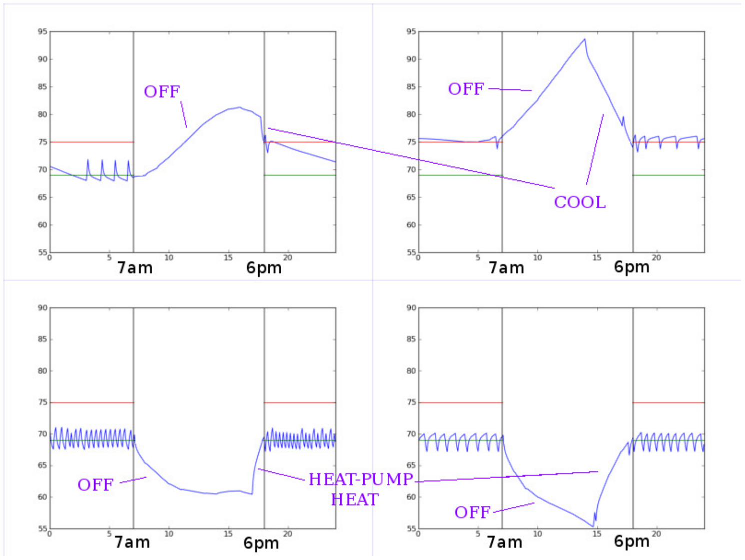

Fig. 6. Our agent controlling the temperature in a house in New York City area, in mild and hot summer days (top-left and top right, respectively), and mild and extreme winter days (bottom-left and bottom-right, respectively).

6 Related Work

RL has been applied to realistic control tasks, however recent successes frequently used policy search methods, rather than value-function-based methods(e.g. [13,9,3]). Value function based RL has had success on some robotic tasks [15], but there the assumption was that the value function can be represented as a predetermined set of basis functions, an assumption that does not necessarily hold in the general case. Non-parametric value function approximation methods have been suggested, e.g. in [4]. The idea of using the Bellman error as a criterion for optimization has been used by algorithms implementing generalized policy iteration, e.g. in [10,1]. The Bellman error has also been used for tuning and generating basis functions adaptation in linear function approximation architectures [11,8,14], while here we use it to tune a non-parametric representation. The model selection proposed here is different then the model selection done by [13]. There, the setup was offline, supervised learning for learning the transition function, while ours is an online reinforcement learning setup, for approximating the value function, where there are no labels over the data, but only the values to which FVI converge to, which could be different then the real state values. A paper that is closely related to ours is [5], which designs an abstract model-selection algorithm and proves theoretical guarantees about it. Similarly to here, they consider batch RL, in which a data set D of sampled transitions from the MDP is given, and is used for selecting a candidate value function by minimizing a Bellman error. In their case they abstract the way value function candidates are generated and assume they are independent of D, while here we actually use D to approximate the model and generate candidates using MSNP. Their theoretical guarantees are proved under a slightly different setup, and it would be interesting to explore whether they can be extended to our setup. The problem of thermostat control was addressed in [20], but there the focus was on solving the complete RL problem, including exploration, model learning and planning, and no value function approximation was used, while here the focus is on investigating the application of value-function based RL to the continuous, realistic domain of HVAC thermostat control.

7 Conclusion

This paper presents the application of value-function-based RL to the real-world smart-energy application of controlling an HVAC thermostat to minimize energy consumption while satisfying temperature comfort requirements, along with detailed empirical results and analysis. In addition, the paper introduces MSNP, which is a general, practical algorithm for approximating the value function for continuous control problems, using an efficient model-selection procedure based on the Bellman error.

This paper opens up several interesting directions for future work. For example, it is worth investigating the Bellman error's basin of convergence as a function of the model-parameters. Another interesting direction is exploring the use of subgradient methods for minimizing the Bellman error, and comparing them with the gradient-free methods we used. Finally, an important future direction is to expand MSNP's empirical analysis by including more domains and competing methods, and to evaluate it in higher-dimensional state spaces.

References

1. Antos, A., Szepesvári, C., Munos, R.: Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. Mach. Learn. 71(1), 89–129 (Apr 2008)
2. Atkeson, C.G., Moore, A.W., Schaal, S.: locally weighted learning (1997)
3. Deisenroth, M.P., Rasmussen, C.E.: PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In: Getoor, L., Scheffer, T. (eds.) Proceedings of the 28th International Conference on Machine Learning. Bellevue, WA, USA (June 2011)
4. Engel, Y., Mannor, S., Meir, R.: Reinforcement learning with gaussian processes. In: In Proc. of the 22nd International Conference on Machine Learning. pp. 201–208. ACM Press (2005)
5. Farahmand, A.M., Szepesvári, C.: Model selection in reinforcement learning. Mach. Learn. 85(3), 299–332 (Dec 2011)
6. Gordon, G.J.: Stable function approximation in dynamic programming. In: in Machine Learning: Proceedings of the Twelfth International Conference. Morgan Kaufmann (1995)
7. Hansen, N.: The CMA Evolution Strategy: A Tutorial (January 2009)
8. Keller, P.W., Mannor, S., Precup, D.: Automatic basis function construction for approximate dynamic programming and reinforcement learning. In: Proceedings of the 23rd international conference on Machine learning. pp. 449–456. ICML '06, ACM, New York, NY, USA (2006)
9. Kohl, N., Stone, P.: Machine learning for fast quadrupedal locomotion. In: The Nineteenth National Conference on Artificial Intelligence. pp. 611–616 (July 2004)
10. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. J. Mach. Learn. Res. 4, 1107–1149 (Dec 2003)
11. Menache, I., Mannor, S., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Annals of Operations Research 134, 215–238 (2005)
12. Munos, R., Szepesvári, C.: Finite time bounds for sampling based fitted value iteration. In: ICML. pp. 881—886 (2005)
13. Ng, A.Y., Kim, H.J., Jordan, M.I., Sastry, S.: Autonomous helicopter flight via reinforcement learning. In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems 16. MIT Press, Cambridge, MA (2004)
14. Parr, R., Painter-Wakefield, C., Li, L., Littman, M.: Analyzing feature generation for value-function approximation. In: Proceedings of the 24th international conference on Machine learning. pp. 737–744. ICML '07, ACM, New York, NY, USA (2007)
15. Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing 71(79), 1180 – 1190 (2008)
16. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition. Wiley (2011)
17. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing. Cambridge University Press, New York, NY, USA, 3 edn. (2007)
18. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA, 1st edn. (1994)
19. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
20. Urieli, D., Stone, P.: A learning agent for heat-pump thermostat control. In: Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (May 2013)
21. Williams, R.J., Baird, III, L.C.: Tight performance bounds on greedy policies based on imperfect value functions. In: Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems (1994), available at `http://leemon.com/papers/1994wb.pdf`