Generalized Demand Response for Flexibility Carriers using Reinforcement Learning

Lucas v. H. S. Pompe (L.vanhoeijschilthouwerpompe@uu.nl)
Thesis Bsc. Artificial Intelligence: 7.5 ECTS
4167546
Utrecht University

Supervisor:
dr. Gerard A. W. Vreeswijk

7 April, 2017

Abstract

In recent work concerning demand response for flexibility carriers, specialized methods have achieved considerable progress for specialized instances of carriers. These methods often rely on complicated design decisions in feature engineering or utility function design. Furthermore, flexibility carriers are often stochastic in their behaviour. In this paper we propose a general model-free reinforcement learning approach using limited feature engineering and a straightforward utility function. We validate our approach on a simulation of a flexibility carrying cold storage cell. Our results indicate significant cost savings can be achieved through our approach, at the cost of a long exploration period. Our approach requires approximately 69 simulated days before offering an improvement in cost over standard carrier behaviour.

1 Introduction

The path towards a world supplied only by renewable resources remains cumbersome. Renewable production methods like solar and wind are erratic by nature, and human consumption seems resilient to radical change. The search for adaptation of our energy consumption to the erratic nature of renewables has seen the emergence of a wealth of algorithms. Most of these methods focus on so called *flexibility carriers*. These systems do not need continuous operation, and thereby allow for flexibility in the moment they do operate. Examples of flexibility carriers include swimming pool heaters, water pumps and cold storages. Shifting the energy consumption of a flexibility carrier to a more desirable (often cheaper) hour of the day is known as *demand response*. Here energy market mechanisms allow for a straightforward objective: As renewable energy sources increase their production throughout a day, the market will adjust by lowering its prices, making it more desirable to operate flexibility carriers. The benefit of operating flexibility carriers in these hours is twofold. First, less energy is needed from pollutant sources. Second: operating during these hours reduces operation costs.

Concurrently, recent advances in artificial intelligence have generated renewed interest in the reinforcement learning paradigm. Reinforcement learning (RL), is a term used for a collection of methods used to allow an agent to learn from its actions through interaction with an environment (Barto & Sutton, 1999). In terms of flexibility carriers, a reinforcement learning algorithm can learn to operate the flexibility carrier so that the carrier is only operational during affordable hours. Exploratory work suggests model-free variants of reinforcement learning offer a benefit over various model-based methods, as carrier behaviour models are rarely known (Ruelens *et al.*, 2015). Moreover, model-based methods often struggle in stochastic environments (Cigler *et al.*, 2013).

However, work surrounding reinforcement learning often leaves room for improved generalizability as adoption of demand response methods becomes more prevalent. Some methods require differentiable reward functions (O'Neill *et al.*, 2015) or heavy feature engineering (Ruelens *et al.*, 2015). In this paper we explore the feasibility of a general reinforcement learning approach on a simulated flexible cold storage system. We show significant cost savings can be achieved through reinforcement learning without the need for heavy feature engineering or differentiable reward functions. The structure of this thesis is as follows: First we explain basic concepts surrounding our objective when applying demand response. Second we concentrate on reinforcement learning and address some of its challenges. In section 4 we explain the specifics of our simulated cold storage cell. Section 5 is used to analyze the results of our experiment. Finally, in section 6, we explain the implications of our results concerning future work and the accuracy of our experiment before reflecting on our objective. Moreover, appendix A contains all parameters used in our experiment along with their assigned values.

2 Basic concepts

Before we describe the specifics of the reinforcement learning paradigm, we will define the tasks objective. Given a cold storage enviroment and a time span, denoted by time steps $t_0, t_1 \dots t_n$, our objective is to minimize the cost c of turning on our cooling cell. The cost of our cell from time step 0 through n is defined as:

$$c = \sum_{t=0}^{n} u_t \lambda_t .$$

Where u_t is the energy consumption of our cell at time step t, and λ_t, the energy price at t. The energy consumption of a cooling cell is controlled by the operational state $z \in \{0, 1\}$, and the energy capacity ρ of the cell:

$$u_t = \begin{cases} \rho, & \text{if } z_t = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Because z is discrete, this is equivalent to $u_t = \rho z_t$. Changes in the operational state influence the internal temperature T_t of the cell as well. Moreover, when minimizing c, we will need to keep the internal temperature of the cell between boundaries T_{min} and T_{max} to ensure proper preservation of the contents of the cell.

3 Reinforcement learning

3.1 Markov decision processes

Given an environment, reinforcement learning allows us to learn to interact with the environment. In our case, an environment would consist of a cold storage system defined by a set of possible states S. An agent interacts with the environment by iteratively choosing an action from the set of possible actions A. Choosing an action changes the state according to a transition function that dictates a state's successor given an action $a \in A$. Often the environment satisfies the Markov property. An environment is said to satisfy the Markov property if its actions only affect a state's successor. If an environment satisfies this property, the accompanying reinforcement learning task is called a Markov decision process (MDP). Depending on the cardinality of state- and action spaces, an MDP is either considered finite or infinite. (Barto & Sutton, 1998) In this paper we will examine the efficacy of reinforcement learning on an infinite Markov decision process involving a cold storage environment.

3.2 Value functions

In general terms, the goal of reinforcement learning is to find the optimal policy π^* which maximizes some value given by the value function V. A policy π selects actions based on the observations an agent makes in a certain enviroment:

$$\pi : S \mapsto A.$$

A value function V assigns a value to all states, thereby allowing an agent to differentiate states based on their desirability:

$$V^{\pi} : S \mapsto \mathbb{R}.$$

The value of a state $s \in S$ under policy π, given by $V^{\pi}(s)$, is the expected reward of state s when following policy π (Wiering & Otterlo, 2014). Because we want maximize the value of different states through the actions given by our policy, it can be useful to distinguish between the values of different state-action pairs. This is achieved through a Q-function:

$$Q^{\pi} : S \times A \mapsto \mathbb{R}.$$

The Q-value of a state-action tuple is given by:

$$Q^{\pi}(s_t, a) = R(s_t, a, s_{t+1}) + \gamma \cdot (\max_{a} Q^{\pi}(s_{t+1}, a)).$$

This defines the Q-function as the sum of the reward gained by taking action a in state s and the discounted value of acting optimally from the next state onward. Where R is some reward function assigning a reward to action a in state s_t, s_{t+1} is the observed state after taking action a in state s_t, and γ is the discount factor. A discount factor is used to assign a weight to rewards gained after taking an action in the current state. Typically, a lower discount will result in a more greedy policy, while higher discount factors will result in a policy geared towards long term reward. However, this definition of the Q-function introduces two dependencies.

3.3 Reward

First: to solve our problem, we will have to design a reward function R that succeeds in capturing our definition of successful demand response. As specified earlier, the goal of our RL approach is to shift energy consumption of cold storage systems to cheaper hours. Therefore the definition of our reward function R should be closely related to the monetary cost of the power consumption as realized by our policy. However, defining the reward function as the monetary cost of our policy will cause any non-trivial policy to cease all power consumption. Not consuming any power will always be cheaper than consuming power. Therefore our reward function will have to incorporate the dissatisfaction of letting the cold storage heat or cool beyond the range $[T_{\min}, T_{\max}]$. Additionally, in most cold storage systems switching the cooling state of the system introduces strain on machinery. Thus introducing unnecessary strain into the system through switching should increase dissatisfaction. Our reward function has three components to address these requirements.The first component, R_{switch} , is meant to penalize excessive state switching:

$$R_{\text{switch}}(s_t, a) = \begin{cases} -1, & \text{if } z_t \neq z_{t+1}, \\ 0, & \text{otherwise.} \end{cases}$$

As our action should dictate the operational state of the cell in the next time step, the first conditional is equivalent to $z_t \neq a$.

The component R_{range} is meant to keep temperatures in the range $[T_{\min}, T_{\max}]$:

$$R_{\text{range}}(s_t, a) = \begin{cases} -(T_t - T_{\max} + 1)^{\eta}, & \text{if } T_t > T_{\max}, \\ -(T_{\min} - T_t + 1)^{\eta}, & \text{if } T_t < T_{\min}, \\ 0, & \text{otherwise.} \end{cases}$$

Where η is a parameter to control the growth of the penalty when T_t exceeds the range $[T_{\min}, T_{\max}]$. With correct parameterization of η, this penalty allows for brief violation of the range, while combatting extended violations. This design differs slightly from Ruelens *et al.* (2015) where any violation would result in a sizeable penalty. Our design allows for minor violations, to allow a policy to exploit the added flexibility. Additionally, minor violations often happen during standard operation. Lastly, R_{price} rewards a policy for consuming less energy, and doing so at lucrative hours:

$$R_{\text{price}}(s_t, a) = -u_t \lambda_t.$$

This leads us to combining these three components into a single reward function:

$$R(s_t, a) = \alpha_0 \cdot R_{\text{price}}(s_t, a) + \alpha_1 \cdot R_{\text{range}}(s_t, a) + \alpha_2 \cdot R_{\text{switch}}(s_t, a).$$

The price reward function R_{price} is a trivial penalization of expenditure. The range control function R_{range} penalizes a policy for exceeding the range $[T_{\min}, T_{\max}]$ by applying a penalty that grows exponentially as the temperature moves further outside the range. The function does not apply any penalty if the temperature is inside normal ranges, as to not impede “bold” policies that move temperatures close to the edges of the range for added performance. Lastly, R_{switch} applies a penalty if the current operational state is not the same as the next operational state, thereby rewarding non-erratic switching behaviour. We introduce parameters $\alpha_0, \alpha_1, \alpha_2$ to mediate the influence of the three individual components on the main reward function R. If parameterized properly, this reward function will reward a policy for minimizing its cost through R_{price}

while keeping temperatures in a normal range as dictated by R_{range}, and minimizing the number of operational switches through R_{switch}.

3.4 Function approximation

Second: given a state s, how do we determine which action has the highest value? Determining the action which maximizes $Q(s,a)$ can be non-trivial in stochastic environments. Stochastic environments can make it difficult to estimate the transition function between states. Without a known or estimated transition function model-based algorithms like value iteration (Bellman, 1957) will not work. One possible option is the model-free Q-learning algorithm (Watkins & Dayan, 1992). As Q-learning has no internal model of the environment it is interacting with, the algorithm will not suffer under the absence of a known transition function. Q-learning works by incrementally updating the expected value of a state-action tuple based on feedback the algorithm receives from testing those tuples against a reward function (Wiering & Otterlo , 2014). However, using Q-learning in our cold storage environment introduces a challenge. The state space of our environment is not discrete. Our state space includes variables like temperature and energy consumption, both are continuous. This means we would have to update our Q-values for an infinite number of states. One possible solution to this problem is function approximation (v. Hasselt, 2012). Function approximation has shown promise in earlier demand response research (Ruelens *et al.*, 2014) and works similar to Q-learning by incrementally updating expected values. The goal of function approximation is to learn a function Θ so that Θ approximates Q. By learning this function we can choose the action with the highest expected value:

$$\begin{aligned} Q^{\pi}(s_t,a) &= R(s_t,a,s_{t+1}) + \gamma \cdot \mathbb{E}[(\max_a Q^{\pi}(s_{t+1},a))] \\ &\approx R(s_t,a,s_{t+1}) + \gamma \cdot (\max_a \Theta(s_{t+1},a)). \end{aligned}$$

This means that after learning Θ, and observing state s_t, we can choose the action with the highest expected value according to:

$$\max_a \Theta(s_t,a).$$

3.5 Gradient boosting

Recalling our reward function, the presence of conditionals when determining the rewards shows this function is non-linear. This entails that Θ should be able to approximate non-linear functions. Our approach uses gradient boosted decision trees. A gradient boosting learner is an ensemble of multiple smaller learners, called base learners (Friedman, 2002). Base learners are fitted on a dataset before a weight is assigned to each learner based on it's accuracy. The output of the gradient boosting learner is subsequently determined by the weighted mean of the output of all base learners. Our implementation uses simple decision trees as base learners. Gradient boosted trees have shown earlier promise in highly complex, non-linear reinforcement learning environments (Abel *et al.*, 2016). When starting a reinforcement learning algorithm early actions are typically chosen at random, subject to some exploration parameter ε. This exploration parameter is gradually decreased as learning continues, making the algorithm more likely to choose the action with the highest expected value. In the case of gradient boosting this allows for bootstrapping of early base learners. However, early base learners are fitted on mostly randomly chosen actions, meaning early base learners' output might not be representative of the output of base learners that are fitted later in the process. This phenomenon, where the distribution of the input data changes over time, while it's relationship with the output does not is called coviarate shift (Sugiyama, 2011). Gradient boosting allows for attractive mitigation of covariate shift, as assigning and updating weights to base learners allows the algorithm to decrease it's dependency on early base learners as more learners are added during the learning process. Learning of Θ is done by iteratively observing a triple containing the state, the chosen action, and the reward: $< s_t, a_t, r_t >$. These triples are added to a dataset before the exploration probability is decreased by $\varepsilon_{\text{decay}}$. Every k time steps the state-action values of the dataset are computed, and b base learners are trained on the dataset. The base learners are then added to the ensemble, and all base learner weights are recomputed.

Like many machine learning algorithms, feature engineering may enhance our learner's accuracy when predicting the value of a state-action tuple. Nonetheless, little feature engineering was performed in an attempt to keep our reinforcement learning approach as general as possible. Our state s_t is given by a vector containing: the maximum and minimum temperatures $T_{\max}, T_{\min}$, the current temperature T_t, the current operational state z_t, the current energy price λ_t, and the difference between the current energy price and the energy price in one hour $\lambda_t - \lambda_{t+60}$. All of these values are easily observable.

Finally, Our implementation of gradient boosting uses an unchanged version of *scikit-learn's* gradient boosting decision tree regressor. (Pedregosa *et al.*, 2012).

4 Simulation

To learn and evaluate our gradient booster for reinforcement learning, a simulation environment was created. This simulation environment simulates temperature progression, energy use and energy cost for a single cold storage cell in discrete 1-minute time steps. Temperature progression is modelled differently depending the operational state z_t of our cell. Most cold storage facilities cool rapidly while heating by external factors is a much slower process. Additionally, temperature change starts to slow down when temperatures approache either external temperatures or the temperature of internal cooling systems. This allows us to model temperature progression as a logarithmic function of time ζ. However, our transition function should not be a function of time, but a function that returns a temperature based on the previous temperature, this is achieved through derivatives $\Delta\zeta_{\text{off}}$ and $\Delta\zeta_{\text{on}}$ (Henze & Schoenmann, 2003). These derivatives model temperature changes when the cooling system is inactive and active respectively. The functions are given by:

$$\Delta\zeta_{\text{off}}(T_t) = \frac{1}{(T_t + \phi_0)\ln(\psi_0)} + v,$$
$$\Delta\zeta_{\text{on}}(T_t) = -(T_t + \phi_1)\ln(\psi_1) + v.$$

Where ϕ_n is a parameters to control the speed of change decrease as the temperature approaches the external or cooling temperatures, and ψ_n is a parameter to control the length of a cooling or heating period. Stochastic noise is added through v, which is sampled randomly from the Gaussian distribution $\mathcal{N}(0, \mu)$, where μ dictates the variance of the noise. Combining these two derivatives completes the transition function:

$$T_{t+1} = \begin{cases} T_t + \Delta\zeta_{\text{on}}(T_t), & \text{if } a_t = 1, \\ T_t + \Delta\zeta_{\text{off}}(T_t), & \text{if } a_t = 0. \end{cases}$$

The starting temperature was set to $\frac{1}{2}(T_{\max} + T_{\min})$. Energy prices are based on the Dutch EPEX APX day-ahead market. This data is publicly available. Our simulation uses the hourly APX prices from 2016 through 2017, normalized to the interval $[0, 1]$.

5 Results

5.1 Benchmark Policy

During and after training our reinforcement learning policy was evaluated against a *benchmark policy*. This benchmark policy is an approximation of the way cold storage software manages internal temperatures. Given the temperature range $[T_{\min}, T_{\max}]$, our benchmark policy switches the operational state of the cell z_t as soon as the temperature T_t violates the temperature range. This results in a policy that switches as little possible while mostly keeping temperatures in the specified range. However, since this policy does not observe nor factor in energy prices during planning, the policies' cost is largely dependent on market patterns.

5.2 Results

To test the performance of our reinforcement learning approach, we trained our gradient booster for a duration of 100 simulated days, sampled randomly from our APX price dataset, while adding new base learners every 10 days. After each day the average discounted reward of that day, and the total energy cost as realized by the policy in that day were recorded. The progression of the energy cost and average daily rewards are depicted in Figure 1. Additionally, the figure depicts the average performance of our benchmark policy as calculated over 100 days. The comparison between the progression of our reinforcement learning policy and the average benchmark performance reveals it takes approximately 69 days before reinforcement offers any improvement.

Like the benchmark policy, we evaluated our reinforcement learning policy on 100 unique simulated days. The results are shown in Table 1. These results contain the mean discounted reward per day, the average sum of the cost per day, the average number of operational switches per day, the average percentage of the time spent cooling per day, and the average realized price per day. The realized price is calculated by dividing the cost of a day by the cooling time of that day, and should give in indication whether time spent cooling was done so during cheaper hours. All metrics include their respective standard deviation σ.

These metrics show both policies perform approximately equal in regards to our reward function, but the reinforcement learning policy achieves an improvement of 50% in cost. The improvement can largely be attributed to the 30% decrease in cooling time, but as indicated by the 9% decrease in realized price, the reinforcement learning policy does cool more during cheaper hours in comparison with the benchmark policy. An example of how the reinforcement learning policy achieves this is depicted in Figure 2. Finally we see a considerable 285% increase in the number of operational switches the reinforcement learning policy makes compared to the benchmark. An increase

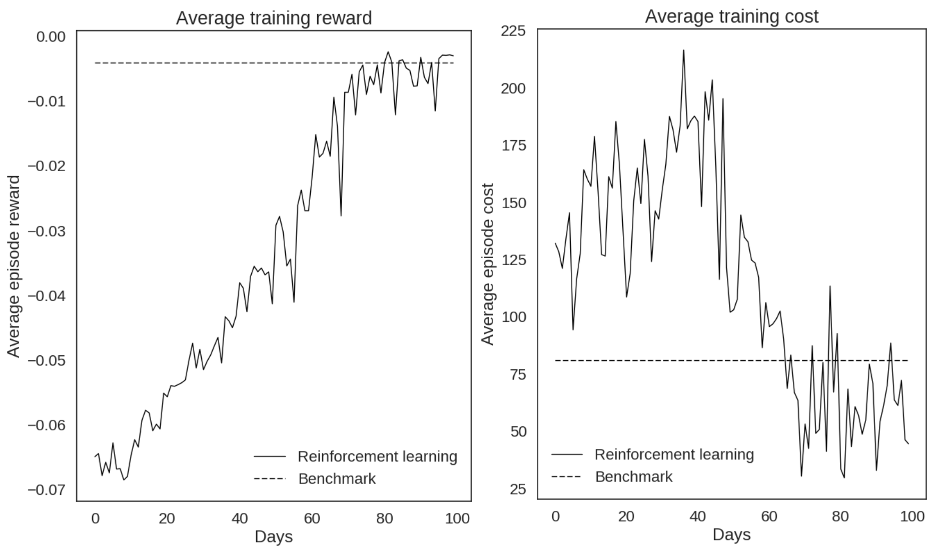

Figure 1: Average episode reward and cost during episodic learning
Figure 1 shows the averaged reward per training episode, and total realized cost per training episode. Both figures include the average performance of the benchmark policy.

Table 1: Averaged performance metrics of RL and Benchmark policies over 100 unique simulated days.

Metric	RL	Benchmark
Mean discounted reward (Q)	-0.0043 ($\sigma = 0.0023$)	-0.0041 ($\sigma = 0.0009$)
Total cost ($\sum_t c$)	37 ($\sigma = 12$)	74 ($\sigma = 23$)
Total switches ($z_t \neq z_{t+1}$)	54 ($\sigma = 26$)	14 ($\sigma = 1$)
Cooling time percentage ($z = 1$)	16% ($\sigma = 0.047\%$)	23% ($\sigma = 0.021\%$)
realized price ($\frac{\sum_t c}{\sum_t z}$)	11.12 ($\sigma = 0.026$)	12.22 ($\sigma = 0.030$)

in the number of operational switches is expected, as the benchmark policy uses the minimum amount of switches needed to keep temperatures in the specified range.

6 Discussion

Our results reveal various shortcomings of a reinforcement learning approach to demand response. Our approach takes approximately 69 simulated days before offering any improvement over the benchmark policy. This 69 day period can be a long time to bridge, as the exploratory phases of learning exhibit undesirable behaviour like erratic switching or violation of temperature ranges. Possible solutions to this problem include bootstrapping a number of base learners on historical data of the cold storage system or fitting various linear models on the temperature progression patterns of the system to build an approximate model of the system, before learning a number of base learners from this approximate model. Both of these solutions offer possible venues for future research. Additionally, our simulation is based on one instance of a cold storage systems, but like any system, cold storage systems exist in a wide array of configurations.

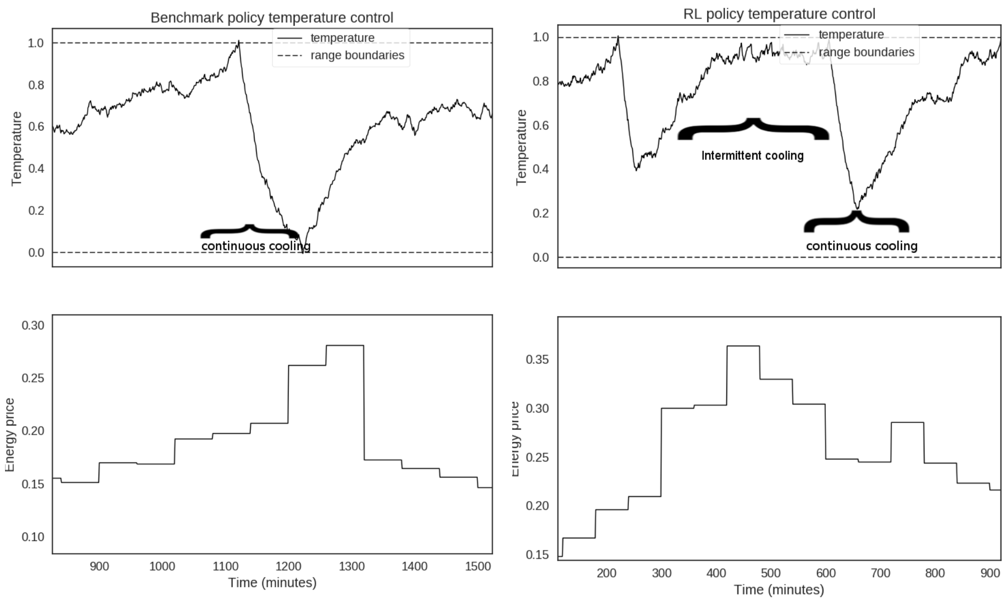

Figure 2: Example control sequences of benchmark and RL policies
Figure 2 Shows examples of a control sequence using the benchmark policy. The benchmark policy does not factor in energy pricing and starts to cool continuously during a peak in energy price. Additionally this figure shows an example of a control sequence as executed by the reinforcement learning policy. While scarcely cooling during high energy prices, the RL policy succeeds in keeping temperatures within the specified range. Once energy prices decrease the RL policy starts cooling continuously and thereby manages to avoid cooling during another small price peak.

These configurations range from systems consisting of a single cell to systems consisting of dozens of cells with unique temperature ranges. Even though our reinforcement learning was kept as general as possible, more work is required to examine the effect of the configuration of a cold storage system on the viability of reinforcement learning for that system. For configurations consisting of multiple cells possible research could be directed towards batch reinforcement learning, as used by Ruelens *et al.* (2014).

Furthermore our results pose a dichotomy: most of the savings achieved through reinforcement learning can be attributed to a reduction in the time spent cooling. This means that, regardless of energy prices, switching as little as possible is not necessarily the best course of action. This observation is a result of the logarithmic pattern that the temperature progression follows. Since this progression is logarithmic, the effort needed to keep the temperature decrease constant grows exponentially as more time is spent cooling. At a certain point, the effort needed to continue cooling may come at an undesirable cost, even if the temperature is far above the lower temperature boundary. Deciding how long to cool, assuming a constant energy price, is a problem beyond the scope of this paper, but the results of our experiments show that reinforcement learning may be applicable to solve this problem, while simultaneously performing demand response.

Our main criticism of earlier work revolved around the lack of generalizability. Here our results show it is possible to achieve some level of demand response without the need for differentiable reward functions (O'Neill *et al.*, 2015) or heavy feature engineering (Ruelens *et al.*, 2015). However, our generalized approach seems to require a longer learning period, which may be undesirable. Ultimately our approach exhibits shortcomings for demand response. Nonetheless, our results show it is possible to achieve considerable cost savings in a specific cold storage system using a general reinforcement learning approach, thereby offering a model-free method that is readily available and easily implementable with limited domain knowledge.

7 References

Abel, D., Agarwal, A., Diaz, F., Krishnamurthy, A., & Schapire, R. E. (2016). *Exploratory Gradient Boosting for Reinforcement Learning in Complex Domains.* Preprint.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, Mass. MIT Press.

Bellman, R. (1957). A Markovian Decision Process. *Indiana University Mathematics Journal*, 6(4), 679-684. doi:10.1512/iumj.1957.6.56038

Henze, G., & Schoenmann, J. (2003). Evaluation of Reinforcement Learning Control for Thermal Energy Storage Systems. *HVAC&R Research*, 9(3), 259-275. doi:10.1080/10789669.2003.10391069

Cigler, J., Gyalistras, D., Široký, J., Tiet, V. & Ferkl, L. (2013) Beyond theory: The challenge of implementing Model Predictive Control in buildings. *2013 Proceedings of the 11th REHVA World Congress (CLIMA).*

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4), 367-378. doi:10.1016/s0167-9473(01)00065-2

Van Hasselt, H. (2012). Reinforcement Learning in Continuous State and Action Spaces. *Adaptation, Learning, and Optimization*, 207-251. doi:10.1007/978-3-642-27645-37

O'Neill, D., Levorato, M., Goldsmith, A., & Mitra, U. (2010). Residential Demand Response Using Reinforcement Learning. *2010 First IEEE International Conference on Smart Grid Communications*. doi:10.1109/smartgrid.2010.5622078

Pedregosa *et al.* (2012). Scikit-learn: Machine Learning in Python, *JMLR*, 12, 2825-2830,

Ruelens, F., Claessens, B. J., Vandael, S., Iacovella, S., Vingerhoets, P., & Belmans, R. (2014). Demand Response of a heterogeneous Cluster of electric Water Heaters using Batch Reinforcement Learning. 2014 *Power Systems Computation Conference*. doi:10.1109/pscc.2014.7038106

Ruelens, F., Iacovella, S., Claessens, B., & Belmans, R. (2015). Learning Agent for a Heat-Pump Thermostat with a Set-Back Strategy Using Model-Free Reinforcement

Learning. *Energies*, 8(8), 8300-8318. doi:10.3390/en8088300

Sugiyama, M. (2011). Learning Under Non-stationarity: Covariate Shift Adaptation by Importance Weighting. *Handbook of Computational Statistics*, 927-952. doi:10.1007/978-3-642-21551-3_31

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4), 279-292. doi:10.1007/bf00992698

Wiering, M., & Otterlo, M. (2014). *Reinforcement Learning: State-of-the-Art.* Berlin: Springer Berlin.

8 Appendix A - parameters

Param. Summary	Param.	Value
Reinforcement learning		
Discount factor	γ	0.9
Base learner interval	k	10 days
Base learner additions	b	20
Starting exploration	ε	0.99
Exploration decay	$\varepsilon_{\text{decay}}$	1×10^{-6}
Reward function		
Range penalty growth	η	4
Price reward weight	α_0	0.0100
Range reward weight	α_1	0.0100
Switch reward weight	α_2	0.0125
Simulation		
Range min.	T_{min}	0
Range max.	T_{max}	1
Power consumption	ρ	1
Heating speed decrease	ϕ_0	0.1
Cooling speed decrease	ϕ_1	0.1
heating period length	ψ_1	1.00185
Cooling period length	ψ_1	1.02200
Noise deviation	μ	0.05