Multi-Advisor Deep Reinforcement Learning for Smart Home Energy Control

Andrew Tittaferrante

April 2021

Abstract

Effective automated smart home control is essential for smart-grid enabled approaches to demand response, named in the literature as automated demand response. At it's heart, this is a multi-objective adaptive control problem because it requires balancing an appliance's primary objective with demand-response motivated objectives. This control problem is difficult due to the scale and heterogeneity of appliances as well as the time-varying nature of both dynamics and consumer preferences. Computational considerations further limit the types of acceptable algorithms to apply to the problem. We propose approaching the problem under the multi-objective reinforcement learning framework. We suggest a multi-agent multi-advisor reinforcement learning system to handle the consumer's time-varying preferences across objectives. We design some simulations to produce preliminary results on the nature of user preferences and the feasibility of multi-advisor reinforcement learning. Further smarthome simulations are designed to demonstrate the linear scalability of the algorithm with respect to both number of agents and number of objectives. We demonstrate the algorithms performance in simulation against a comparable centrallized and decentrallized controller. Finally, we identify the need for stronger performance measures for a system of this type by considering the effect on agents of newly selected preferences.

Acknowledgements

I am grateful for the support and guidance provided by my supervisor, Dr. Abdulsalam Yassine. His continuing guidance provided clear direction for this work, while simultaneously allowing me the freedom to research and innovate to my heart's desire. His unique perspectives always provided clarity on where to best channel my efforts. His unwavering faith in me enabled me to continually be looking forward in some of the most difficult times I've experienced and for this I am truly grateful.

For financial support, I'd like to thank Lakehead University and the department of Engineering for providing a graduate assistanceship position and entrance scholarship. I also am grateful for being given lab-space to complete my work, this provided immeasurable improvements to productivity in my time spent there. Finally, I'm thankful for being given the opportunity to teach, which provided extremely valuable experience and insight into the perspective of my own professors.

I'd like to thank my friends and lab-mates in ATAC4013. My kindhearted lab-mates were all extremely welcoming when I arrived, and continued to be throughout my time in the lab. The general advice, support, assistance, inspiration and distraction from my colleagues was always welcome. The technical ideas discussed on the whiteboard have greatly helped shape both this work and my future trajectory, and have greatly helped sharpen my own technical prowess. I miss those of you who have left and hope to return soon.

Contents

Chapter 1

Introduction

1.1 Introduction

Electricity companies face an operational inefficiency due to consumer energy consumption patterns [45]. Consumption trends exist for consumers in relation to time of day, which leads to energy demand peaks. This causes a problem for electricity generation companies, which are required to generate more energy during these peaks. Due to this increased demand for electricity, some operational efficiencies of power production are decreased, leading to a higher cost per. unit of energy. To curb this inefficiency, utility companies are adopting the concept of Demand Response. In Ontario, the current demand response scheme divides a day into three different pricing periods, On-peak, Mid-peak, and Off-peak. This is a simple scheme, where prices are higher On-peak, and lower Off-peak and Mid-peak. In this scheme, the prices are adjusted twice yearly[18] [36].

Two possible improvements to Ontario's scheme, or those like it, could be to a) Divide the day into more than three time-slots, and b) Adjust the prices more frequently than twice yearly. The main scheme considered in this work is one where electricity has a price which is specified for every hour. The hourly pricing for a day is known before the day starts. The scheme may be ineffective, because while it more precisely motivates the consumer to control their energy consumption, it is also inconvenient to them; the consumer would need to adjust their appliance settings by the hour to effectively account for the varying price [63].

One approach to increase the feasibility of precise consumer motivation signals such as hourly time-of-use pricing (TOU) for residential consumers is that of Automated Demand Response [43] (ADR), where the consumers smart-home appliances automatically adjust their control policy to balance the

compensation from the utility company with the appliance's primary objective. The ADR problem is thus a control problem, where the controlled devices have some primary objective (the original purpose of the appliance) they are trying to achieve, which then must be augmented by a secondary demand response objective. A concrete and prevalent example of this is assigning a thermostat to automatically adjust a heater's energy consumption to balance its primary objective of set-point tracking (maintaining user's desired indoor temperature) with the secondary objective of reducing energy consumption.

The previously described example hints at what we believe is the inherent multi-objective nature of the problem. The thermostat in a DR setting clearly has at least 2 objectives, set-point control and consumption minimization. Therefore, we assume that an effective ADR control scheme must be able to handle multiple objectives. Furthermore, it seems necessary that a device be capable of adapting to previously unspecified objectives, as capabilities of devices may evolve. It seems reasonable that issues of cooperation may occur unexpectedly when a new device is added. It may be possible to inspire cooperation by considering cooperation as an objective for the agent to pursue.

In this work we make use of the recent advances in reinforcement learning (RL)[47]. The standard RL problem deals with training an agent to act optimally in a Markov Decision Process (MDP). The trained agent can be viewed as implementing a learned policy, a function assigning actions for any given state the agent finds itself in. The desired policy should chose actions as to maximize expected return. A common approach to learn a policy is through the use of action-value functions, with recent implementations such as neural-network based deep Q-networks (DQN) [26]. Specifically, we utilize Multi-Advisor Reinforcement Learning (MAd-RL) [20], which is an approach to solving a single-objective RL problem by decomposing the reward signal into multiple separate rewards and separately learning an action value function for each reward. Actions are then selected by aggregating these action-value functions into a specific action.

1.2 Motivation

Recently, there was been a growth in smart-home technologies available for purchase by consumers. When considering that smarthome appliances can be purchased individually, it becomes apparent that a system composed of these appliances must be capable of growth; a thermostat might need to cooperate with a dishwasher purchased at some future date. This is currently notable when it comes to energy scheduling in demand response; two devices will have to coordinate in some way to avoid overtaking a smart-contract's consumption threshold.

In terms of growth capabilities, along with the previously described decentralized/cooperation as-

pect, a further complication is the time-varying nature of the user's opinions and objectives. For example, a consumer may decide to take a more active role in the battle against climate change and request that their smarthome devices adjust their behaviour to reduce energy consumption. After two weeks, the consumer decides that they dislike the cold indoor temperatures, and requests a more moderate reduction in energy consumption. In essence, this requires that the device may seamlessly begin making decisions based on a brand-new objective that wasn't previously considered, and make appropriate trade-offs between these objectives. This capability would also encompass new/unpredictable objectives which may arise in the future, a powerful capability to offer in a smart device. Furthermore, it seems possible to mend cooperation issues between devices by explicitly defining clear cooperative objectives for the appliances.

Associated with nearly every control task is the notion of feedback, where some sensor provides information about the environment to help the controller estimate the state of the system. In a home that gradually becomes 'smart', appliances will be added over time, and these may require sensors be added. This is clear in the task of temperature control; a user may desire a temperature sensor be added to different rooms to control different temperatures. Commercial products are available with similar capabilities, for example [31].

These ideas motivate the work in this thesis. The motivation comes from a desire for a smart-home control system which is capable of seamless growth while incorporating newly included appliances, objectives, and sensors. This is a non-trivial task which requires algorithms that perform effectively and are scalable in terms of objectives, appliances and sensors.

1.3 Technical Challenges

A first basic challenge is the that of versatility. Essentially, the conditions in each smart home will be vastly different. A small house in Thunder Bay interacts with the environment that is drastically different than a large house in Dubai. Another challenging aspect is the typical seasonal changes in the environment, the controller must be adaptable to handle these. Furthermore, it is near essential that a controller be capable of generalization, as the space of observations is much to wide to explore effectively. There is also difficulties with the user's preferences, which generally change over time. This represents a change in objective structure and control algorithms need to be selected that can adapt to these changes too.

Some main technical challenges of this work revolves around the heterogenenous nature of appliances and objectives. Normally, each appliance is designed to achieve some primary objective such as

maintaining indoor temperature, charging an electric vehicle, or cleaning laundry. Each of these has a fundamentally heterogeneous nature, which makes comparing performance of each task difficult. This is further exasperated by the inclusion of new objectives, eg. consumption reduction or consumption thresholds. In this setting each device will need to make trade-offs between these objectives, and an algorithm for achieving such is required.

A further challenge results from the heterogeneous nature of appliances. Each device faces a unique task; the problem of indoor heating requires the controller minimize the temperature's distance from ideal, while lighting needs to keep consistent visibility in the residence. Mathematical models of these two situations would be unique, the first could use differential equations, the second would use relatively high-dimensional presence and lighting variables utilizing discrete logic-based reasoning [1]. Control across a variety of tasks using a uniform standard requires extremely versatile control algorithms. While standard linear controllers (eg. Proportional Integral Derivative (PID) controllers) are fantastic in situations that can be modelled with linear systems, they could not easily be applied to automated lighting.

The high-dimensional action and observation space of a futuristic smart-home comes with a high degree of computational complexity. In theory of dynamic programming for MDPs, this is referred to as the "curse of dimensionality", referring to the exponential growth of a Q-table size w.r.t. the dimensionality of both action and observation spaces. This nearly necessitates the controlling agent be organized in a decentralized fashion, which requires efficient algorithms to be applied in the multi-agent case.

Finally, scalability is of fundamental concern in design of computationally expensive systems. This is noted previously as concern with the "curse of dimensionality" but also includes consideration for the control algorithm itself. For example, model-predictive control approaches reference a model to plan the next step of their actions, but these simulations can take time to perform with complex models. In a neural-net based controller, performing the forwards and backwards passes through the network take considerable time, requiring matrix multiplications which take polynomial time to complete. Further complexity in this system includes the number of agents considered, as well as the number of objectives considered by each agent.

[1] For example, the XOR operation would be noted, as the user is exclusively in one room at a time

1.4 Research Approach

This research aims to tackle the technical challenges as described previously. This begins with a look at optimization methods, as they provide an abstract view of the smart home problem, complete with models of the environment and the devices. Under these schemes, abstract models of appliance classes are available in the literature. Given a reliable model of the appliance/environment interaction, these approaches can be extremely effective, and form a major branch of control approaches in smart-homes, commonly named "model predictive control". Under this framework, the literature demonstrated that user preferences are normally accounted for taking a linear combination of the basic objectives. This provides a means of mathematically modeling user preferences given utility functions of specific objectives. Furthermore, this work provided a starting point for considering models of specific aspects, particularly time of use pricing and models of electric heating.

While the optimization techniques are useful and commonly efficient, there are some core reliability issues. Particularly, when the model isn't correct, there is a disconnect between what the controller expects to happen and what actually happens. This can be better accounted for by considering stochastic models, such as Markov Decision Processes. These models are generally more difficult to design controllers for, but dynamic programming techniques have been known to be effective for relatively low-dimension observation spaces.

Deep neural networks are capable of generalization and handling of high-dimensional data, and thus seems a strong candidate for smart home control. The approximation provided by the network allows the system to reduce the problem complexity. The learning aspect allows the network to adapt to changes. Furthermore, deep learning is known to be effective in cases with high-dimensional inputs, enabling strategies like temporal conditioning, where the season / month / day of week are provided as input to the network. This enables more adaptability with the natural seasonal changes of the environment. Furthermore, neural networks are known to scale to large data sets, which are prevalent in smart environments.

A natural concern when considering utility functions is that of their source; defining a utility function for a task is generally not a simple task. In this sense, we are motivated to approximate the cumulative utility function using an action-value function, prominent in reinforcement learning. This motivates the use of reinforcement learning for smart home control, and is supported by the plethora of research applying reinforcement learning to smart home tasks. Reinforcement learning approaches are commonly described as model-free, as they do not need to deliberately model the environment dynamics

to be able to act, and can therefore be implemented regardless of model accuracy. There is a catch to this claim, as currently reinforcement learning techniques require many failed interactions with the environment, and thus take too long to learn. Nonetheless, given a model, a reinforcement learning algorithm can be run in the model to gain experience. If future algorithms in reinforcement learning can increase data-efficiency to improve the learning time, then they can be described as truly model-free algorithms.

The idea of approximating cumulative utility functions as action-value functions falls under the approach named "multi-advisor reinforcement learning" [20]. It is a simple strategy that has been shown to converge when using off-policy action-value based algorithms with a fixed aggregation function. The hybrid reward architecture [55] appears to be an implementation of multi-advisor reinforcement learning, and has demonstrated performance in the specific game of Mrs. Pacman, an Atari game of relative difficulty for DQNs.

To evaluate these techniques, we develop a series of mathematical models designed to be appropriate to demonstrate the functionality of the algorithms in specific aspects of the smart home environment. This begins with a simple convex optimization problem model based on the work of [39], which simulates multiple appliances and considers the objectives for each device as well as energy cost. Explored next is a simple deterministic single-agent MDP model for an electric heating problem. This model uses energy reduction as a secondary objective, and tests the behaviour of MAd-RL under scheduled user preferences. Finally, a more complex MDP which models both electric heating and electric vehicle charging is developed. This model includes a cooperative objective which subjects the system to concerns associated with multi-agent coordination. Furthermore, the electric vehicle charging dynamics are stochastic, providing more robust evaluation of the algorithm. Finally the algorithm is simulated to determine the scalability of its operation w.r.t. to the number of objectives, as well as the number of agents.

1.5 Contributions

- Analyzed and identified "importance scaling", a weighting mechanism for which a consumer's preferences between distinct utility functions can be effectively represented. This work can be found in chapter 4, based on [51], where a convex optimization problem with known utility functions is used to demonstrate a qualitative performance improvement which arises by considering varying importance weights for the system.

- Formulated the smart-home control problem for the first time[2] as a multi-objective reinforcement learning problem. This is a more-or-less obvious perspective, as the demand response optimization approaches in the field all inherently consider multiple objectives, and other work in reinforcement learning for smart homes does as well. Nonetheless, the multi-objective structure had yet to be harnessed by a dedicated reinforcement learning algorithm.

- Proposed multi-advisor reinforcement learning as an algorithm to grapple the growing multi-objective nature of the problem. The algorithm allows the inclusion or exclusion of multiple objectives as seen fit by the aggregation function. This enables the aggregation function to act as the importance weights for the system. If a new objective is desired another Q-function may simply be added/learned, followed by adjusting the weights to allow it to affect decision making.

- Demonstrated that state-based importance weighting as an aggregation function can enable the shifts in behaviour typically desired by smart home owners. This work is presented in chapter 5, based on [52], and is demonstrated by the behaviour that results from assigning a low importance during mid-afternoon for the task. This shows that the algorithm will learn to shift between tasks according to the scheduled weight scheme. These results are based on simulation of a simple first-order deterministic thermal model.

- Formulated a MDP model for a smart home which considers 2 appliances and 4 unique objectives. The model continues with the previously used thermal model, and further includes dynamics and rewards for an electric vehicle charging situation, which are inspired by the work of [57]. Furthermore, this includes a new MDP formulation for a smart contract, which attempts to resolve the credit assignment problem by providing punishment to the agent proportional to their contribution to failure.

- Demonstrated the performance of the proposed multi-agent multi-advisor reinforcement learning agent in the previously described environment. The proposed agent is compared with 2 similar DQN-based algorithms, one of which is centralized, the other decentralized. The metric for comparison is that of cumulative weighted reward, in which the 3 agents each exhibit comparable performance. This indicates the potential of the proposed algorithm to achieve competitive results. The results indicate that the proposed agent takes a unique strategy compared to the benchmark agents.

[2]The work of [15] claims this as well, but our publication on [52] predates their claim.

- Empirical and theoretical analysis of the algorithm scalability. The measured task is the operations which must be performed between each discrete time-step. These results can also be found in chapter 6. It is shown that the algorithm scales linearly with both objectives and agents. The theoretical results follow from considering the number of forward passes through a neural network in order to select an action, while the empirical results involve instantiating the agents with smaller networks, but varying number of agents/objectives. The empirical data measures the "learn" operation, which encapsulates both action selection and neural network training.

1.6 Organization

This thesis is organized as follows:

- Chapter 1: Introduction - This chapter, provides a high-level description of the thesis and the various aspects involved.
- Chapter 2: Background - Brief descriptions of fundamental ideas which are essential building blocks to understanding the work in this thesis. Describes basic aspects of demand response, optimization, and reinforcement learning.
- Chapter 3: Related Work - Provides brief summaries of related work in the relevant fields. This primarily includes a look at the smart-home control approaches as prevalent in the literature, follows up with a brief sample of some recent reinforcement learning algorithms. A brief summary of the sampled work is provided at the end.
- Chapter 4: Importance Scaling of Elastic Appliance Utility Functions for Automated Power Management in Smart Homes - This is the work from [51], where a convex optimization problem is formulated and solved to demonstrate the effect of considering variable objective importance from a smart-home perspective.
- Chapter 5: Multi-Advisor Reinforcement Learning for Residential heating - This work is from [52], where a multi-advisor agent is simulated to control an electric heating environment. This work demonstrates that state-based weighting can achieve desirable performance.
- Chapter 6: Multi-Agent Smarthome Control with Multi-Advisor Reinforcement Learning - This work describes the MDP formulation for a smarthome with 2 appliances, presents the proof of scalability, and shows the results from simulation, including comparison against some benchmark agents.

- Chapter 7: Conclusion and Future Work - This chapter describes the overall conclusions to be made from this work's results, and presents some aspects which could be interesting to expand on in future work on this topic.

Chapter 2

Background

2.1 Demand Response

Demand response (DR) is motivated by utility distribution companies' desire to decrease peaks in energy demand caused by common consumer energy consumption schedules. Demand Response is defined as "The changes in electric usage by end-use customers from their normal consumption patterns in response to changes in the price of electricity over time"[1]. This concept is useful because electricity companies face operational inefficiencies in their energy consumption patterns [45], and it generally describes the techniques used by electricity companies to incentivize energy consumers to adjust their consumption habits, ie. financial incentives. This can include approaches such as Time-of-Use pricing, where at every time-step during the day, a specific price (eg. in $/kwh) is set by the utility company. Another scheme is that of "Direct Load Control", where the utility company forms an arrangement with the consumer, whereby the utility company can turn off the consumers devices remotely, and in return the consumer is financially compensated. A third approach is that of "smart contracts", where the utility company offers the consumer a financial compensation to reduce their energy consumption below some predefined threshold during a specific time-slot, with some financial penalty associated with overtaking the threshold [21]. These techniques all provide some mechanism for the utility company to reduce costly peaks in energy demands, and thereby reduce operational costs. This work will focus primarily on Time-of-use pricing and smart contracts, as they decentralize the consumption adjustments away from the utility company, providing more freedom for the consumer.

2.2 Optimization

In mathematical optimization, the standard problem is given in a form similar to:

$$
\begin{aligned}
& min_x \ f(x) \\
& s.t. \quad g_i(x) \leq 0, \quad \forall i \in \{1, \ldots, n\}
\end{aligned}
\tag{2.1}
$$

which describes selecting a vector of variables x that minimizes the objective function $f(x)$ while obeying the constraints specified by a set of n inequalities, defined by the functions $g_i(x)$. The set of vectors x that satisfy all of the constraints is called the feasible set. This covers a wide variety of optimization problems. Notable common classes of optimization include linear, convex, nonlinear, and multi-objective optimization, each described briefly in the following sections.

2.2.1 Common Problem Classes

Linear Optimization

Linear Optimization [56], also called Linear Programming, has problem structure as shown in equation 2.2. For the problem to be considered a linear optimization problem, the constraint and objective functions need to be linear w.r.t. the decision variable x. Linear programming problems are usually considered simple to solve, with fast performing algorithms such as the simplex method [9] used in practice.

Convex Optimization

Convex Optimization [5] is another category of simple optimization problems. There are polynomial-time algorithms available to solve many of these problems [5]. These problems again follow the same form as given in equation 2.2, but restricts that the objective function must be convex w.r.t. x, and all feasible solutions must belong to a convex set. The feasible set is normally defined using epigraphs of convex functions along with an affine set. A convex optimization problem can be represented as:

$$
\begin{aligned}
& min_x \ f(x) \\
& s.t. \quad g_i(x) \leq 0, \quad \forall i \in \{1, \ldots, n\} \\
& \qquad Ax = b
\end{aligned}
\tag{2.2}
$$

Nonlinear Optimization

Nonlinear optimization covers a broad class of optimization problems, including those which have no guaranteed linear or convex structure on the objectives or constraints. These problems are not easily

solved. Often it is acceptable to produce a locally optimal solution to a nonlinear optimization problem, as globally optimal solutions can be intractable.

Multiobjective Optimization

Multiobjective optimization problems follow the standard optimization problem form, but is fundamentally unique due to the vector nature of the objective function. This is a notably complicating feature, as comparison between vectors are ill-defined. In single-dimensional objective functions, the comparison between two feasible solutions has a clear outcome, either one solution is superior, or they tie. In the multi-objective case, there is only a partial ordering.

For example, consider a maximization problem which has 2 feasible solutions: $[1, 0]$ and $[0, 1]$. These 2 solutions cannot be generally compared, as each is in some sense both better and worse than the other. These vectors can be mathematically described as 'incomparable'. Notably, if third and fourth solutions of $[1, 1]$ and $[0, 0]$ were also available, it becomes clear that $[1, 1]$ is the best, and $[0, 0]$ is the worst. A fifth point of $[2, 0]$ would again mean that there are 2 optimal solutions, this time $[2, 0]$ and $[1, 1]$. These 5 points illustrate the idea of partial ordering, where a vector can be greater, less than, equivalent, or incomparable to any other vector it is compared to.

This property of performing comparisons between vectors leads to a critical aspect of multi-objective optimization problems; there may be multiple incomparable solutions to the same optimization problem. This induces the idea of maximal/minimal solutions (different from the commonly used maximum / minimum), which describe solutions who are either 'incomparable to' or 'better than' all other solutions.

2.3 Reinforcement learning

Reinforcement Learning [47] refers to a set of techniques used to train an agent to interact with an environment. The environment is modelled as a Markov Decision Process. The usual diagram from [47] can be seen in figure2.1. This standard picture describes the interaction between the agent and environment. Due to the discrete nature of computers, the interaction generally follows discrete time-steps, denoted with a subscript t. During a single time-step, the agent sees the reward from the previous time-step and inputs the current state. The agent next selects an action (which is input to the environment/actuators). The environment then responds to the action and produces a next state and reward for the transition [1]. The goal of Reinforcement Learning is to determine a policy, usually

[1] The sequence of events occurs like $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots)$

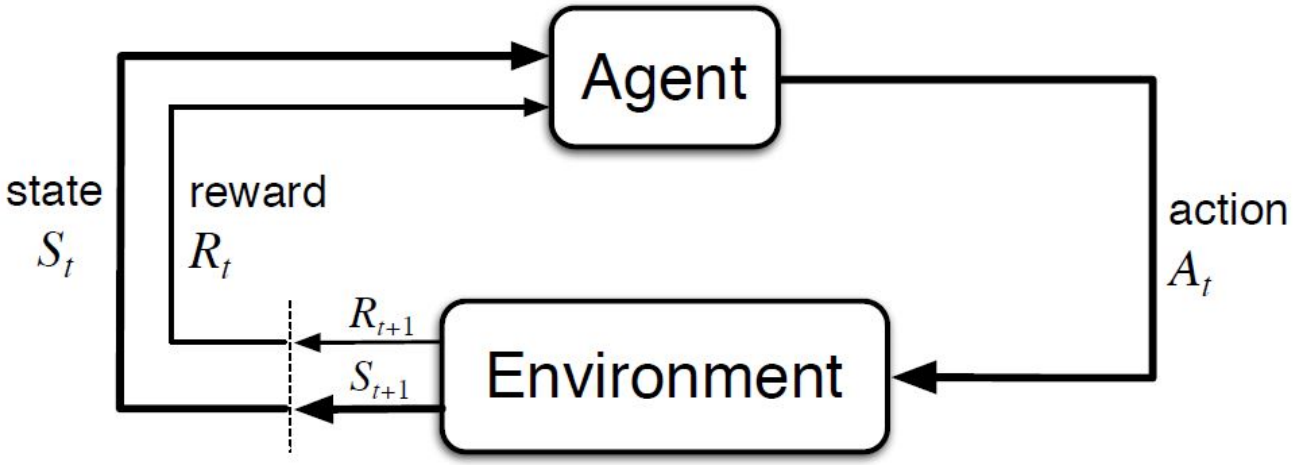


Figure 2.1: Standard Agent/Environment interaction diagram, taken from [47].

denoted π, that prescribes an action for any given state [2], ie:

$$\pi(s) : S \mapsto A \tag{2.3}$$

The policy desired is one that maximizes some sense of the reward (eg. average, overall, discounted) achieved by following it.

2.3.1 Markvov Decision Processes

A Markov Decision Process (MDP) [4] can be described as having 5 elements, (S, A, R, T, γ). The first element, S, represents the set of all possible states, which can be analogously interpreted as being the set of situations the agent can find itself in at a moment. The second element, 'A', is the set of all actions the agent can take (eg. the voltage applied to a motor). 'R' is the reward function, which normally can be described as a function $R(s, a, s')$[3] which provides a scalar value for every transition that can occur in the environment. The agent is concerned with acting to maximize the signal 'R'. 'T' describes the agent's (stochastic) transition dynamics, ie. the probability of arriving in a given state, s_{t+1}, when in an arbitrary state s_t after taking some action a_t.

State in MDP

The concept of state in a MDP is related to that of state in deterministic control systems. The common control textbook [33] describes the state as being the minimal set of variables necessary to determine

[2] Commonly, a stochastic policy is desired instead. In this case, the function mapping is $\pi(a \mid s) : S \times A \times S \mapsto R$

[3] Note that s' denotes the 'next state', as in the state which occurs after s

the new state that occurs at some later time, as long as the inputs are known for the intermediate period.[4] This notion necessitates that prior states are irrelevant, as in the definition only the current state and an arbitrary length sequence of inputs are required to determine the future state. In his 2019 lectures at Arizona State University, Dr. Bertsekas describes state as "something that separates the past from the future" [53]. This idea is clear because generally in sequential decision-making, the state dynamics are a function of the entire history, H_t, the record of all observation-action sequences since the beginning:

$$H_t = \{O_\tau, A_\tau | \tau = 1, \ldots, t\} \tag{2.4}$$

This is theoretically sound, although often not helpful due to the continual dimensionality increase of the history. As following Dr. Bertsekas' intuitive description, the state provides a (normally-fixed length) representation of the history as it is necessary for determining future states. In the stochastic case of Markov Chains, upcoming states are unknown but follow some distribution. Therefore the state need not be sufficient to determine the next state, but rather to determine the distribution over next states. Mathematically, this describes the Markov Property, ie:

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|H_t) \tag{2.5}$$

This requirement in the specification of the dynamics is seemingly complex, as it requires the agent designer to specify a means to represent all the history in a reasonable manner. This difficulty is commonly resolved by considering the previous set of n observations, which intuitively relates to the task of approximating a k-order derivative using the previous n observations. Alternatively, recent work has approached this problem using recurrent neural networks which only input observations [17].

2.3.2 Q-learning

Q-learning [58] was one of the first, and is one of the most popular Reinforcement Learning Algorithms. The algorithm explores the environment, and iteratively learns to estimate the action-value function of the policy being used to explore. The action value function, denoted $Q(s, a)$ is defined as:

$$Q(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a] \tag{2.6}$$

[4] Exact wording from text: "The state of a dynamic system is the smallest set of variables (called *state variables*) such that knowledge of these variables at $t = t_0$, together with knowledge of the input for $t \geq t_0$, completely determines the behavior of the system for any time $t \geq t_0$"

$Q(s,a)$	a_1	a_2	...	a_m
s_1				
s_2				
...				
s_n				

Figure 2.2: Visualization of a table of Action-value functions, considering n states and m actions

This function is defined to be the average sum of rewards succeeded by the current state, given that policy π is being followed to select actions in all succeeding states. The coefficient, γ, is called the discount factor, which provides theoretical convergence guarantees (by bounding the sum), and usually is explained as accounting for the fact that near rewards are more valuable that distant rewards.

The action value function is represented (stored in computer memory) as a table, see fig 2.2, where each row represents a single state, and each column represents a single action. As a greedy Q-learning agent observes a state, it checks the corresponding row for that state, and selects the action (column) with the highest number (the action-value of that state-action pair), as it is expected to give the largest return.

The algorithm progresses roughly as follows; the values in the table are all initialized to some default values (often zero, but initial values could be carefully chosen to guide exploration through the environment). The agent then selects actions according to some exploration-exploitation scheme, for example ϵ-greedy, where the agent selects the highest-value action with probability ϵ and a uniformly random action with probability $1 - \epsilon$. The actions cause the environment to evolve, and the agent observes a new state and some reward for the transition. The agent then makes an update on the entry corresponding to the observed state/action pair using the following update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \tag{2.7}$$

or equivalently,

$$Q(S_t, A_t) \leftarrow (1 - \alpha)Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')] \tag{2.8}$$

Here α is called the learning rate, and the factor multiplied by it in 2.7 represents the error between the value and the sample. The form in 2.8 makes it clear that this brings the estimated Q-value away from it's current estimate and towards the new sample estimate.

The total number of elements in this table is the Cartesian Product of the action space and the state space. This means that the table size is exponential in terms of both the action space and the state space, and therefore is not suitable for large-scale problems. In the case of fig. 2.2, the total number is $m \times n$ values, where m and n will grow exponentially w.r.t. their own corresponding dimensionality.

2.3.3 Deep Reinforcement Learning

Deep Reinforcement Learning [12] is a recently growing approach to Reinforcement Learning. The approach can be described as utilizing the generalization power of neural networks to learn to act in relatively complex environments. This is demonstrated by the DQN in [26] surpassing the performance of a professional games tester in a wide array of ATARI games. These ATARI games, while not truly representing a real-world environment, serve as a benchmark for the capabilities of such a system to learn to act in a complex domain.

There are many deep reinforcement learning algorithms being researched and utilized today. Primarily, these algorithms fall under 2 main categories; policy-gradient methods and value-based methods. Intuitively, the two methods both use a neural network to approximate some function. In policy-gradient methods, the network parameterizes the policy directly, while in value-based methods, the network parameterizes the action-value function. Two of the most popular algorithms of each type are DDPG and DQN respectively. Notably, these sorts of algorithms can be used together for more efficient actor-critic algorithms.

2.3.4 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning considers the case of multi-dimensional actions, each of which is selected independently by it's own agent. The commonly used dynamics model for this situation is that of Stochastic games [44], an extension of MDPs to the case where there are multiple agents, and therefore multiple reward functions. In the purely cooperative case, all agents share the same reward function and the problem framework is that of decentrallized partially observable MDPs (Dec-POMDPs)[34].

Two primary concerns of this problem setting are that of optimality (related to that of previously described multiobjective optimization), and secondly that of credit assignment [6]. Credit assignment

is a prevalent problem in fully cooperative environments. In this case, as all agents share a reward function, it is difficult for any single agent to determine how much it's own action contributed to the outcome. This makes it difficult for the agent to learn whether it's own actions were good or bad. In some cases, as in this work, individual reward functions can be applied to help aim towards centralized goals.

Optimality is of concern first due to the vector nature of the performance measure. The performance measure of a policy is the value of the initial state. In the multi-agent case, this initial value becomes a vector and indicates that there are multiple joint policies which all provide maximal value. Furthermore, when policy-selection in this setting is decentralized, the concept of Nash Equilibrium arises [30]. If all but agent i has a stable policy, then agent i will select it's best response given the other agent's policies. This issue is apparent when considering the prisoner's dilemma. In the prisoner's dilemma, the best response is always to betray your partner. Therefore, each agent should select the betray action, and this outcome is clearly not optimal in a multi-objective sense.

A simple, suboptimal, and common approach to multi-agent reinforcement learning is that of Independent Q-learning [50], where each agent is trained using the single-agent Q-learning algorithm. This approach has been shown to be surprisingly effective in empirical evaluation performed in [64]. This approach is used in the work of this thesis.

Chapter 3

Related Work

The open literature includes a plethora of work in various research on demand and energy management for residential appliances. Although the review of these studies in this section is not exhaustive, it provides a representative view of related work. This chapter begins by describing the work of various smart-home control approaches considered in modern research. These mainly include optimization and reinforcement learning based schemes, both of which imply the need of dynamics and reward models for the system. These are followed by a brief sample of modern reinforcement learning algorithms, particularly focused on approaches based on neural networks. These are briefly described due to their relevant nature as possible approaches in regards to this work.

3.1 Smarthome Control Approaches

In [39], the smart-home control problem is given an abstract formulation, relying on known and convex utility functions that represent the primary objective of each appliance. The problem is formulated as a mixed integer nonlinear programming (MINLP) problem, which are generally difficult to solve. TheY use a specialized algorithm, Generalized Bender's decomposition, which involves iteratively solving (simple) convex subproblems including only decision variables of elastic appliances. This approach approach is effective and fast, but seems as though it will be impractical in reality, as the restriction of utility functions as convex may be impractical. Nontheless, the system model as described provides a good framework to view smarthome control as an optimization problem.

In [65], three types of appliances are considered, fixed loads, regulate-able loads, and deferrable loads. The fixed loads are not considered in the optimization problem, because they cannot be adjusted. Secondly, the problem is simplified because it considers that regulate-able loads can be optimized sep-

arately than deferrable loads, because their problem formulation contains no constraints which include decision variables from both types of appliances. Notably, this is a limiting factor as global constraints such as "maximum household energy consumption" are used in other work, such as [39]. The work mainly focuses on regulate-able loads, particularly the heating ventilation and air conditioning (HVAC) system. They use machine learning methods to learn a mapping from the environment variable, temperature (desired, current, and outdoor), to the corresponding energy consumption.

The work presented in [32] is one of the earliest works applying reinforcement learning to the automated demand response problem. They present an algorithm, named CAES (Consumer Automated Energy Management System), which is based on Q-learning [58] to schedule device energy usage. The problem is treated as an energy consumption scheduling problem, where the system inputs a state-feature representation that is derived from assigned tasks, previously selected actions, and price signals. The overall objective of the system is to minimize a weighted sum of energy cost with an abstract "disutility" of delaying the device operation to the user. The actions in the system represent the amount of energy provided to each device, and therefore is a multi-dimensional action space. The results contrast the performance of the Q-learning system with an uncontrolled (easy to beat) system and demonstrate that the energy usage is reduced during peak cost times. While this approach demonstrates that reinforcement learning can be used to reduce energy cost while considering user objectives, there are limitations in that the system only considers cost of waiting, rather than a more general type of objectives that a user may have. Secondly, standard Q-learning is a tabular approach, and therefore scales exponentially with the dimensionality of the state and action space. Therefore, this approach in its given form does not scale to the possibility of having many devices present in a single smart-home.

The work of [59] builds on the work of [32], mainly by decentralizing the control across different devices. This can be viewed as transforming the system into a multi-agent reinforcement learning system, as individual Q-functions would be used to determine actions independently for each device. Some other adjustments to the previous work are also included: 1) Using user feedback instead of assuming a form for dissatisfaction, 2) allowing requested tasks to be at prespecified times, and 3) tasks can by assigned by the EMS as well as the user. The first is completed by surveying the user (eg: a scalar value the user believes represents their satisfaction) on the performance of that device for it's episode and rescaling (by user's believed importance) that number to include it in the reward function. The second is more-or-less a trivial modification that allows more flexibility for the user. The third allows the EMS to select actions to complete jobs that are not provided by the user (in a sense, it

predicts that a job will occur in the future). The state-space in this problem includes only 3 elements; the current price of electricity, time since last episode / time to request, and the user's specified priority for the task. This work is limited in that it still applies tabular Q-learning, which limits the capacity of the state-space due to the exponential growth of the table size with respect to the dimensionality of the state-space. Interestingly, the formulation provided for user feedback may be able to effectively represent more than just the dissatisfaction with delaying, although the paper assumes that user's dissatisfaction only is concerned with the schedule. Furthermore, the action space is strictly on/off, which eliminates the possibility of considering elastic loads (eg. dimmable lighting, EV charging rates, etc).

In [19], the proposed demand response scheme follows the idea of direct load control, where the utility company directly adjusts the home's appliances to suit their needs. The scheme only considers that devices can be turned off, not continuously variable devices. The scheme presented does consider a priority list for which appliances to turn off, which is based on a k-means clustering algorithm as well as the number of times the device has been turned off. This scheme is rather different than the work presented in this thesis, although it supports our claim that appliances should have variable importance in terms of trade-offs between cost and comfort.

In [21], the problem of rebound peaks arising from automated demand-response systems is considered. A rebound peak can occur because every automated energy management system will get the same energy pricing, so each home is motivated to schedule their energy consumption to the lowest price period, creating a new peak at that time instead. The first solution considers a flat price overnight, and suggests that the system should randomly select which part of the solution should be selected. A second solution proposed is to provide different pricing schedules for different homes overnight, leading to different optimal schedules for the various homes. The third proposed solution is to signal each home with a power threshold that must not be exceeded. In this thesis, we don't directly consider the effects of rebound peaks, but note that many of these proposed solutions should fit nicely in both the optimization and reinforcement learning schemes presented in later sections.

In [2], the proposed system takes an alternative approach to optimizing for both comfort (called satisfaction) and energy cost. The approach is to treat the satisfaction level as a constraint, then optimizing to get the minimum possible price. If the problem is infeasible, the satisfaction constraint is decreased, so satisfaction is gradually reduced until a desired minimum daily energy cost is met. This is an alternative approach to dealing with multi-objective optimization, which seems practical, but also seems limiting as tradeoffs between satisfaction and cost aren't apparent. An issue with this paper is

that satisfaction only considers curtailable appliances, (ones which can be switched off without needing to be switched on) and the order which appliances are turned off is from a priority list given by the user.

The work of [57] approaches the problem of EV charging by utilizing deep reinforcement learning. They formulate the environment dynamics using a MDP. Their network takes as input electricity price data, as well as the EV's current state of charge. Furthermore, the presence of the electric vehicle is considered, although only used to determine the start / end of an episode. Their network utilizes the DQN algorithm to train their network to estimate action values, as in standard Q-learning. The neural network used includes two sections; the first is an LSTM-based representation network, which extracts relevant features from the past 24h of electricity prices. The second section is fully connected, and aggregates the previous-price features with the current state of charge to determine actions-values for the system. The action space is single-dimensional, selecting what amount of energy to charge/discharge into the EV battery during the next time step, including negative energy (to sell back to the grid at the current price of electricity, or possibly, reduce the energy consumption of other devices). The system model for the EV's arrival, departure, and state of charge on arrival are each modelled using a truncated normal distribution. We utilize this model for EV charging in later sections. The performance measure / reward function uses a linear combination of two separate rewards. The first is cost, which includes energy cost and battery degradation. The second is called "range anxiety", which is the squared error from full charge, calculated and applied at departure time. They demonstrate the validity of their results using a variety of control techniques, with the best performing benchmarks including a few modifications of MPC, as well as their deep RL approach, which is surpassed only by a MPC approach that knows the future energy prices.

The work presented in [62] describes a multi-agent reinforcement learning method for home-energy management. They use extreme learning machines (a type of neural network) [14] to predict upcoming values for energy cost and expected energy production from solar panels. These predicted values are then fed into a Q-learning algorithm, which is responsible for selecting actions for the appliances. This work is interesting in that the features used for Q-learning are extracted from a supervised learning procedure, which could be called a transfer learning technique as a separate objective signal is being used to extract features for the reinforcement learning algorithm. It would be interesting to see the effect of using the features of the ELM as state for the Q-learning section of the algorithm, rather than predictions themselves.

Furthermore, [24], similarly to [62], proposes a multi-agent reinforcement learning based algorithm to home energy management. They use a neural network to forecast the prices, which are apparently input to the Q-learning [58] based agents to determine optimal energy usage. They claim to benchmark their performance against a MILP solver, Gurobi[16], by limiting the optimizer to not include any foresight of future rewards (corresponds to a discount factor of 0). Unsurprisingly, the multi-agent reinforcement learning approach performs better than Gurobi, due to the MILP being short-sighted in it's solutions (ie, not considering future reward for each time step). The environment model isn't clearly presented in this work, so the implications of the results are unclear.

The work of [40] applies batch reinforcement learning [10] to thermostatically controlled loads (specifically, an electric water heater, and a heat-pump thermostat for a residential building). Batch Reinforcement learning is a variant of the standard reinforcement learning problem, which [60] describes as "Originally defined as the task of learning the best possible policy from a fixed set of a priori-known transition samples". The proposed framework assumes a backup controller, which can override the action of the agent to guarantee user safety/comfort. Forecasted data is integrated into the algorithm by replacing some sections of the next-state data with the forecasted values as opposed to the true values. The action-values are estimated using function approximation via. "an ensemble of extremely randomized trees". An expert policy adjustment mechanism is also included. It leverages expert knowledge of the monotonicity of the desired policy by treating the expert knowledge as a set of convex constraints on a fuzzy model. This fuzzy model, which they claim is convex w.r.t. parameters θ, can then be optimized to match the policy output by the Q-function approximator, subject to the monotonicity constraints.

The work in [28] compares two deep reinforcement learning schemes, one based on policy gradient methods [49] and another based on DQN [26]. This paper approaches the problem in a centralized manner, where the neural networks determine the actions for all devices simultaneously. They assume 3 types of devices, time-shiftable (defferable), time-scalable (curtailable) and a joint time-shift and time-scalable (electric vehicle charging). The framework assumes that controllable devices can only be switched on/off, in both cases of curtailable and defferable loads. Device rewards are composed of 3 components, comfort, total energy consumption, and total cost. Comfort is modelled by considering how many times a device has already been turned on/off/delayed. It is unclear how the rewards are combined in this setting. The results indicate that the policy gradient approach performs better than the DQN approach.

The work of [15] claims (similar to this thesis) to be the first to describe the problem of smart-home control as a multi-objective reinforcement learning problem. The state space considered consists of each device's previously selected actions, which makes the dynamics deterministic. The dual rewards considered include a comfort metric, consisting of difference from user's previously prescribed action, and secondly an energy saving reward, which is the energy saved by the device compared to the prescribed action. The primary algorithm presented int his work seems to an unnamed version of multi-advisor reinforcement learning, although the work doesn't acknowledge the general sub-optimality of the approach. The work in this thesis differs in that it considers the multi-agent case, applies neural networks to learn Q-functions as opposed to a table, and the environment description includes much more detail in the state-space.

The work of [3] applies RL to HVAC control using Q-learning considering on/off control actions. They consider the state-space with 3 components; indoor temperature, outdoor temperature, and expected time to occupancy. The reward function is formulated to reduce heating costs by providing a negative reward for user discomfort, as well as a negative reward for consuming energy. This approach demonstrates the feasibility of reinforcement learning applied to thermostatically controlled residential heating, although savings are questionable when compared with a manually programmed thermostat. Our work differs in it's capability of automatically responding to demand response incentives, and instead of estimating time-to-occupancy, we use importance weights to account for the users schedule.

3.2 Reinforcement Learning Algorithms

Deep Q-networks, described in [27][26], are one of the first great successes of deep reinforcement learning. They use a neural network to estimate the action-value function, Q(s,a) and achieve human-level performance on simple atari games. To achieve this performance, they use 2 main tricks. The first trick is to use an experience replay buffer which is randomly sampled to determine a loss during the gradient descent step of training the neural network. Sampling from the experience buffer attempts to un-correlate the samples, as experience at time t is normally more correlated with experience at $t + \Delta t$ than it is with experience collected at $t + 5\Delta t$. The second trick involved the use of a target network, a previously stored version of the current network. The slower changing target network reduces the bias caused by the target y_t shown in equation 3.1. The issue is caused by the fact that the update for $Q(s_t, a_t)$ will likely affect the output given for $Q(s_{t+1}, a_{t+1})$. The update can lead to divergence, but is

less likely with the gradually changing target network.

$$y_t = R_t + argmax_a Q(s_{t+1}, a) \tag{3.1}$$

The work of [11] describes a multi-agent algorithm used in the cooperative setting, where each agent shares the same reward function. This work uses the centralized training and decentralized execution paradigm common in the multi-agent setting. They propose a policy gradient, which utilizes a centralized critic to compute the gradients for each agent's policy. This technique provides an approach for dealing with the credit assignment problem. They propose a method based on difference rewards [61], which utilizes a default action to determine the effects of the individual actions. Essentially, each policy i uses an advantage-based gradient which uses a baseline calculated by assuming only agent i has it's own actions changed. This algorithm is tested utilizing the popular game StarCraft, by putting trained policies into each of the units, and having them battle against a team of the same units controlled by the internal StarCraft AI. The results demonstrate that the proposed algorithm dominates independent techniques, and is competitive with fully centralized approaches.

The work in [23] proposes a multi-agent modification of the traditional actor-critic algorithm, applicable to both cooperative and competitive settings. The multi-framework used to model the interaction is Markov Games. They take the common framework of centralized critics and decentralized policies. First, they describe a policy gradient that relies on one centralized Q-function for each reward function. This Q-function requires knowledge of each agent's actions, as well as knowledge of each agent's policy (required to determine the joint action a' in the target $y = r_i + \gamma Q(s', a')$). This fits into the policy gradient simply as is standard with multi-dimensional actions. Secondly, they propose removing the necessity of policy knowledge by having each agent instead estimate the others' policies. Finally, they propose learning multiple policies for each agent, and randomly selecting which policy to use. This makes the agents more robust in the sense that opponents cannot easily counter their policy. They apply the proposed algorithm to some seemingly simple environments, involving both competitive and cooperative situations. The results demonstrate the the potential of the algorithm to function in in multi-agent environments.

The work of [29] describes an algorithm for the multi-objective reinforcement learning utilizing DQNs. The algorithm trains one DQN each with a particular relative importance weight vectors. These weight vectors are carefully selected using a subroutine, which attempts to select a small set of weights that well-covers the optimal boundary of the set of policies. After the algorithm has completed, the user

need only input the weights for the current episode, and is returned to with a DQN which is optimal for this set of weights. Each DQN is trained independently (possibly with parameter reuse) in the environment until convergence. This algorithm is intuitive but has a seemingly devastating requirement (somewhat resolved by parameter reuse); each neural network has to interact with the environment from scratch to be trained on a single weight vector. Overall, the algorithm seems to show promise, provide a theoretically sound approach, but the limited results restrain the perceived capabilities of the proposed algorithm.

3.2.1 Multi-Advisor Reinforcement Learning

Multi-Advisor Reinforcement Learning (MAd-RL)[20], is an unpublished paper that describes a general scheme of decomposing a single reward function into a set of reward functions, learning action-value functions for each decomposed reward signal. These action-value functions are then aggregated into selected actions, resulting in a control policy. They describe egocentric, agnostic, and empathic planning for the advisors, which are approaches for selecting a_{t+1} when evaluating $Q(s_{t+1}, a_{t+1})$ in the Bellman equation. The experiments in this work are minimal yet promising, matches the scheme in our work, and provides good theoretical foundations for the algorithm's convergence.

The Hybrid-Reward Architecture, presented in [55] can be considered as an implementation of MAd-RL. This system approaches the game Ms. Pac-Man by learning a General Value Function (GVF) for each location on the map, and using these GVFs to estimate the action-value for each object on the screen which include ghosts, pellets, blue ghosts, and fruit. This results in more than 400 objectives. This demonstrates the capability of a multi-advisor system to achieve strong performance in a complex environment, and demonstrates scalability of including many objectives.

The work of [42] is some of the earliest work on decomposing reward functions, and notes at least one possible issue present in decomposing reward functions for RL problems. In particular, the Dollars vs. Euros example demonstrates that non-optimal behaviour can occur from learning separate action-value functions using off-policy learning. Nonetheless, we accept the possible sub-optimality of using MAd-RL in this paper due to the demonstrated successes of such an approach in complex environments.

3.3 Discussion

In the previous sections, various techniques for smart-home control are considered. There is variety in the types of devices considered, with distinctions among the control signals, the system dynamics, and the objective structures. Overall, it seems that both reinforcement learning and optimization are commonly researched and well established approaches for control of smart-homes, especially under the

motivations provided by demand response applications. These schemes are related, "Reinforcement Learning is Direct Adaptive Optimal Control"[48].

Nature of Objectives

As we describe smart-homes as multi-objective control problems, we can consider how each of the schemes handles the multi-objective nature of the system, where we observe 2 primary approaches. Firstly, the individual objectives performance measures are aggregated (often linearly) into a single, scalar performance measure. Secondly are approaches in which some objective is treated as the optimization objective, and the others are considered as constraints, often of the "must exceed minimal value" form. In RL approaches, these objectives are described by a reward function rather than the optimization objective. These reward functions are normally aggregates of individual objectives. In the multi-agent case, there are multiple reward functions, but these correspond to the different agent's unique objectives, rather than a single-agents joint objectives. The only work which describes a deliberate multi-objective reinforcement learning framework is [15], which postdates our original multi-advisor work presented in chapter 5.

Translation: Reward, Utility, Action-Value

Generally speaking, a utility function is a function which maps inputs to a scalar value. This scalar value represents a preference for the inputs, where higher values indicate more preferred inputs. If utility functions are available during a sequential decision-making, one would expect they could differentiate between actions, similar to an action-value function. In the optimization approaches seen, some utility functions are more akin to reward functions, as they are accumulated across time-steps and don't represent the future. The accumulated utility functions usually are the objective functions, and match the action-value function for the initial state. In this sense, utility as used in the literature is more similar to the reward function, rather than the intuitive consideration that utility functions are akin to action-value functions. Furthermore, the optimization approaches inherently have a finite horizon, and thus do not consider the value of the terminal state. This could lead to sub-optimal performance, but is also neglected when considering episodic reinforcement learning.

Devices and Dynamics

Each approach discussed selects some type of devices which it considers. Very commonly, the categories of devices can be represented by a smaller set of device categories, as described in [39]. Common appliances include HVAC, electric vehicles, lighting, dishwashers, and vacuum cleaners. Normally, similar devices have similar dynamics, the first order thermal model used in this thesis is common to both optimization and reinforcement learning approaches. The main uniqueness in dynamics is whether stochastic dynamics are considered or not. In the stochastic case, MDPs are used, which generalize the deterministic case to include stochastic state transitions. The stochastic formulations are generally more applicable to the real world, but come with more complexity.

Chapter 4

Importance Scaling of Elastic Appliance Utility Functions for Automated Power Management in Smart Homes

4.1 Introduction

The automatic adjustment of appliance settings is formulated here as an optimization problem, where the energy consumption of each appliance in each period is selected to balance user comfort with cost saving. A trade-off ratio is assumed to be defined by the user. Due to the diverse nature of residential appliances, different models will need to be included for different appliance types. Furthermore, each appliance will have constraints limiting the types of adjustments that can be made. For the purpose of this work, only appliances which have a continuously variable power output during a period are considered, which are referred to as elastic appliances. The categories of elastic appliances are further divided along how their performance depends on previous energy consumption.

In [39], the primary reference for this section, the problem is presented as a mixed-integer non-linear programming problem, where they include both elastic and inelastic appliances. The inclusion of the inelastic appliances make the optimization problem more difficult to solve, so Generalized Benders Decomposition [13] is used to find the optimal solution. This procedure breaks the problem down into a subproblem and a master problem. In that case, the subproblem is set by fixing the decision variables for the inelastic appliances, and finding an optimal solution for the decision variables for elastic appliances. The subproblem of [39] matches the problem in this section with minor modifications.

Following the work in [39], we classify appliances into three categories based on their memory property. The 3 appliance categories are named as memoryless, full memory, and partial memory. A memoryless appliance's performance depends only on the current energy consumption, a full memory appliance's performance depends on the cumulative energy consumption during the considered horizon, and a partial memory appliance's performance is affected more by recent energy consumption than past energy consumption. An example of each would be a dimmable light bulb, a battery charger with variable charging rates, and an electric heating system respectively. This chapter does not consider inelastic appliances as in [39], this exclusion allows us to formulate the problem as convex which is tractable. The problem is then extended to also consider that in [39], the individual appliance utility functions are not scaled relative to each other, and therefore the system does not account for which appliances are more important to the user's comfort. We make this adjustment is made by including a weight for each appliance which re-scales the utility function so that user preferences between appliances are represented.

4.2 Problem Formulation

As mentioned earlier, the model presented in this chapter is based on the work provided by [39] with the exclusion of the inelastic appliances. Therefore, this section considers a subset of the problem formulation pertaining to the main problem that we are considering.

4.2.1 System Representation

To model the energy consumption of a single residence, the appliances need to be modelled in terms of their energy consumption. First we define an appliance as $a \in \mathbf{A}$, where $\mathbf{A}$ is the set of all appliances in the household being considered. All appliances considered are divided into one of the 3 mentioned categories, $\boldsymbol{A_{EML}}$ denotes the set of memoryless appliances, $\boldsymbol{A_{EFM}}$ denotes the set of full memory appliances, and $\boldsymbol{A_{EPM}}$ denotes the set of partial memory appliances.

This work considers a finite planning horizon, which could be considered to be one day. The horizon is broken down into T time slots. Each appliance $a \in \boldsymbol{A}$ is assigned a vector $\boldsymbol{x_a}$ of length T, where each element of the vector represents the energy consumed in that time slot. That is, each appliance, a has energy consumption for the given planning horizon given by

$$\boldsymbol{x_a} = [x_a^1, x_a^2, \ldots, x_a^T], \quad \forall a \in \boldsymbol{A} \tag{4.1}$$

where x_a^t acts as a decision variable for the amount of energy consumed by appliance a during time slot $t \in [1, 2, \ldots, T]$. In general, the set of all energy consumption vectors for all appliances will be

denoted $\boldsymbol{x}$, which can be interpreted as a matrix of decision variables, where each row represents the consumption for a single appliance $a \in \boldsymbol{A}$, and each column represents the energy consumption for a time slot, $t \in \boldsymbol{T}$

For each time slot, a unit price of electricity is considered, denoted λ^t, and thus the entire cost of electricity for the horizon can be represented as

$$C(x) \; = \sum_{t \in \boldsymbol{T}} [\lambda^t \sum_{a \in \boldsymbol{A}} x_a^t] \tag{4.2}$$

4.2.2 System Constraints

The first constraint considered is that each appliance can be assumed to have a schedulable interval $\boldsymbol{T_a} = \{S_a, S_a + 1, \ldots, F_a - 1, F_a\}$ which is represented as a constraint as

$$x_a^t = 0, \quad \forall a \in \boldsymbol{A}, \quad \forall t \in \boldsymbol{T} \setminus \boldsymbol{T_A} \tag{4.3}$$

Secondly, it is considered that the household energy consumption for a period may be bounded to some maximum energy threshold, E_{th}^t, for each time slot. This can be represented as

$$\sum_{a \in \boldsymbol{A}} x_a^t \; \leq \; E_{th}^t, \quad \forall t \in \boldsymbol{T} \tag{4.4}$$

4.2.3 Appliance Performance

Each appliance is assumed to provide it's own constraints and utility function. The utility function for each appliance, denoted $U_a(x_a)$, is a measure of appliance performance as a function of energy consumption, and for this problem utility functions are all assumed to be convex in order to simplify the solution. The assumed form of the utility function for each type of appliance follows.

Memoryless Appliance Performance

For memoryless appliances, it is assumed that the performance depends only on the current energy consumption, and is unaffected by previous consumption. This indicates that the utility function will be defined for individual time slots, $U_a^t(x_a^t), \; \forall t \in \boldsymbol{T}, \; \forall a \in \boldsymbol{A_{EML}}$. The total utility for the appliance in the given horizon is

$$U_a(\boldsymbol{x_a}) \; = \sum_{t \in \boldsymbol{T}} U_a^t(x_a^t), \quad \forall a \in \boldsymbol{A_{EML}} \tag{4.5}$$

Full Memory Appliance Performance

For appliances with full memory, performance depends only on the cumulative energy consumption throughout the day. Therefore, the utility function is assumed to be represented as a function of the

sum of the total consumed energy, which is represented as

$$U_a(\boldsymbol{x_a}) \;=\; U_a(\sum_{t\in\boldsymbol{T}} x_a^t), \quad \forall a \in \boldsymbol{A_{EFM}} \tag{4.6}$$

Partial Memory Appliance Performance

Appliances with partial memory provide the most complex utility functions, in that partial memory appliance are assumed to be in control of some environment variable, denoted $\theta_a^t(x_a^t)$. It is assumed that the time-evolution of this variable can be represented as

$$\begin{aligned} \theta_a^t(x_a^t) \;&=\; \epsilon_a\theta_a^{t-1} + (1-\epsilon_a)(W_a^t + K_a^t x_a^t), \\ &\forall t \in \boldsymbol{T_a}, \quad \forall a \in \boldsymbol{A_{EPM}} \end{aligned} \tag{4.7}$$

Where ϵ_a represents the time constant for the system, W_a^t represents the default value θ_a^t will converge to with no energy input. In the case of an electric heater, W_a^t could represent the outdoor temperature during timeslot t. If no energy is input, then the indoor temperature will eventually reach the outdoor temperature. Finally, K_a^t represents a conversion rate from input energy to the change in the environment variable.

The utility function for partial memory appliances is assumed to be a function of the environment variable, θ_a^t as opposed to the energy consumption. Regardless, the environment variable is given as a function of energy consumption (and previous state) in (4.7). Due to the time varying nature of θ_a^t, it is assumed that the utility function will be defined for the current timeslot being considered. The overall utility function over the entire planning horizon is considered as

$$U_a(\boldsymbol{x_a}) \;=\; \sum_{t\in\boldsymbol{T_a}} U_a^t(\theta_a^t(x_a^t)), \quad \forall a \in \boldsymbol{A_{EPM}} \tag{4.8}$$

4.2.4 Objective Function

In [39], the objective function is called the net utility, and is defined as the objective considering two weights which represents the rate of trade off between appliance performance and total cost. The net utility in [39] is represented as

$$NU(\boldsymbol{x}) \;=\; w_u \sum_{a\in\boldsymbol{A}} U_a(\boldsymbol{x_a}) - w_c \sum_{t\in\boldsymbol{T}} [\lambda^t \sum_{a\in\boldsymbol{A}} x_a^t] \tag{4.9}$$

where $w_u \geq 0$ and $w_c \geq 0$ represent the importance of appliance performance and cost savings respectively. This allows the simultaneous maximization of performance, while minimizing the cost, according to the user's preferences.

This net utility function accounts for the importance between performance and cost, but doesn't consider the differences in importance for the utility functions. When considering the source of these utility functions, it would be reasonable to assume they are provided by the appliance developers. This could cause an issue because a greedy developer could try to produce a higher scaled utility function for their appliance, leading to it being prioritized while optimizing, regardless of the preference of the consumer. Furthermore, this approach doesn't account for the importance of any appliance relative to the others based on the users preferences because all appliances are weighted equally.

A simple modification to reduce these issues is to introduce a new vector of weights, $\boldsymbol{w} = (w_a)_{a \in \boldsymbol{A}}$ where each term represents the importance of one particular appliance relative to the others. This set of weights could possibly be learned automatically as the user adjusts their appliances from the policy prescribed by the system. These manual changes could be use to indicate the adjusted appliance requires a greater importance. The new net utility function including these weights can be represented as:

$$NU(\boldsymbol{x}) \; = \; w_u \sum_{a \in \boldsymbol{A}} w_a U_a(\boldsymbol{x_a}) - w_c \sum_{t \in \boldsymbol{T}} [\lambda^t \sum_{a \in \boldsymbol{A}} x_a^t] \tag{4.10}$$

Where w_a is the new addition which accounts for the relative importance of the various appliances.

This utility function could be simplified while still representing the same problem by removing some weights. In the work of [39], only 2 weights are considered, w_u and w_c. This could be reduced to only be a single weight by only considering the ratio of w_u to w_c. Furthermore, our proposed inclusion of w_a serves same purpose as w_u, so w_u can be removed in future work.

In this work, all the weights are considered in the experiments for more control during tuning. In future work, another resource other than energy could be considered, for example water usage, internet data usage, or natural gas. In this case it would be worth considering a weight for each resource.

4.2.5 Appliance Constraints

Each appliance is assumed to have it's own set of constraints. Two main types of constraints on some appliances are a) bounds on the appliance's energy consumption in a period, and b) minimum performance threshold. The only remaining considered constraint is on partial memory appliances, that the environment variable should remain within some reasonable bounds to guarantee user comfort. These constraints are articulated in the following.

Appliance Energy Consumption Bounds

Every appliance is assumed to have some minimum and some maximum threshold for it's energy consumption in a period. This could be interpreted as the maximum and minimum settings for the appliance. For example, a computer completing some job could reduce it's power consumption to the point where the job is paused, but needs to maintain a minimum power consumption to stay powered on. This constraint is represented as bounds on the decision variable as

$$x_a^{min} \leq x_a^t \leq x_a^{max}, \quad \forall a \in \boldsymbol{A}, \quad \forall t \in \boldsymbol{T_a} \tag{4.11}$$

Minimum Performance Threshold

For memoryless and full memory appliances, we assume there is some minimum performance threshold denoted R_a that must be met for that appliance's utility function. The utility function types vary, so the formulation is different depending on the appliance type.

For a memoryless appliance, the performance doesn't depend on previous states, so the minimum performance threshold is assumed to only apply to the current state, which can be represented as

$$U_a^t(x_a^t) \geq R_a, \quad \forall a \in \boldsymbol{A_{EML}}, \quad \forall t \in \boldsymbol{T_a} \tag{4.12}$$

where R_a is assumed to be constant across all time slots considered. For full memory appliances, the performance depends only on the total energy consumption, so the minimum threshold applies to the overall performance threshold as

$$U_a(x_a) \geq R_a, \quad \forall a \in \boldsymbol{A_{EFM}} \tag{4.13}$$

Partial Memory Appliance Comfort Constraint

A partial memory appliance controls some environment variable, and the utility function is based on the state of this environment variable. Instead of setting a minimum performance constraint for partial memory appliances, bounds are set on the controlled environment variable in order to constrain performance for user comfort in a more direct way. This is formulated as bounds on the variable as

$$\theta_a^{min} \leq \theta_a^t \leq \theta_a^{max}, \quad \forall a \in \boldsymbol{A_{EPM}}, \quad \forall t \in \boldsymbol{T_a} \tag{4.14}$$

where the bounds on the environment variable are assumed to be constant across all time slots. For example, these bounds could be limitations that the indoor temperature must remain between 20 and 25 degrees Celsius.

4.2.6 Overall Formulation

Considering all constraints and the objective function described above, the overall problem can be formulated as

$$\begin{aligned}
\max_{\boldsymbol{x}} \quad & w_u \sum_{a \in \boldsymbol{A}} w_a U_a(\boldsymbol{x_a}) - w_c \sum_{t \in \boldsymbol{T}} [\lambda^t \sum_{a \in \boldsymbol{A}} x_a^t] \\
\text{s.t.} \quad & U_a^t(x_a^t) \geq R_a, \quad \forall a \in \boldsymbol{A_{EML}}, \quad \forall t \in \boldsymbol{T_a} \\
& U_a(x_a) \geq R_a, \quad \forall a \in \boldsymbol{A_{EFM}} \\
& \theta_a^{min} \leq \theta_a^t \leq \theta_a^{max}, \quad \forall a \in \boldsymbol{A_{EPM}}, \quad \forall t \in \boldsymbol{T_a} \\
& x_a^{min} \leq x_a^t \leq x_a^{max}, \quad \forall a \in \boldsymbol{A}, \quad \forall t \in \boldsymbol{T_a} \\
& x_a^t = 0, \quad \forall a \in \boldsymbol{A}, \quad \forall t \in \boldsymbol{T} \setminus \boldsymbol{T_A} \\
& \sum_{a \in \boldsymbol{A}} x_a^t \leq E_{th}^t, \quad \forall t \in \boldsymbol{T}
\end{aligned} \tag{4.15}$$

which is the same as the subproblem presented in [39], with the addition of the weights, $\boldsymbol{w_a}$. They classify this problem as a convex non-linear programming problem, due to the assumption that utility functions are all convex.

4.3 Simulation Results

For simulation purposes, one of each type of appliance is considered, and the partial memory appliance considered is an electric heater. $\boldsymbol{T}$ is 24, representing 1 hour time slots in a one day planning horizon. Hourly electricity prices and other parameters are selected to be similar to values used in [39] for numerical results. See Fig. 4.1 for hourly specific pricing. The selected settings result in a utility between 60 and 70 for the day for each individual appliance, and a total cost of approximately 72 for the day. These values assume all importance weights in the net utility are 1. The goal of the experiments is to examine if varying importance weights can lead to greater user happiness.

4.3.1 Experimental Parameter Settings

For experiments, generally all parameters other than noted differences are constant. Appliance weights, w_a, are all 1 unless otherwise stated. Following are parameters associated with any specific section of the problem. Energy consumption thresholds, x_a^{min} and x_a^{max} are selected to be 0 and 1 respectively for all appliances, except for partial memory appliances, where x_a^{max} is instead selected to be 4. Temperature thresholds for for the electric heater are selected to be $\theta_a^{min} = 20$ and $\theta_a^{max} = 25$. The minimum utility threshold, R_a is selected to be 2 for memoryless appliances, and 60 for full memory appliances. All appliances have a schedulable interval for the entire day, except for full memory

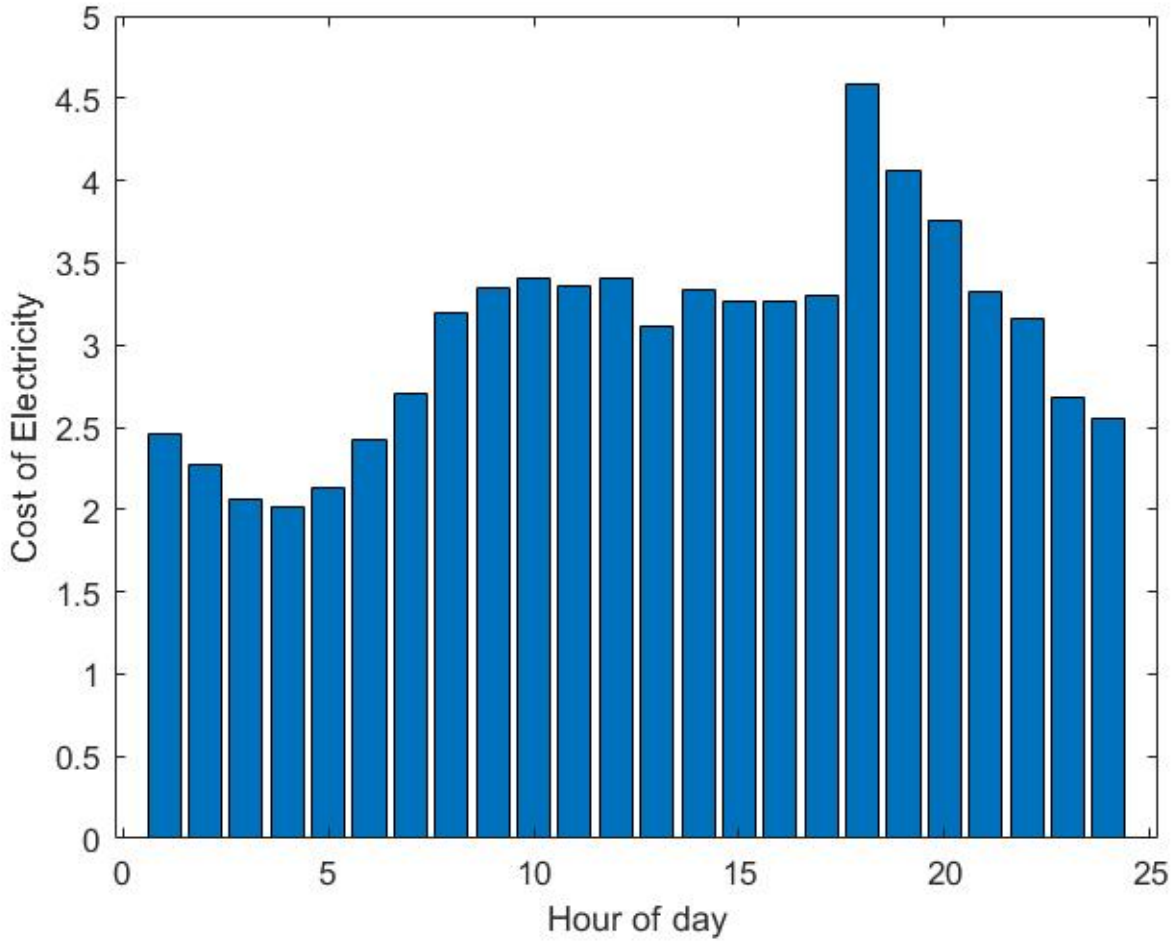


Figure 4.1: Cost of Electricity by hour of day

appliances, where S_a is considered as 9 and F_a is 24, ie. it won't run for the first 8 hours of the day. The limit for total energy expenditure was set to 4 for every hour of the day. The utility functions for each appliance are shown below:

$$U_a^t(x_a^t) = 5log(1 + x_a^t), \quad \forall a \in \boldsymbol{A_{EML}}, \quad \forall t \in \boldsymbol{T} \tag{4.16}$$

$$U_a(\boldsymbol{x_a}) = 30log(1 + \sum_{t \in \boldsymbol{Ta}} x_a^t), \quad \forall a \in \boldsymbol{A_{EFM}} \tag{4.17}$$

$$U_a^t(\theta_a^t) = log(1 + 0.8\theta_a^t), \quad \forall a \in \boldsymbol{A_{EPM}}, \quad \forall t \in \boldsymbol{T} \tag{4.18}$$

The initial indoor temperature is selected as 22.5 in order to see initial changes. The model for the time-evolution of the temperature as a function of input power is given below.

$$\begin{aligned} \theta_a^t(x_a^t) \ &= \ 0.7\theta_a^{t-1} + 0.3(19 + 1.5x_a^t), \\ &\forall t \in \boldsymbol{T_a}, \quad \forall a \in \boldsymbol{A_{EPM}} \end{aligned} \tag{4.19}$$

4.3.2 Solution Algorithm

This optimization problem is a convex programming problem, [39] indicates they are easy to solve. The problem solution was implemented in MATLAB using the optimization toolbox. The algorithm used by the solver is an interior point algorithm [7].

4.3.3 Results

Unweighted

In Fig. 4.2, all weights are set to 1. This is considered as the default case, with no priority between cost and the net utility. It is noted that the average energy consumption in an hour is below half of the maximum hourly energy consumption, and the temperature remains near the threshold early on, indicating that the utility for temperature is rather low compared to the cost of generating that temperature.

Cost Independent

In Fig. 4.3, the weight for cost is set to 0. This is the maximization of the utility functions, each is equally important for overall performance. Fig. 4.3 indicates both full memory and memoryless appliances take priority to improve the utility, as both maintain their maximum hourly energy consumption in a time period. In this case, the temperature doesn't stay near the maximum of 25, which indicates that the constraints for power consumption are too stringent to effectively maximize the performance.

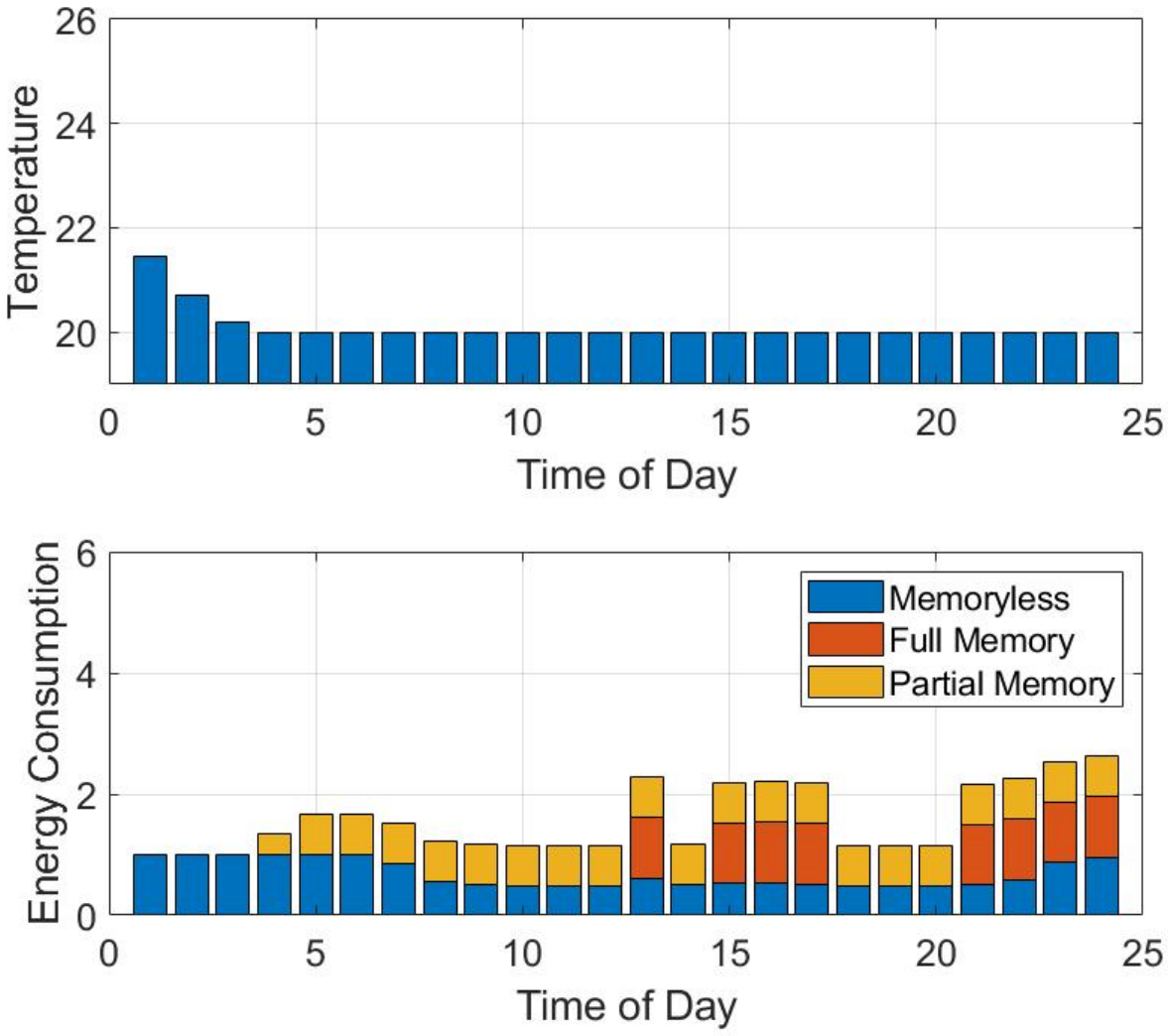


Figure 4.2: Appliance Energy Settings with all weights equal

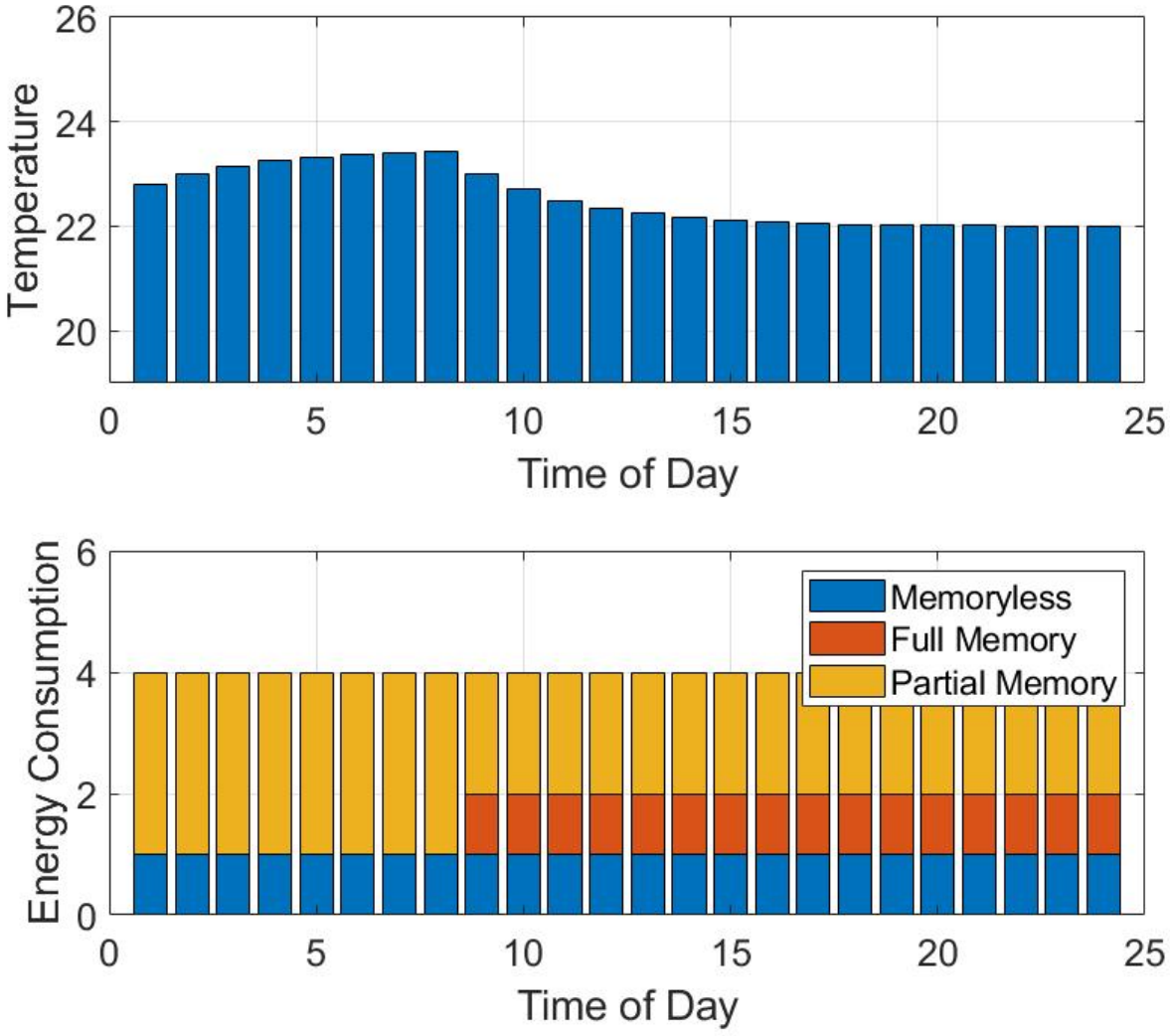


Figure 4.3: Appliance Energy Settings, $w_c = 0$

Performance Independent

In Fig. 4.4, the appliance performance component of the net utility is neglected by setting $w_u = 0$. This results in minimal energy consumption for all periods while maintaining the minimum thresholds for performance.

Heater Dominant

In Fig. 4.5, the utility function for partial memory appliances is scaled up by a factor of 100, which indicates a much greater reward from keeping the temperature higher. A factor of 100 is rather large, and could be reduced by first normalizing the utility functions for each appliance type. This factor results in a higher average temperature compared to cost independent results, indicating that a relative scaling factor may result in greater comfort with constraints on household energy consumption in a period.

4.4 Conclusion and Future Work

In this paper, it is shown that adding a scaling factor to the performance measure of various appliances can possibly better represent the users comfort preferences. This is indicated by the fact that average temperature is greater when scaling the importance factor for the electric heater than when the optimization is independent of cost, indicating that a user who prioritizes indoor temperature when considering indoor comfort is getting a greater overall satisfaction. This indicates that a more tunable system can result in greater user comfort, although this would require the user to manually indicate which appliances are important, and at which relative scale. This seems infeasible to expect a general consumer to provide directly. Future work could include methods of normalizing individual appliance utility functions so that weights are more individually meaningful, and to possibly learn the optimal value of the performance weights from user interactions.

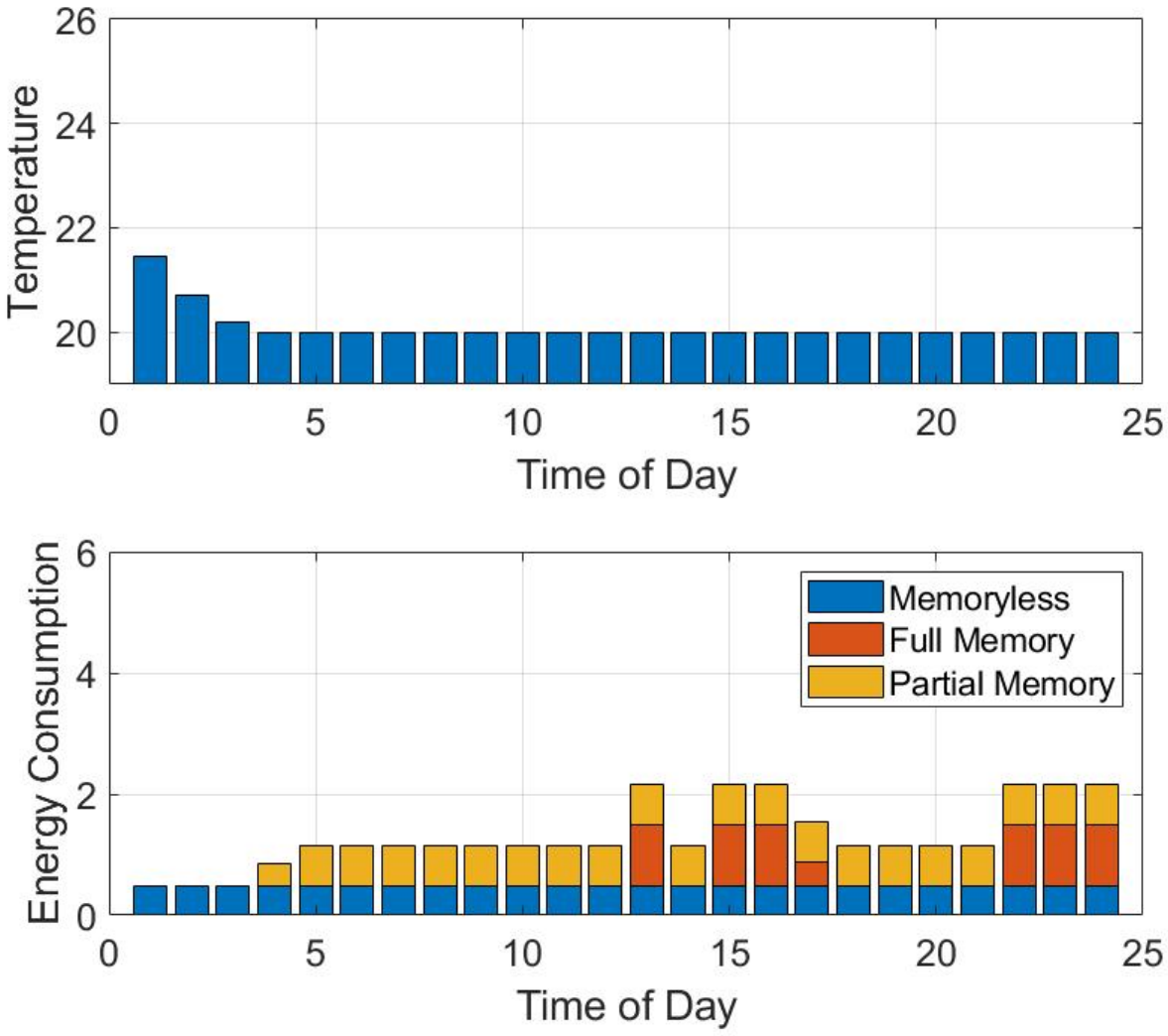
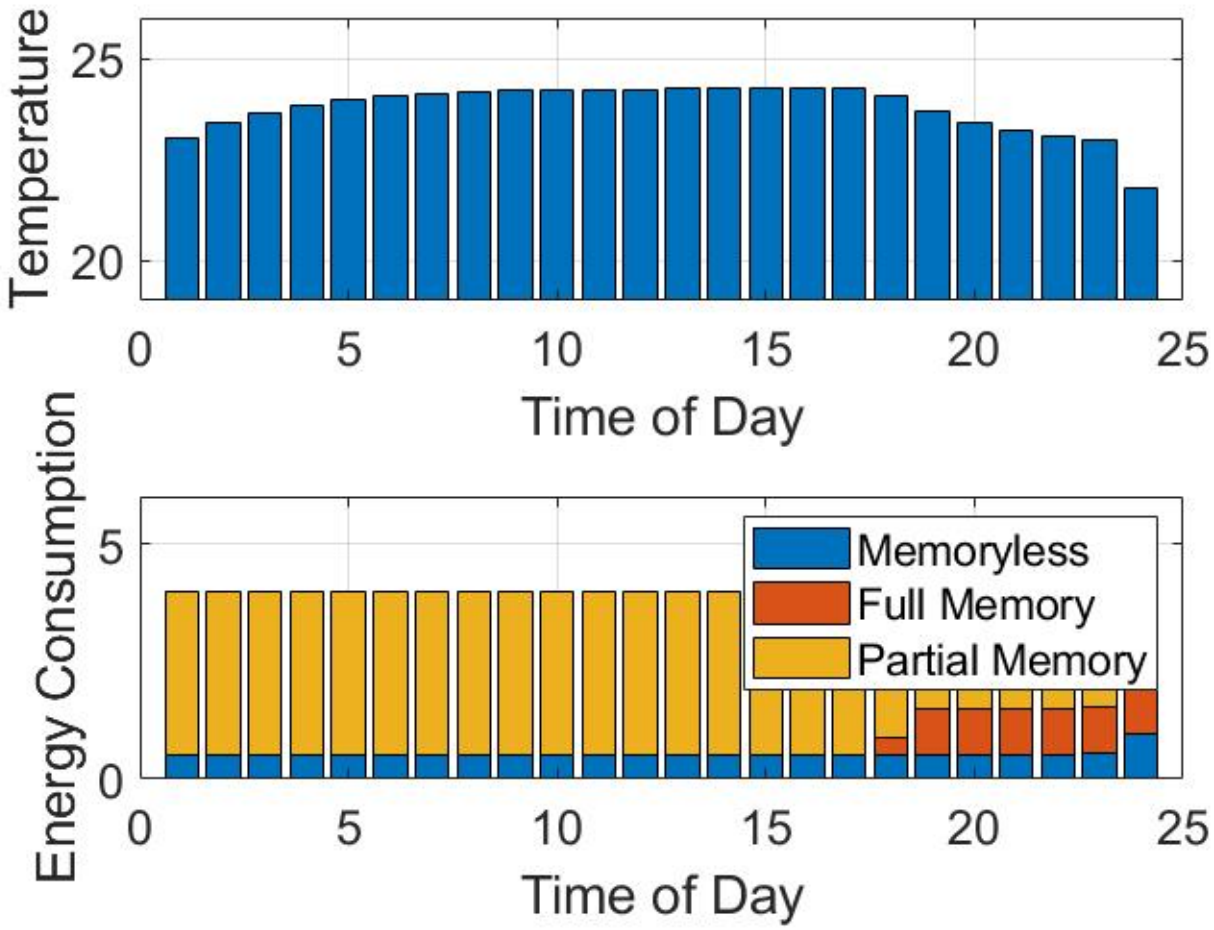


Figure 4.4: Appliance Energy Settings, $w_u = 0$



v

Figure 4.5: Appliance Energy Settings, $w_{EPM} = 100$

Chapter 5

Multi-Advisor Reinforcement Learning for Residential Heating

5.1 Introduction

In the previous chapter, the effect of the weighting on the cumulative utility functions is considered, and it was demonstrated that performance can be improved by carefully selecting the weights to account for varying user preferences. In this chapter, we consider the single-appliance task of electric heating under a MDP formulation to determine the effectiveness of multi-advisor reinforcement learning to achieve trade-offs between objectives. In this setting, we consider state-based weighting, where the user's preferences for achieving objectives is a function of the state.

This work uses multi-advisor reinforcement learning (MAd-RL) [20], which learns an action-value function for each each of a set of reward functions. In the original context, the reward function is described as being decomposed, and each element used as a distinct learning signal for a Q-function. In this setting, we consider that rewards are naturally independent learning signals, and use MAd-RL to learn representations from these signals separately as well as aggregate the resulting representations.

The formulation utilizes a simple discrete first-order heating dynamics model, with realistic values noted from [8]. This task considers two objectives, indoor heating (set-point tracking) and energy consumption reduction. The reward functions for these 2 tasks are negative squared error and energy consumed respectively. To control the indoor temperature, a state space is selected which provides all the relevant information used by the dynamics, reward function, and aggregator.

5.2 System Model

The standard RL problem involves an agent interacting with an environment. At each time step, t, the agent receives reward R_t and observes state S_t. It then selects an action, A_t, and the cycle repeats forever (or until the end of the episode). In the MAd-RL setting, we consider that each agent sees the same state-space, and provides an action-value for each action in the action space.

Parameter and variable notation for this section is as follows:

- S represents the set of all states
- t is an index for time
- q_t represents the energy used by the heater during time-step t
- q_{max} is the maximum energy input to the heater in a single time-slot
- T_t^{in} represents the indoor temperature at time t
- T_t^{out} represents the outdoor temperature at time t
- ϵ represents the system inertia
- η represents the efficiency of the heater
- A represents the thermal conductivity
- $w_{i,t}$ represents the importance of objective i at time t
- K is the number of discrete action steps
- k indexes action steps

5.2.1 Environment Description

The environment that the agent interacts with receives an action at every time step t. It uses this action with the current state to determine the next state, and the reward. Due to the reward decomposition approach, rewards are returned as a vector, where each element corresponds to a single objective.

Reward Functions

We consider two objectives in this work: Thermal Set-point Tracking, and Energy Consumption Minimization. Each has it's own associated reward function, which are defined subsequently. The thermal set-point tracking reward is selected as squared error from set-point, given as:

$$R_{1,t} = -(T_t^{in} - T_t^{set})^2 \tag{5.1}$$

The reward selected for the energy consumption minimization objective is as follows:

$$R_{2,t} = -q_t \tag{5.2}$$

The overall reward received by the agent is defined as follows:

$$R_t = w_{1,t}R_{1,t} + w_{2,t}R_{2,t} \tag{5.3}$$

where $w_{i,t}$ is the importance of objective i during time slot t.

Thermal Model

For the purposes of this paper, we assume that a residential thermal control environment can be approximated using a first-order equivalent thermal parameter model which is commonly used in the literature [39], [65], [8], and can be expressed as follows:

$$T_{t+1}^{in} = \epsilon T_t^{in} + (1-\epsilon)(T_t^{out} + \eta q_t/A) \tag{5.4}$$

State Space

In order to apply RL, the state-space needs to be defined such that it effectively approximates the Markov property, that is, the distribution of the next state only depends on the current state and action, not on previous state and actions:

$$P(S_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t, \cdots, S_0, A_0) \tag{5.5}$$

We interpret this as state requiring any information necessary to accurately predict the next state. Therefore, we select state to include all variables deemed necessary to accurately predict the next temperature, as well as rewards received by each agent. Therefore, the state space for each time S_t is defined as

$$S_t = \{T_t^{in}, \quad T_t^{out}, \quad t, \quad T_t^{set}\} \tag{5.6}$$

In future work, performance may possibly be improved by further increasing the dimensionality of the state-space to include other variables.

Action Space

For a heater, the action space can be considered as the amount of energy that the heater uses in a time-slot. The action selected at every time slot is denoted q_t. Due to selection of DQN to estimate the Q-values for each state, it is required to use a discrete action-space. Therefore, the action space is discretized. This discretization is done by considering that the energy consumed by the heater is bounded between 0 and some maximum value, q_{max}, and assuming a fixed number of discrete actions, K. In other words, the agent selects q_t, which can be expressed as

$$q_t = \frac{q_{max} k}{K}, \quad k \in [0, K-1] \tag{5.7}$$

5.2.2 Agent Description

The agent is a MAd-RL based agent, which estimates Q-values for each objective. Each advisor provides it's action values to the aggregation function, which then aggregates all opinions into a final selected action. Each advisor uses a DQN to estimate the action values for the current state.

Advisor Description

The neural network used for each advisor consists of 4 fully connected layers, using Rectified Linear Units (ReLU) as activation functions. Each layer uses 32 neurons, which provided enough capacity to achieve reasonable performance in this environment model.

Aggregation Function

At every time-step, each advisor provides it's estimate for $Q(s, a)$ for each agent. Every Q(s,a) is normalized to allow more freedom in the scale of reward definitions. The normalized action values, $Q_{norm}(s, a)$ are calculated as follows:

$$Q_{norm}(s, a) = \frac{Q(s, a) - \min_a Q(s, a)}{\max_a Q(s, a) - \min_a Q(s, a)} \tag{5.8}$$

This normalization technique reduces the range of $Q(s, a)$ for any action to be between 0 and 1, while still expressing the advisor's relative preference between actions. The normalized action-values are then used to select an action as follows:

$$A_t = \arg\max_a [w_{1,t} Q_1(S_t, a) + w_{2,t} Q_2(S_t, a)] \tag{5.9}$$

where $Q_i(S_t, a)$ is the action value provided by advisor i to the aggregation function during time period t.

5.3 Simulation Results

The MAd-RL approach is assessed by selecting reasonable parameters and observing the agent's performance using single day episodes with hour-long time-steps. Action space is defined to include 50 steps (K=50). During training, the importance schedule of each objective is kept to a fixed vector. This vector indicates the importance of each objective for a select time-slot. Selected values are shown in fig. 5.1.

Thermal parameters were selected based on [8] for the heater's operation. The major difference is that the heater's capacity is reduced in order to achieve more effective performance with a relatively smaller action-space. These selected parameters are as follows:

- Inertia: $\epsilon = e^{-\frac{1}{25}}$
- Thermal Conductivity: $A = 0.0778 \quad kW/^\circ C$
- Max Heater Output: $q_{max} = 5 \quad kW$
- Heater Efficiency: $\eta = 1$

In all cases set-point temperature is $20^\circ C$ to to improve interpretation. Outdoor temperature is a constant value through each episode, selected randomly at the start of the episode to be between $5^\circ C$ and $15^\circ C$. The initial indoor temperature is selected randomly to be between $18^\circ C$ and $22^\circ C$.

For analysis, episodes are run using different types of importance vectors to illustrate adaptability. For each importance vector, 7 episodes are plotted to show the system's variance in performance.

The first case is shown in fig. 5.2, where the importance vector is the same as in training, ie. fig. 5.1. The mid-day decrease in temperature in all cases demonstrates that the switch in priority is achieved through the importance vector.

The second case is shown in fig 5.3, where the importance vector for set-point tracking objective is set to 1 for the entirety of the episode, indicating that only the set-point tracking advisor is considered. In this case, the temperature tracks $20^\circ C$ with reasonable accuracy for all 7 episodes.

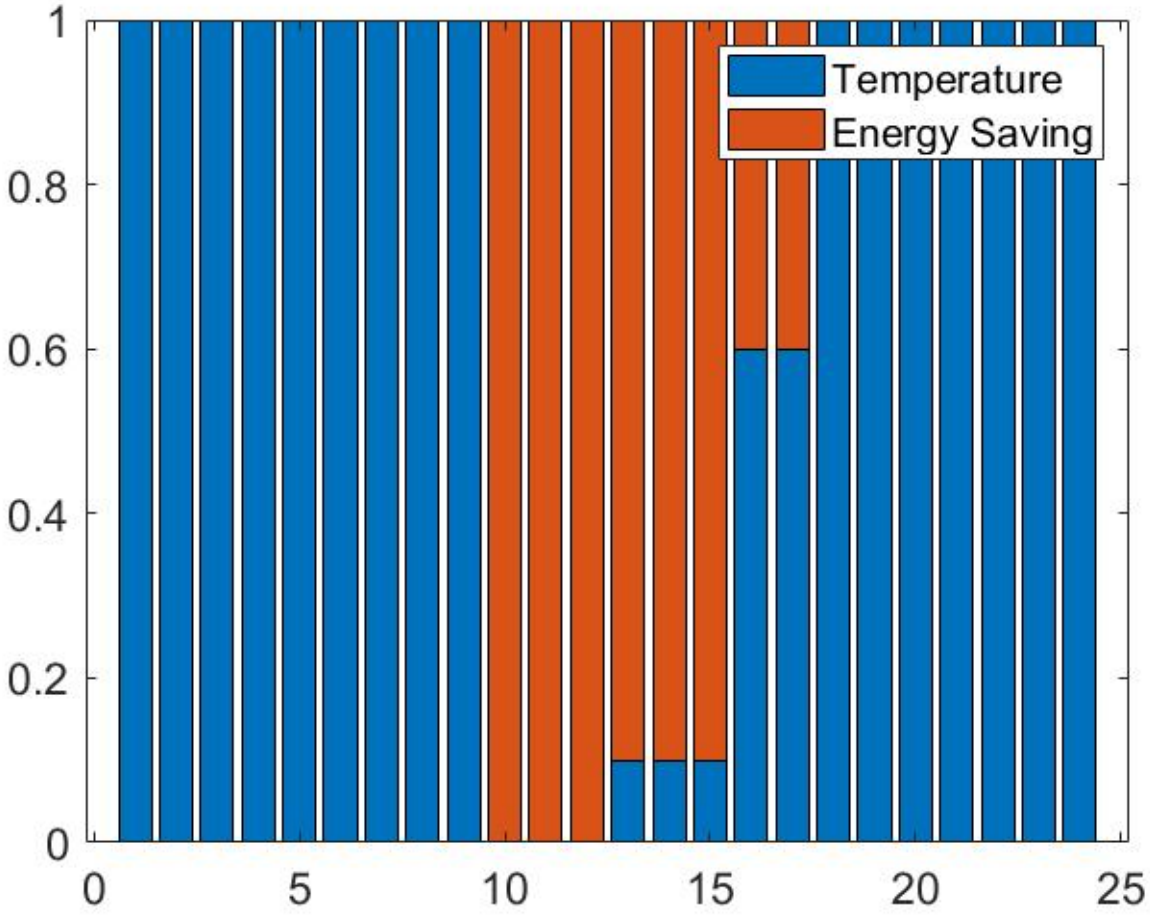


Figure 5.1: Importance of each objective during training

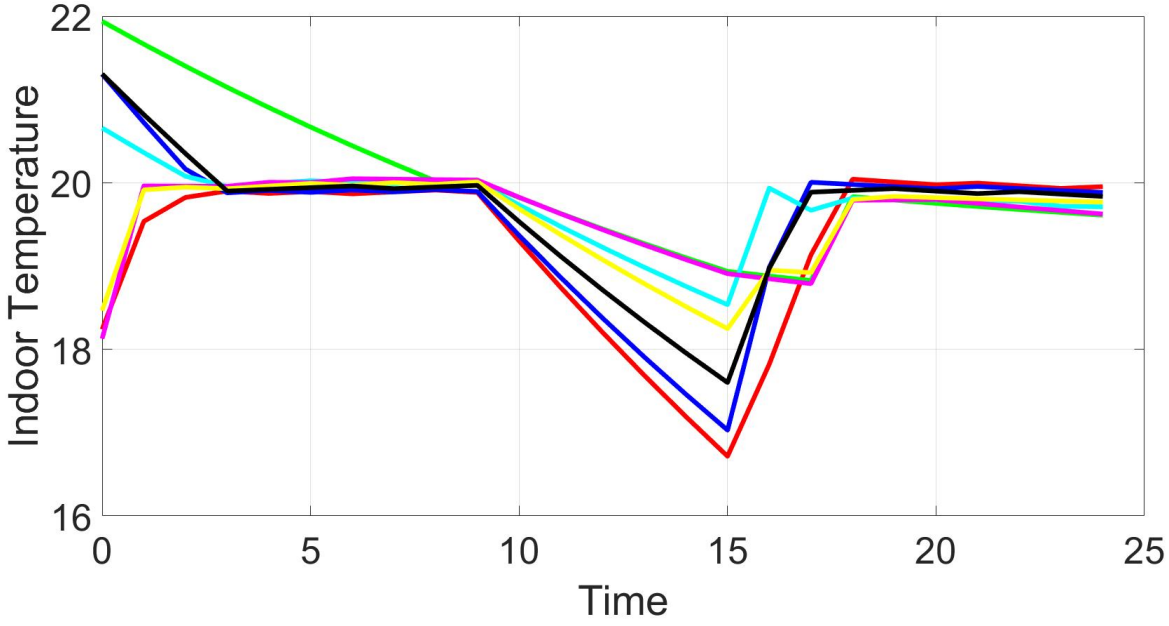


Figure 5.2: Same Importance as during training

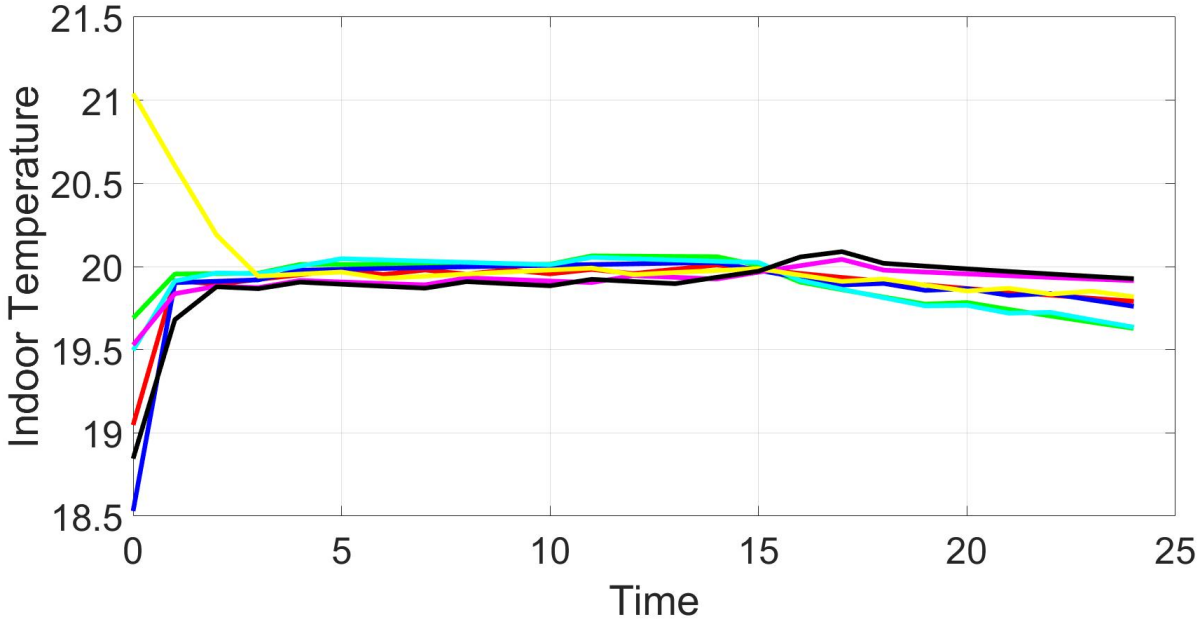


Figure 5.3: Only Temperature Importance

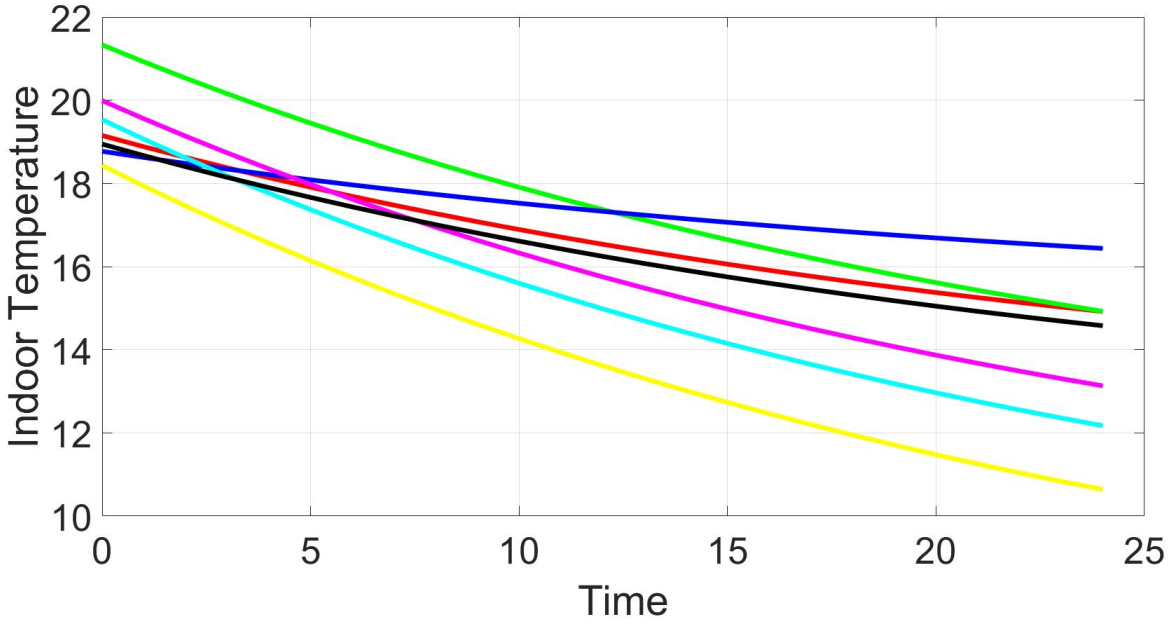


Figure 5.4: Only Energy Saving Importance

The third case is shown in fig 5.4, where the importance vector for energy saving is set to 1 for the entirety of each episode. As expected, the temperature constantly decreases in all cases from the beginning. In this case, the heater uses no energy, as should be expected when only energy consumption minimization is considered as an objective.

Finally, results for equal importance for both consumption minimization and set-point tracking is shown in fig. 5.5. In this case the performance shows more variation during early time-periods. Nonetheless, the temperature seems to stick between $19°C$ and $20°C$, where the average clearly decreases from the set-point of $20°C$, indicating some compromise between the multiple objectives is achieved.

5.4 Conclusion and Future Work

The numerical results indicate that a MAd-RL agent utilizing a DQN to act as the advisor for each objective is capable of balancing multiple objectives, evidenced by the adaptation to different weight schemes. In the TOU pricing setting, the importance vector could be determined based on the hourly energy pricing, enabling the smart-home to automatically adjust to pricing signals.

Future work should include a standardized approach to determining ideal importance vectors. Experimentation using real data / environments should also be considered to better demonstrate feasibility. It may be possible to improve performance by including more information in the state-space. An ex-

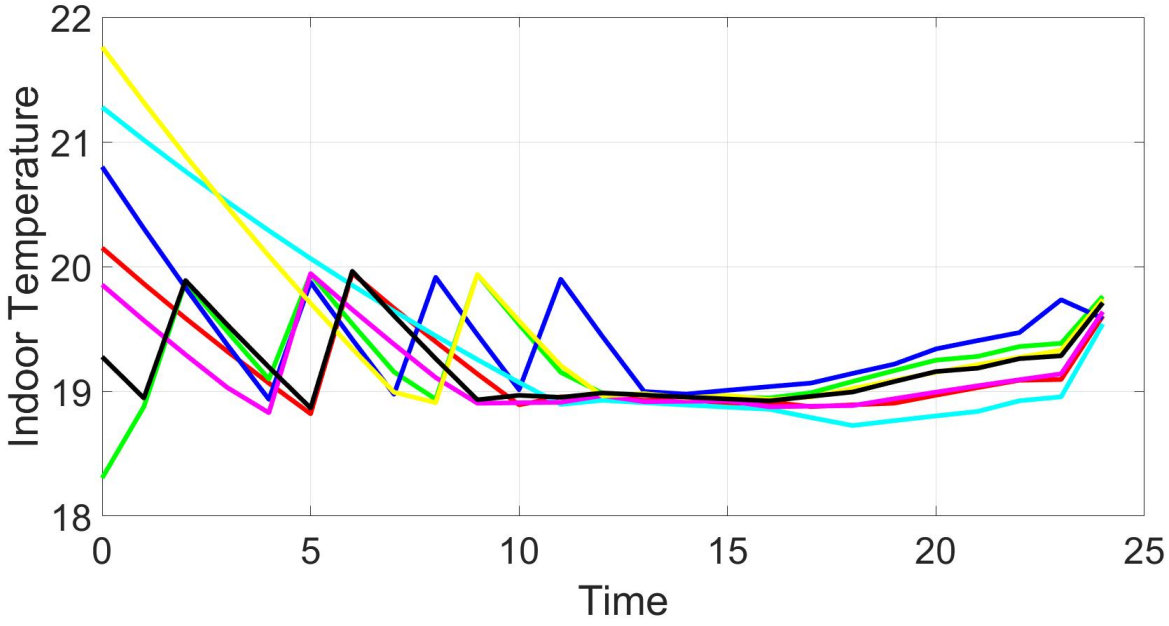


Figure 5.5: Same Importance throughout

ample could be to include the objective's current and possibly future importance to allow the advisor to better consider the intents of the aggregation function when advising actions.

Chapter 6

Multi-Advisor Reinforcement Learning for Multi-Agent Multi-Objective Smart Home Control

6.1 Introduction

In the previous chapter, the feasibility of Multi-Advisor Reinforcement Learning was demonstrated by applying the algorithm to a simple multi-objective control task, that of thermostatically controlling an electric heater. The two objectives considered included temperature set-point tracking and energy consumption minimization. This work demonstrated that MAd-RL allows switching between objectives according to a schedule of weights in the aggregation function. Multiple objectives were achievable simultaneously, and the results demonstrates that the algorithm can easily switch between objectives effectively.

The results presented in the previous chapter are limited by the lack of complexity in the environment model. We increase the complexity of the environment in multiple aspects for the experiments in this chapter, which further demonstrates the feasibility of MAd-RL for control in smart homes.

The previously presented model is deterministic in non-starting states. While this still acceptably falls under the Markov Decision Process formulation, it is limited in it's implications. Generally speaking, stochastic environments are more difficult to solve, as multiple trials must be undertaken by the agent to allow it to consider the variety of outcomes than can occur from an individual action. In this chapter, we include an electric vehicle charging formulation, which includes more stochasticity,

particularly in the state of charge on arrival, the arrival time, and departure time from the home.

Furthermore, this work considers multiple appliances. At minimum, this extends the action-space to include a second dimension. This is the heart of the issue of including multiple appliances at once, the amount of possible joint-actions grows exponentially w.r.t. the action-dimensionality, which limits the scalability of centralized control approaches to such a system. The environment is modelled by including an electric vehicle (EV) charging problem (inspired by [57] which utilizes DQNs for EV charging) as well as the previously described electric heating formulation.

We consider heating and EV charging as two of the most critical devices to control for residential demand response applications. Notably, including another appliance induces a new primary objective for the appliance. The term for this objective is "range anxiety" [57].

Finally, this work improves on the previous formulation by also considering a global objective. This fourth objective corresponds to the common demand-response objective of smart-contracts; a penalty associated with overtaking a contracted threshold [21]. This objective allows potential demonstration of coordination, as the household requires that overall consumption is determined jointly by the actions of all agents. This is of fundamental concern in designing multi-agent learning systems, as the results cannot be achieved by a single agent, both must act in a particular manner to achieve the objective.

6.2 Environment Model

The description of an environment in the reinforcement learning context includes 2 main components; the environment dynamics, ie. the mathematical model that describes how the state (stochastically) evolves according to the given actions. The second main component is the reward structure, which is more notable in the multi-objective setting as it requires a different function for each objective being considered. The reward structure is described as having 6 components, of which 4 represent unique objectives; namely, temperature set-point tracking, range anxiety, energy consumption reduction, and smart contract success. Other primary components necessary to describe the environment include the state-space specification and action-space specification.

6.2.1 State-Space

The state-space used in this environment considers 12 dimensions, each relevant to modelling the dynamics of some smart-home appliance control objective. A clear description of each dimension is elaborated in the following sections on dynamics and rewards, but a quick summary of the 12 elements of the state-space is denoted as follows:

$$S = \{T_{in},\ T_{out},\ T_{set},\ Presence,\ SOC,\ contract,\ E_{th},\ q_{c,1},\ q_{c,2},\ E_c,\ t_{end},\ t\} \tag{6.1}$$

6.2.2 Action-Space

The action space considered in this environment consists of one dimension for each appliance. These single-dimensional actions correspond to the amount of energy used by each of the 2 appliances noted previously. As we utilize a DQN [26] for each of the advisors in the controller, we therefore assume a discrete action-space. This is accomplished by selecting each appliance to have some predefined maximum power output, and then evenly diving that maximum power output into a set of "power modes". Essentially, each action corresponds to a percentage of the maximum allowed power output for it's associated appliance. Under the notation used in this work, the action space is described as:

$$A = \{P_{heater},\ P_{EV}\} \tag{6.2}$$

6.2.3 Dynamics

The dynamics describe the state evolution as a function of the previous state and action taken. In this work, the dynamics considered are of the abstract form:

$$s_{t+1} = f(s_t, a_t, \sigma_t) \tag{6.3}$$

where s_t and a_t represent the state and action taken at timestep t, while σ_t represents some stochastically generated variable that determines the next state. σ_t is generally hidden from the agent.

In this section the dynamics related to each objective in the system are described. These dynamics are described using a sample-based model, ie. the dynamics function (stochastically) returns a next-state for a given state/action pair. This is opposed to a distribution model, which returns a distribution over next-states instead. Sample based models are used because of their simplicity in implementation.

Consumption Reduction Dynamics

The objective corresponding to this section is designed to reduce energy consumption of each device. There are arguably no dynamics induced by the energy consumption objective, as dynamics relate to the evolution of state, and this objective has no associated state variables. This section is noted for completeness, as it seems the dynamics for the environment are most easily described from the perspective of the objective associated with them.

Electric Heating Dynamics

The electric heating dynamics follow the standard first-order model as presented in previous sections [51][52], and is common in other work [39][65][8]. The dynamics related to the electric heating are governed by the following equation (note that superscript 't' denotes the time-step that the reading was observed):

$$T_{in}^{t+1} = \epsilon T_{in}^{t} + (1-\epsilon)(T_{out}^{t} + \eta \frac{P_{heat}^{t}}{A}) \tag{6.4}$$

The state variables $\{T_{in}, T_{out}\}$ are included in this equation, and it describes how T_{in} evolves. The parameters listed[1] include η (unit-less), the efficiency of the heater (generally taken to be 100%). The parameter A is the the overall thermal conductivity, measured in $kW/^{\circ}C$. The last parameter, ϵ (unit-less) is the inertia of the system, and can be calculated for a system depending on the system's time constant, usually denoted τ, and the amount of time between two states, the time-step, Δt. Epsilon is calculated as[2]:

$$\epsilon = e^{-\frac{\Delta t}{\tau}} \tag{6.5}$$

The relevant state variables for these dynamics includes the first 3 elements, namely; $\{T_{in}, T_{out}, T_{set}\}$. These variables represent the current indoor, outdoor, and set-point temperature. The set-point temperature is required to calculate the reward. The outdoor and set-point temperature are treated as exogenous variables, those which are unaffected by the agent's control actions. In our experiments, these variables are treated as a constant. In real systems, these would be derived from sensor readings/user input/weather data.

Electric Vehicle Charging Dynamics

The electric vehicle charging formulation is adapted primarily from the work of [57], which demonstrates DQN performance on the task of charging / discharging an electric vehicle's battery. The model assumes that the user's interactions can be modelled considering three hidden stochastic variables: arrival time, (t_a), departure time (t_d), and state of charge on the battery at arrival time (SOC_a). These three hidden variables are each sampled from their own truncated normal distribution[3] at the beginning of each day. These distributions are specified in this work using 4 parameters; mean, variance, and

[1] These parameters, η, ϵ, A are treated as constants for a given environment implementation.

[2] A source for these calculations can be found in [8]

[3] A truncated normal distribution is a normal distribution where the probabilities of outcomes above or below specific thresholds cannot occur

upper/lower threshholds. Some notable effects of using a truncated distribution are to guarantee arrival and departure occur once daily, and preventing overlap between the two.

The elements of the state-space most relevant to the EV charging dynamics include $\{Presence, SOC, t\}$. $Presence$ is a binary flag denoting whether the user is at home. SOC is a continuous variable (measured in kWh) denoting the current charge on the EV battery. Unsurprisingly, t denotes the current time of day. The variable t is a discrete variable, ie., we consider fixed time-step length, Δt. The variable t evolves as [4]:

$$t \leftarrow t + \Delta t \tag{6.6}$$

The environment determines the variable $Presence$ by checking whether the current time period lies between the stochastically selected arrival time t_a and departure time t_d. Mathematically, this can be represented as follows[5]:

$$Presence^t = \neg((t_d \leq t) \wedge (t \leq t_a)) \tag{6.7}$$

The third state variable of note is SOC, the current charge on the battery. On arrival, it is set to SOC_a, the stochastically selected value representing charge on the battery when the EV arrives at home. The value is limited to not exceed the battery's maximum energy storage (denoted SOC_{max}), and otherwise it increases according to how much energy is put into the battery (assuming no losses). These conditions can be described mathematically as:

$$SOC^t = \begin{cases} SOC_a & t \leq t_a \leq t + \Delta t \\ SOC_{max} & SOC^{t-1} + P_{EV}\Delta t \geq SOC_{max} \\ SOC^{t-1} + P_{EV}\Delta t & otherwise \end{cases} \tag{6.8}$$

Note that the variable SOC continues to evolve when the vehicle isn't present. This is a more-or-less arbitrary decision, as the variable becomes meaningless when the user isn't home. Other options could be to set SOC to zero on departure, and/or enforce that P_{EV} remains zero when the vehicle isn't home. To ease implementation, we allow the variable to continue evolving meaninglessly when the user isn't present.

[4] t is reset to 0 when it reaches 24 (midnight).

[5] Note: $\neg$ and $\wedge$ denote the logical operations 'NOT' and 'AND' respectively

Smart Contract Dynamics

A smart contract in this work is considered as a daily short-term contract between the consumer and the utility company. The user agrees to reduce their energy consumption for a period specified in the contract to below a fixed threshold, assuming some penalty associated with overtaking the threshold. This objective is global, because the sum of each agent's energy consumption determines whether the contract was a success or a failure. Essentially, this model keeps track of the energy consumed while a smart contract is active, and provides a negative reward if the global energy consumption exceeds a maximum threshold.

As this work utilizes reinforcement learning, we must model a smart contract as a MDP. The dynamics follow an abstract logical system dependant on both the utility company's contract specifications and the smarthome's energy consumption. The contract specifications can be modelled considering 3 state variables [6], start time (t_{start}), end time ($t_{end,p}$), and energy threshold ($E_{th,p}$)[7]. The most relevant state-variables in this model are as follows:

$$\{contract,\ E_{th},\ q_{c,1},\ q_{c,2},\ E_c,\ t_{end},\ t\} \tag{6.9}$$

contract is a binary flag that designates whether a contract is currently active or not. It effectively indicates to the agent when a contract has begun. It can be represented mathematically as[8]:

$$contract = (t_{start} \leq t + \Delta t \leq t_{end}) \tag{6.10}$$

E_{th} represents the energy threshold that the smarthome must not exceed when the contract is active. It is considered unspecified when the contract isn't active, and therefore set to zero. When the contract is active, it's set to the threshold for the contract. Mathematically:

$$E_{th}^{t} = \begin{cases} E_{th,p} & contract^{t} = 1 \\ 0 & otherwise \end{cases} \tag{6.11}$$

The variables $\{q_{c,1},\ q_{c,2},\ E_c\}$ each denote a running sum of the energy consumed since the beginning of the contract. $q_{c,i}$ denotes the energy consumed by device 'i' since the beginning of the contract, where $q_{c,1}$ and $q_{c,2}$ refer to the consumption of the heater and the EV charger respectively. E_c denotes the

[6] Contract success is the objective, the aggregation function can account for the magnitude of the financial incentive.

[7] Subscript p denotes environment parameter rather than state variable.

[8] Note that t is the last variable updated, ie. $t + \Delta t$ represents the current time.

total energy consumed by the heater and EV charger since the beginning of the contract. The first can be expressed mathematically as:

$$q_{c,1}^{t+1} = contract^t(q_{c,1}^t + P_{heater}\Delta t) \tag{6.12}$$

Secondly, noting that P_{EV}^* denotes the true power consumption[9] of the EV charger, then $q_{c,2}$ can be described as:

$$q_{c,2}^{t+1} = \begin{cases} contract^t(q_{c,2}^t + P_{EV}^*\Delta t) & presence^t = 1 \\ contract^t q_{c,2}^t & otherwise \end{cases} \tag{6.13}$$

E_c, the total consumed energy, sums each appliance's consumed energy:

$$E_c^t = \sum_{\forall i} q_{c,i}^t = q_{c,1}^t + q_{c,2}^t \tag{6.14}$$

The last state variable to describe, t_{end} denotes when the contract will end. It is assumed zero when no contract is active. Mathematically:

$$t_{end}^t = \begin{cases} t_{end,p} & contract^t = 1 \\ 0 & otherwise \end{cases} \tag{6.15}$$

6.2.4 Dynamics Summary

Overall there are 12 state variables considered. Excluding time t, these can be categorized by their purpose of inclusion. The first 3, $\{T_{in}, T_{out}, T_{set}\}$ are included for their relevance in temperature control, followed by $\{Presence, SOC\}$ which are included for EV charging, and $\{contract, E_{th}, q_{c,1}, q_{c,2}, E_c, t_{end}\}$ which are all included to model the smart contract. These state variables are selected to make the environment obey the markov property, ie, the best predictions can be made based on the state variables given.

6.2.5 Reward Function Specification

The reward functions are selected to represent the objectives as previously described. We consider 4 types of objectives in this work; the inherent goals for the considered appliances supply the first 2, namely, temperature set-point tracking and range anxiety. The remaining 2 objectives are demand-response motivated, they relate to reducing energy consumption. These two objectives represent energy consumption reduction and threshholds for energy consumption.

[9]Correcting for possibility that the prescribed action would overfill the EV. Usually P_{EV}^* is the same as P_{EV}.

These objectives can be organized in various ways, for example, a centralized controller could represent these as just 4 terms in a scalar reward function. A non-joint action selection (multi-agent / decentralized) approach could be represented as 2 scalar reward functions, each with 3 components. In the multi-advisor approach we consider in this work, each advisor is associated with one reward function, for a total of 6 advisors and reward functions. Therefore, we consider the importance-weighted sum of the 6 considered reward functions as the global objective for the system.

Formulations of each of the 6 reward functions are provided in the following sections, organized by the 4 objectives they represent. Range anxiety and set-point tracking each provide a single reward function, while consumption reduction & consumption thresholds provide 2 reward functions each.

These reward functions will follow a double-subscript notation, where the first digit represents the appliance/agent, and the second digit represents the objective. The objectives are ordered as: inherent, reduction, threshold, given indices 1, 2, and 3 respectively. For example, $R_{2,1}$ represents the EV charger's range anxiety objective, and $R_{1,2}$ represents the heater's consumption reduction objective.

Temperature Set-Point Tracking

Set-point tracking refers to the problem of keeping some aspect of the state as close to a pre-specified set-point value as possible. Set-point tracking assumes the preferred temperature is specified by the user. The associated reward function is represented by the squared error from the indoor temperature, which is a common metric for performance in set-point tracking, and is used in the previous chapter. This objective is to be minimized, so a negative coefficient is used. Mathematically, this function is described as:

$$R^t_{1,1} = -(T^t_{in} - T^t_{set})^2 \tag{6.16}$$

Range Anxiety

The range anxiety objective is to have the vehicle fully charged as possible when it's needed by the user. In the simple dynamics model we used, this occurs only once daily (in the morning), and is represented by the departure time. The objective is represented again with a squared error term, but this reward is comparatively more sparse. This is due to the fact that the range anxiety reward is non-zero for a single time-step, the departure time, whereas the set-point tracking objective provides feedback every time-step, making electric vehicle charging a more difficult problem. Mathematically, this reward function

can be described as:

$$R_{2,1}^t = \begin{cases} -10(SOC^t - SOC_{max})^2 & (Presence^{t-1} = 1) \wedge (Presence^t = 0) \\ 0 & otherwise \end{cases} \tag{6.17}$$

Consumption Reduction

The consumption reduction objective is to reduce the energy consumption for every time step. The objective pushes each appliance to consume less energy. It could be increased in importance during periods of high energy cost to account for time-of-use pricing. The objective is formulated per appliance, and provides a scaled punishment for every unit of energy consumed. For the electric heating, this is represented as:

$$R_{1,2} = -10 P_{heater} \Delta t \tag{6.18}$$

Similarly, the electric vehicle charger represents this as[10]:

$$R_{2,2} = -10 P_{EV}^* \Delta t \tag{6.19}$$

Consumption Threshold

The consumption threshold objective represents the case where the utility company forms a smart contract with the smart-home. In this contract, the user receives a financial incentive to decrease their household energy consumption to below a fixed value during for a predetermined interval of time. This is represented by providing a negative reward at the end of the contract if it results in a failure.

This objective is global in the sense that it requires the contribution of each agent to not fail the contract. An issue with global objectives is the credit assignment problem [6]. The issue is that each agent needs to be able to determine whether it's own or others' actions were responsible for a high/low reward. We approach this issue by considering a heuristic for credit assignment. Instead of a simple scaled binary reward representing success or failure, we multiply this by the agent's energy consumption relative to the total consumption. This way, the agents have some idea how their actions affected the total reward. Therefore, we select the reward function for appliance i to be:

$$R_{i,3}^t = \begin{cases} -100 \frac{q_{c,i}^t}{E_c^t} & (contract^t = 1) \wedge (contract^{t+1} = 0) \wedge (E_c^{t+1} \geq E_{th,p}) \\ 0 & otherwise \end{cases} \tag{6.20}$$

[10] P_{EV}^* is same definition as in the smart-contract dynamics.

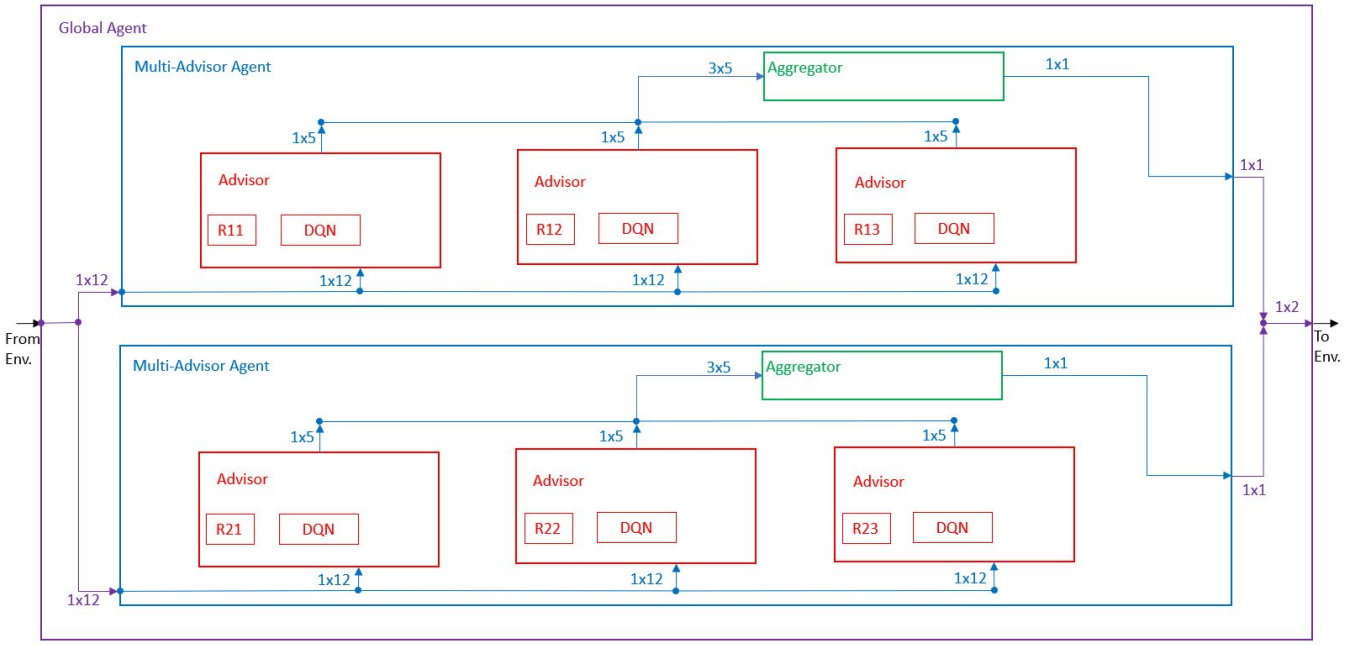


Figure 6.1: Multi-Agent Multi-Advisor Architecture. State is input on the left from the environment, is processed through the agent and it's components, resulting in an action to send to the environment on the right.

6.3 Proposed Architecture

The architecture proposed in this work is multi-agent, multi-advisor, deep reinforcement learning based. Each agent in the multi-agent controller is a multi-advisor agent, and selects actions for it's corresponding appliance. There is one agent for each appliance, and one advisor for each of the appliance's associated reward functions. All learning in the system is performed by the advisors. The architecture for the environment used in this work is shown in Fig. 6.1

Multi-advisor reinforcement learning presented in [20]. This architecture is known to be sub-optimal [42] for 'egocentric' [20] planners (as used here), but similar approaches have demonstrated to be effective [55] in relatively difficult problems. In this architecture, the agent is composed of a set of advisors, each provided with it's own reward function. The advisor estimates the Q-function, $Q(s, a)$, associated with it's own reward signal. The estimated Q-functions are input to the aggregator, which utilizes the estimates to select an action.

The following sections describe the outer-loop of the system, followed by a clarification of the multi-agent aspect of the controller. These are followed by a description of the advisor architecture, and the

aggregation function used by our multi-advisor agent.

6.3.1 Outer Loop

Each agent observes state s from the environment, and passes this information to each of it's advisors. Each advisor uses this to produce it's estimates of $Q(s, a)$ for each available action[11]. For an agents action space of $\mathcal{A}$, there are $|\mathcal{A}|$ actions, and given that the agent has K advisors, there are $|\mathcal{A}| \times K$ scalars $Q(s, a)$ produced in total for the agent[12] (one for each action-advisor pair). These $|\mathcal{A}| \times K$ values are input to the agent's aggregation function f to select the action to be taken:

$$f : \mathbb{R}^{|\mathcal{A}| \times K} \mapsto \mathcal{A} \tag{6.21}$$

This is performed for each agent, and the selected actions are then sent to the environment, which then evolves according to it's dynamics, and provides a new state s to each agent. The cycle then repeats, possibly indefinitely in a real system, or until some conditions are met in simulation.

Algorithm 1: Training Loop

Input : Environment, Agent

Output: Trained Agent

while *Still Training* **do**
- Get initial state s_0 from Environment.reset()
- Get initial action a_0 from Agent.getInitialAction(s_0)
- $t \leftarrow 0$
- **while** *Episode Still Running* **do**
  - $\{s_t, a_t, r_{t+1}, s_{t+1}\} \leftarrow$ Environment.step(s_t, a_t)
  - trainDQNs($s_t, a_t, r_{t+1}, s_{t+1}$)
  - **if** *(rand* $\in \{0, 1\}) \leq \epsilon$ **then**
    - $a_{t+1} \leftarrow$ *random action*
  - **else**
    - $a_{t+1} \leftarrow$ *Agent.getAction(s_{t+1})*
  - **end**
  - $t \leftarrow t + 1$
- **end**

end

[11] The advisor's "learn" operation may be performed here.

[12] For N agents, this is $N \times |\mathcal{A}| \times K$ total estimates produced per step.

6.3.2 Multi-Agent Interactions

The smarthome has 2 appliances, and thus there are 2 distinct action components that a centralized controller would need to select at each time-step. In a multi-agent approach, the action components are each selected by their own agent. In this sense, each appliance is provided with a multi-advisor agent which controls it's actions. The environment evolves according to the actions of both agents, and in this work, we consider that each appliance has access to the entire state-space of the system. Essentially, the multi-agent is responsible for concatenating the actions of the individual agents. This is noted because the software implementation of multi-agent provides an interface for arbitrary agents to select actions.

6.3.3 Advisor Model

The purpose of each advisor is to estimate a Q-function, $Q(s, a)$. This function acts as advice for the agent's decision making on the advisor's objective/reward function. There are many approaches to learn a Q-function, for example, Q-learning [58], SARSA [41], neural fitted Q-learning [38], or DQN [27], indicating that appliance designers using multi-advisor reinforcement learning have a variety of choices for selecting an architecture for an advisor.

In this work, we select DQN [27] as the algorithm to implement each advisor. The performance of DQN [26] is the primary motivator for this choice, as it demonstrated the potential of the algorithm to scale to large/high-dimensional state-spaces. We consider DQN to be the first major success of deep reinforcement learning, and note that many subsequent deep reinforcement learning algorithms [22] [54] [25] [35] have arisen since.

Implementation of DQN relies on MATLAB's deep reinforcement learning toolbox [37]. For simplicity, each advisor is given the same neural network architecture. This architecture is composed of a set of fully connected layers, each with the same number of neurons, followed by an output layer with a single neuron which predicts the action value. These networks take as input both the state and action, and output a scalar which estimates $Q(s, a)$ for the advisor's reward. Each fully-connected layer is followed by a ReLU activation function.

The basic fully-connected networks are appropriate for the task at hand, as the state vector is carefully designed as to obey the Markov property. Notable alternative network architectures include convolutional and recurrent layers. Generally, convolutional architectures are effective for image related tasks, quite different from our task. Recurrent networks provide memory to the agent, which potentially equips it to handle partially observable environments. This was notably demonstrated in [17], which enables the agent to learn to play Atari while only observing a single game-frame at a time.

6.3.4 Aggregation Function

The aggregation function is responsible for taking the advice of each advisor, and aggregating that into a single action. The selected aggregation function is inspired by our previous work on importance scaling [51], where we demonstrated that individual smarthome performance is dependant on the user's selected importance for each objective. We design the aggregator assuming that action-value functions are similar in nature to utility functions provided.

The aggregation function can be described in 4 steps. First, each advisor's action-value function is normalized[13] to contain values between 0 and 1. Second, the aggregator then identifies the importance (weight) of each advisor's objective. Third, the normalized values are aggregated using the importance weights into a single scalar for each action. Finally, the action selected is that with the highest aggregated value.

The normalization approach is linear, similar to our previous work [52]. The worst action will have value 0 and the best action will have value 1. The normalized values for advisor i, denoted $Q^i_{norm}(s,a)$[14], are calculated as follows:

$$Q^i_{norm}(s,a) = \frac{Q^i(s,a) - \min_a Q^i(s,a)}{\max_a Q^i(s,a) - \min_a Q^i(s,a)} \tag{6.22}$$

These $Q^i_{norm}(s,a)$ need to be scaled according to their importance. For this section, we can denote the weights as $w_{j,i}$, where $j \in \{1,2\}$ denotes the agent number, and $i \in \{1,2,3\}$ denotes the objective number. Objective i of agent j has an importance $w_{j,i}$ which is considered a function of the state, ie. $w_{j,i}(s)$. Agent j may call this function to determine the importance of objective i in the given state. This importance function[15] is heuristically based and stationary. We assume this fits the definition of 'any fixed aggregator' from theorem 1 in [20]. The following heuristics are used to design the importance-weight function:

- 1. The importance weights considered for a single agent sum to 1.
- 2. Normally, primary objectives are most important, followed by energy reduction and thresholds respectively.
- 3. During user absence, temperature set-point tracking is not very important.
- 4. During a smart contract, succeeding in the contract is important.

[13] This accounts for the fact that reward may have different scales/units.

[14] $Q^i_{norm}(s,a)$ can be considered as a row vector with $|\mathcal{A}|$ elements.

[15] The importance is a hard-coded function of state.

The importance weights follow some predefined default values when no special conditions are present. The default settings assign the weights to be $\{0.7, 0.2, 0.1\}$ for the primary, energy reduction, and energy threshold objectives respectively. These elements are individually reassigned based on the last 2 heuristics, and finally the resulting set of weights for each agent is normalized to sum to 1. The reassignment for heuristic 3 is as follows:

$$w_{1,1}(s_t) \leftarrow 0.2, \quad presence^t = 1 \tag{6.23}$$

For heuristic 4, the reassignment is:

$$w_{j,3}(s_t) \leftarrow 1, \quad contract^t = 1 \tag{6.24}$$

The normalization is performed as the last step (after reassignment):

$$w_{j,i}^{norm} = \frac{w_{j,i}}{\sum_i w_{j,i}} \tag{6.25}$$

A dot-product is performed by the agent for each action. The dot-product is between advisor's value $Q^i(s,a)$ and the advisor's importance $w_{j,i}^{norm}$ across index i. This dot-product results in a scalar for each action, denoted $Q_{sum}(s,a)$. Mathematically this is:

$$Q_{sum}(s,a) = \sum_i w_{j,i}^{norm}(s) Q_{norm}^i(s,a) \tag{6.26}$$

Finally, the non-exploratory action selection is greedy with respect to the aggregated action-values:

$$a_t = argmax_a Q_{sum}(s_t, a) \tag{6.27}$$

Overall, the proposed architecture can be viewed as assigning one agent to each appliance. Each appliance's policy is determined by aggregating an estimation of each objective's action-value.

Algorithm 2: Multi Advisor Action Selection

Input : Set of advisors Ad, Set of actions A, Current State s

Output : Selected action, a_t

Initialize $Q^i(s,a) = 0 \quad \forall \{i,a\} \in \{Ad, A\}$

foreach $i \in \{0, N\} \in \mathbf{Z}$ **do**

  foreach $a \in A$ **do**

    $Q^i(s,a) \leftarrow Ad^i.evaluate(s,a)$

  end

  $a_t = aggregate(Q)$

end

Algorithm 3: Aggregation Function

Input : Action values from all advisors $Q^i(s_t, a) \in \mathbf{R}^{N \times |A|}$

Output : Selected action, a_t

Parameters: Weights w_i for each objective i, $W \in \mathbf{R}^N$

$Q^i_{norm}(s,a) = \frac{Q^i(s,a) - \min_a Q^i(s,a)}{\max_a Q^i(s,a) - \min_a Q^i(s,a)}$

$Q_{sum}(s,a) = \sum_i w^{norm}_{i,j}(s) Q^i_{norm}(s,a)$

$a_t = argmax_a Q_{sum}(s_t, a)$

6.4 Complexity Analysis

Computation of concern for this algorithm is that of action-selection. The complexity analysis therefore considers the question "How long does it take for an agent to select an action after it observes state s?". We assert that the computation time, assuming it takes $O(F)$ to complete a forward pass in a neural network, for a centrallized, decentrallized, and decentralized multi-advisor agent architecture follow the results of equations 6.28, 6.29, and 6.30 respectively. We note that action-selection is linear w.r.t. number of objectives, K, and number of agents, N, in the proposed architecture.

$$O(\text{action selection - Centralized}) = O(|a_i|^N F) \tag{6.28}$$

$$O(\text{action selection - Decentralized}) = O(N|a_i|F) \tag{6.29}$$

$$O(\text{action selection - Multi-Advisor Decentralized}) = O(NK|a_i|F) \tag{6.30}$$

Equations 6.28, 6.29, and 6.30 follow from considering that it takes one forward pass of the DQN for each action being considered. If each action-dimension has $|a_i|$ elements, in the centralized case there are $|a_i|^N$ actions in total to consider. Therefore, at one forward pass per action, this will take $O(|a_i|^N F)$.

In the decentrallized case, there is one DQN-agent selecting each action dimension. Therefore, each agent requires $|a_i|$ forward passes, giving $O(|a_i|F)$ computation required per agent, and $O(N|a_i|F)$ in total.

For the multi-advisor case, in each agent there are K DQN-advisors, each of which considers the actions for it's agent. Therefore, each advisor requires $O(|a_i|F)$ to compute action values, it takes

$O(K|a_i|F)$ to calculate all the action values for a single agent, and $O(NK|a_i|F)$ for computation in total.

6.5 Experimental Results

We assess the performance of the multi-advisor system by performing a series of experiments on some representations of the architecture and it's interactions with the defined environment. This begins with demonstrating the computational scalability of the agent, followed by training curves of the proposed agent, benchmarked against some alternatives. Next is simulation results, demonstrating typical behaviour of the agent and benchmarks, at first using the same weight-scheme the agent was trained on, followed by some analysis of performance under modified weight functions.

6.5.1 Execution Time

To further solidify the theoretical results presented in section 6.4, we measure the computation time required for "agent.learn" operation, which includes both the action-selection and learning procedures for the agent. This extends the theoretical work to include the training step of the agent. This is performed using smaller DQNs[16] as advisors for tractability. Each implementation of the agent performs the 'learn' operation with placeholder data, and is repeated for 2000 iterations[17]. In the two experiments performed, the number of agents and objectives are individually varied from 2 to 20, and the resulting execution time is plotted. A line of best fit is included for reference. These results are shown in figures 6.2 and 6.3 respectively. These results indicate that for modest numbers of agents/objectives, the multi-advisor agent scales linearly with respect to both number of agents and number of objectives.

6.5.2 Environment Parameters

The Environment parameters selected for training and simulation are described in this section, which includes parameters for the dynamics followed by the initial state-distribution.

Dynamics Parameters

The length of a time-step, Δt is 1 hour. The parameterized thermal dynamics are given by:

$$T_{in}^{t+1} = e^{-\frac{\Delta t}{\tau}} T_{in}^{t} + (1 - e^{-\frac{\Delta t}{\tau}})(T_{out}^{t} + \eta \frac{P_{heat}^{t}}{A}) \tag{6.31}$$

$$T_{in}^{t+1} = e^{-\frac{1}{25}} T_{in}^{t} + (1 - e^{-\frac{1}{25}})(T_{out}^{t} + 1 \frac{P_{heat}^{t}}{0.0778}) \tag{6.32}$$

[16] This includes 1 hidden layer with only 4 neurons, 2 actions/dimension, 2 agents, and 2 objectives for each agent.

[17] 2000 is selected as the appropriate number of iterations to reach convergence, based on simple experiments. This damps out the speedup results from primarily selecting exploratory actions

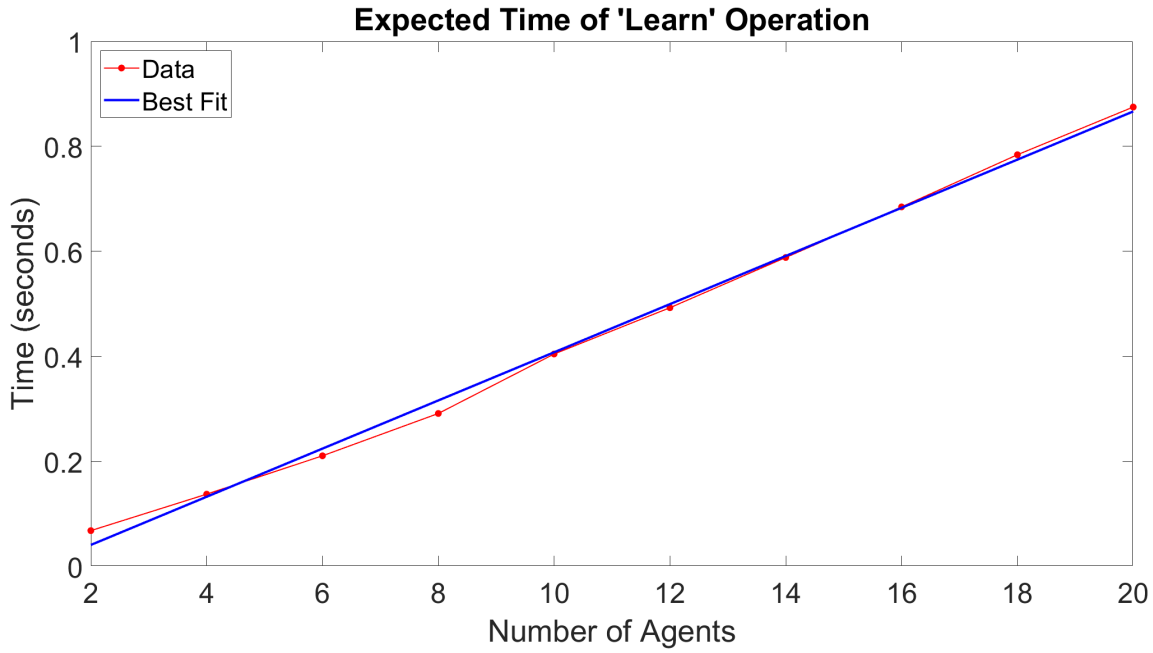


Figure 6.2: Execution Time of 'agent.learn' operation of multi-advisor agent with variable number of agents

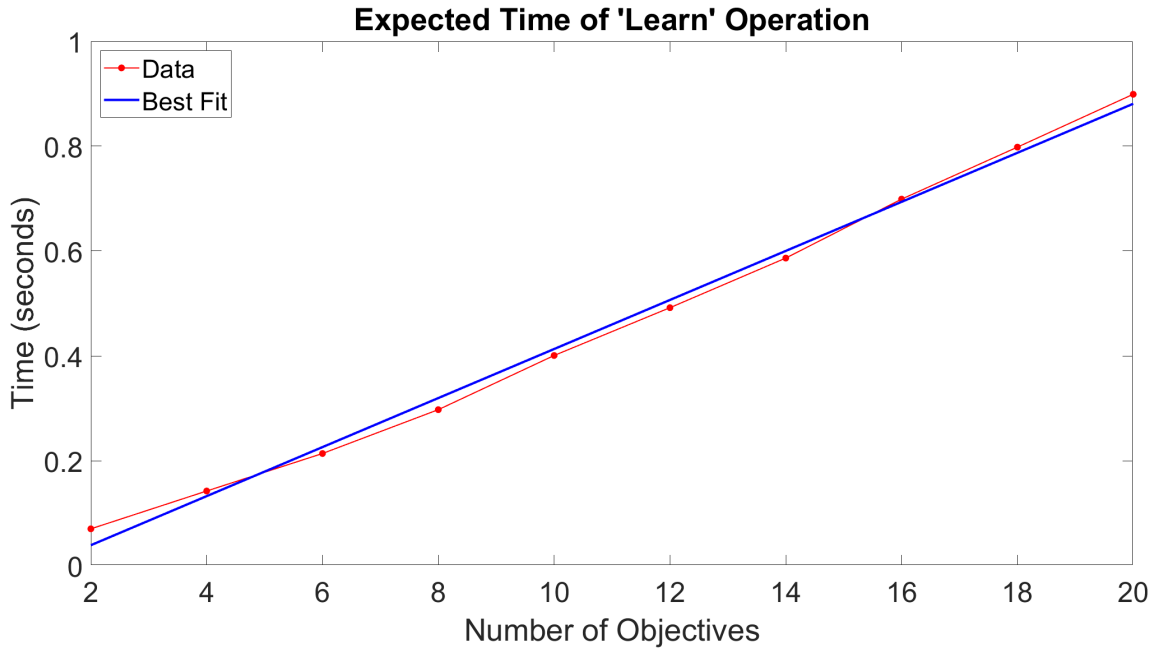


Figure 6.3: Execution time of 'agent.learn' operation of multi-advisor agent with variable number of objectives

The maximum charge on the EV battery is 24 kWh. The stochastic hidden variables, $\{t_a, t_d, SOC_a\}$ are each sampled from a truncatednormal distribution (daily, at midnight). Denoting a truncated normal as $\tilde{N}(\mu, \sigma, L, T)$, (inputs are: mean, variance, lower threshold, and upper threshold respectively) then the hidden variables are sampled as follows[18]:

$$t_a \sim \tilde{N}(18, 1, 15, 21) \tag{6.33}$$

$$t_d \sim \tilde{N}(8, 1, 6, 11) \tag{6.34}$$

$$SOC_a \sim \tilde{N}(.5, .1, .2, .8) \tag{6.35}$$

In regards to the smart contract parameters, t_{start} is 18, and t_{end} is 20. The maximum energy threshold, $E_{th,p}$ is 4 kWh.

Initial State Distribution

The initial state of an episode s_0 follows the distribution:[19]:

$$s_0 = \begin{cases} T_{in}^0 \sim 18 + unif(0,4) \\ T_{out}^0 \sim unif(0,20) - 10 \\ T_{set}^0 = 20 \\ presence^0 = 1 \\ SOC^0 = 0.5 SOC_{max} \\ contract^0 = 0 \\ E_{th}^0 = 0 \\ E_c^0 = 0 \\ q_{c,1}^0 = 0 \\ q_{c,2}^0 = 0 \\ t_{end}^0 = 0 \\ t^0 = 0 \end{cases} \tag{6.36}$$

[18] $\sim$ denotes 'sampled from'

[19] Unif(a,b) denotes uniform distribution between a and b.

6.5.3 Training

To verify the performance of the multi-advisor system, we select a concrete network for each of the advisors, and train them in the environment. We compare the multi-advisor system with 2 alternate DQN-based controller architectures, one centralized and one decentralized. The centralized approach uses a single-network DQN by considering the actions jointly. The decentralized approach uses two networks, each DQN is associated with one appliance's actions.

Performance Metric

As a performance metric, we consider the weighted reward, which we define as [20]:

$$R(s_t, a_t, s_{t+1}) = \sum_{i,j} w_{i,j}(s_t) R_{i,j}(s_t, a_t, s_{t+1}) \tag{6.37}$$

Naturally, the performance metric provided in equation 6.37 is provided as a reward-signal for the centrallized agent. In the multi-agent case, each agent j receives a reward corresponding to the weighted objectives, defined as:

$$R_j(s_t, a_t, s_{t+1}) = \sum_{i} w_{i,j}(s_t) R_{i,j}(s_t, a_t, s_{t+1}) \tag{6.38}$$

Agent Hyperparameters

Each DQN (including advisors) use the same network and agent hyperparameters. These hyperparameters were chosen based on simple trial-and-error experimentation. Hyper parameters for the DQN follow MATLAB defaults, except learning weight, set to 0.001 (normally 0.01) mini-batch size, set to 32 (normally 64), target update frequency, set to 10 (normally 4) and minimum ϵ, set to 0.001 (normally 0.01). Each network has 3 fully connected layers, 16 neurons each, followed by an output layer with a single neuron (the network outputs a scalar). The maximum and minimum values of $\{P_{heater},\ P_{EV}\}$ are $\{5,\ 4\}$ and $\{0,\ 0\}$ kWh respectively. Each appliance is allowed 5 discrete actions, evenly spaced and including boundaries.

Training Results

The centralized, decentralized, and multi-advisor decentralized controllers are trained for 1000 episodes each. A standard episode lasts 3 days, amounting to 72 time-steps in total. Training curves for each

[20] This seems to not be an ideal metric for the system.

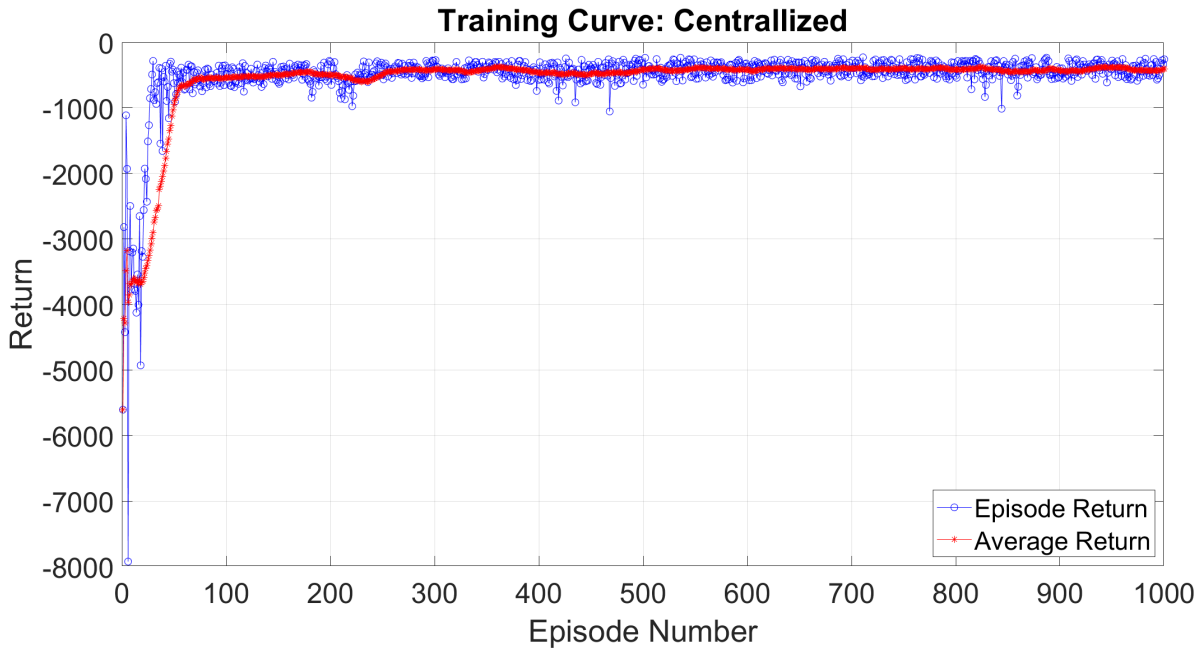


Figure 6.4: Training Curve for centralized controller

agent are shown in figures 6.4, 6.5, and 6.6 respectively. The blue curve represents the return for each episode, while the red curve represents the average return, averaged over the previous 30 episodes. A plot of average return (red curves) for the last 500 training episodes is provided for each agent in figure 6.7. Boxplots for episode return (blue curves) distributions are shown in figure 6.8. From these results, it seems that the multi-advisor agent exhibits comparable performance to the centrallized and decentrallized agents.

6.5.4 Simulation

In this section, we simulate the three previously trained agents with the specified environment parameters. The indoor temperature and SOC on the EV are plotted for typical days of all 3 agents in figures 6.9, 6.10, 6.11, 6.15, 6.16, 6.17. A boxplot distribution of each is provided in figures 6.12, 6.13, 6.14, 6.18, 6.19, 6.20. Finally, a bar-graph including both the smart-contract failure rate and the percentage charge on the EV battery and provided in figure 6.21.

Analysis

The behaviour of the multi-advisor agent is notably different from that of the benchmark agents. The 4 objectives noted for this work include temperature control, range anxiety, consumption reduction,

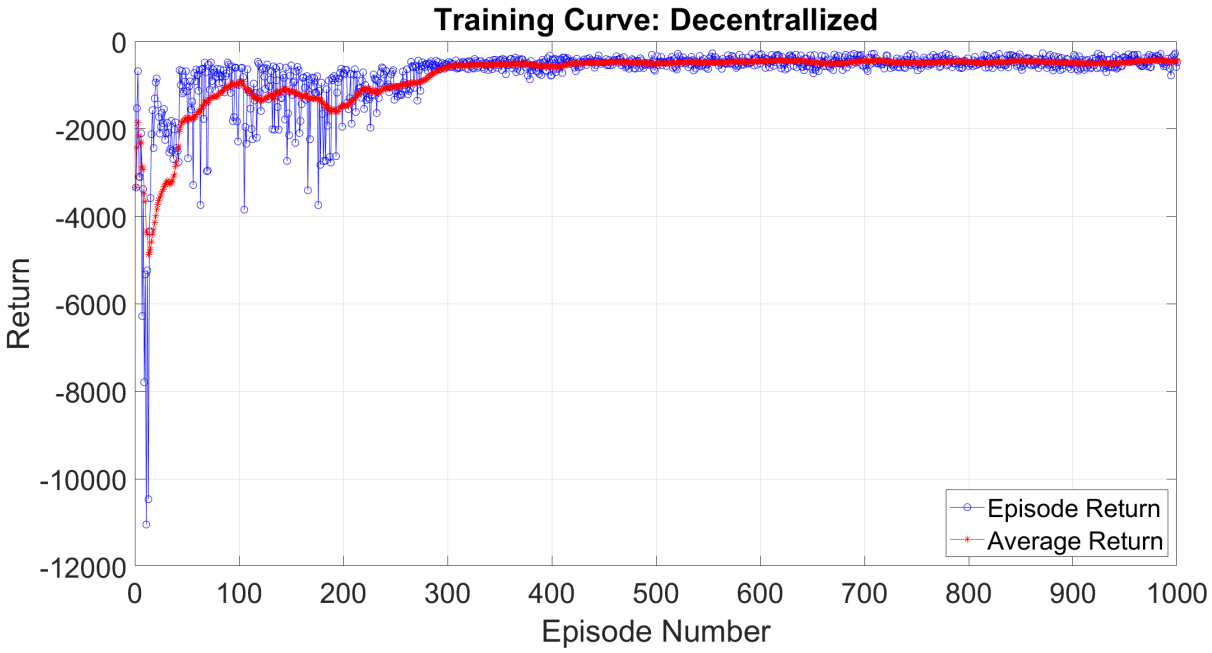


Figure 6.5: Training Curve for decentralized controller

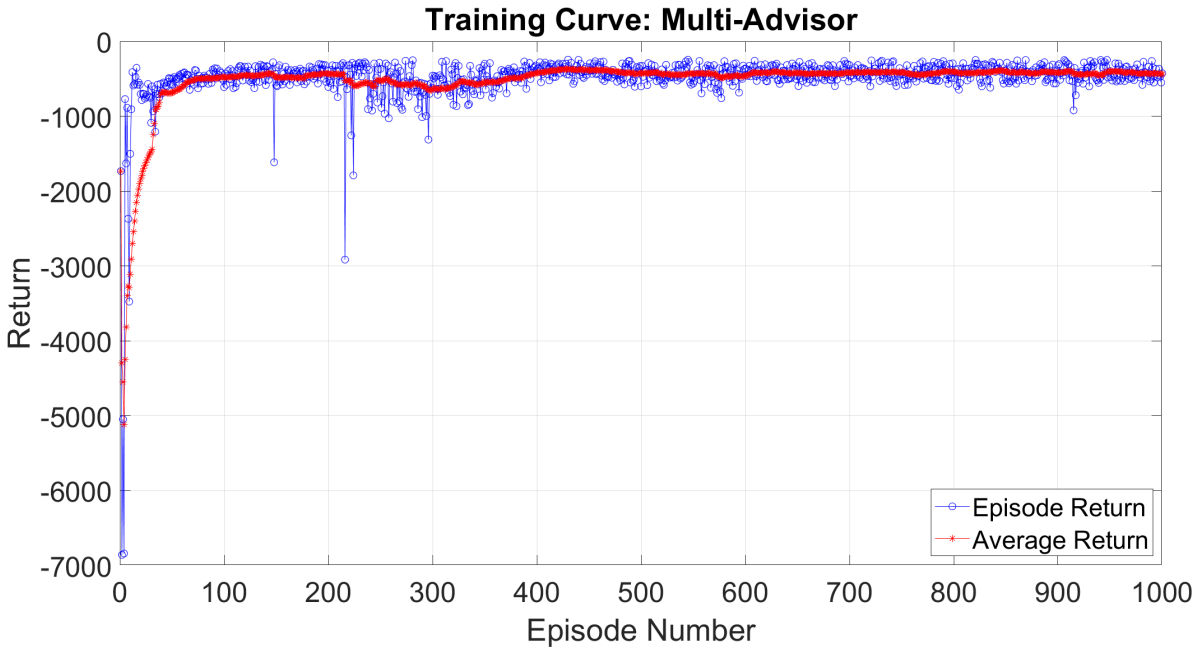


Figure 6.6: Training Curve for multi-advisor multi-agent controller

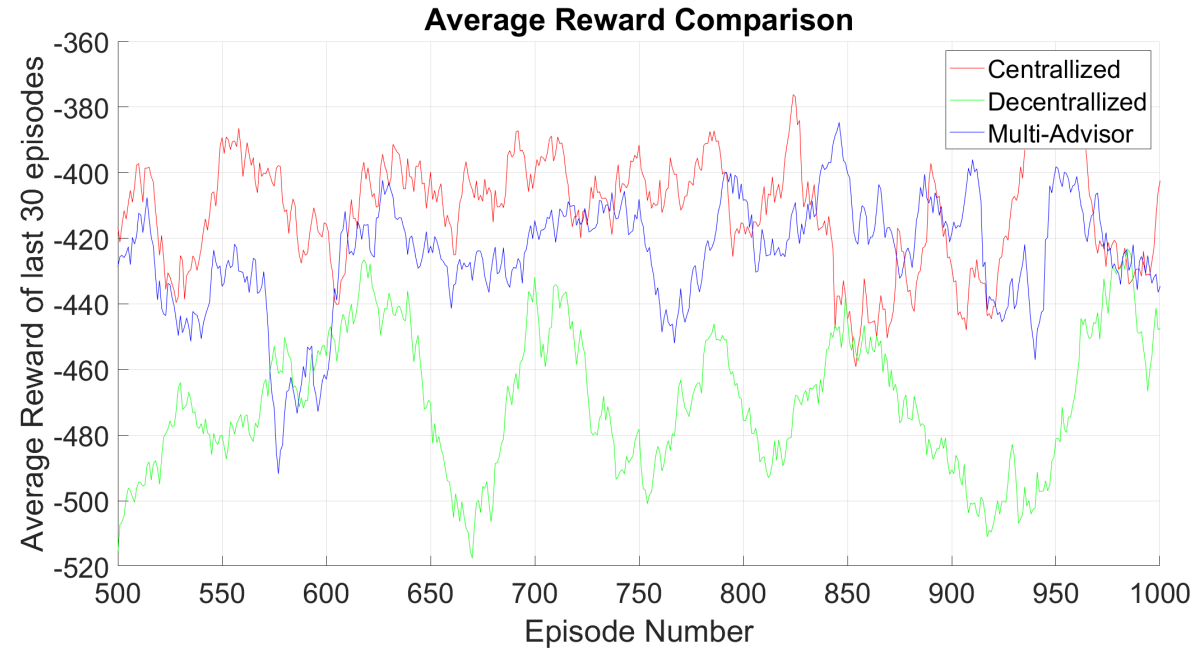


Figure 6.7: Average Return of episodes 500-1000 for each agent.

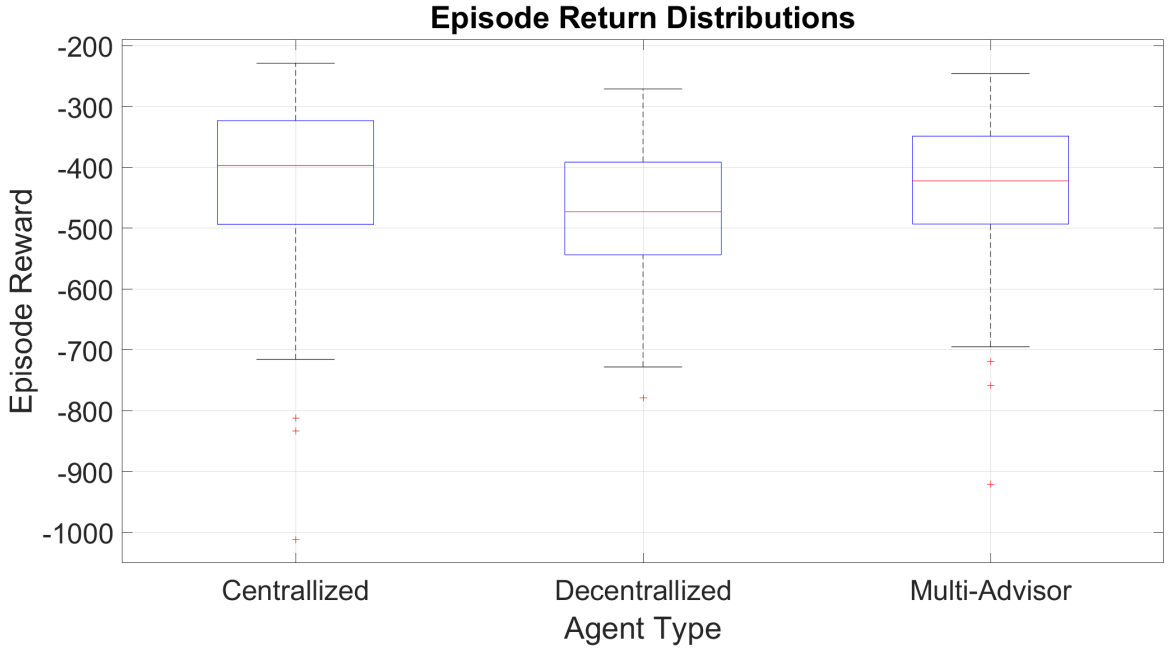


Figure 6.8: Boxplots of episode returns for episodes 500-1000 of training.

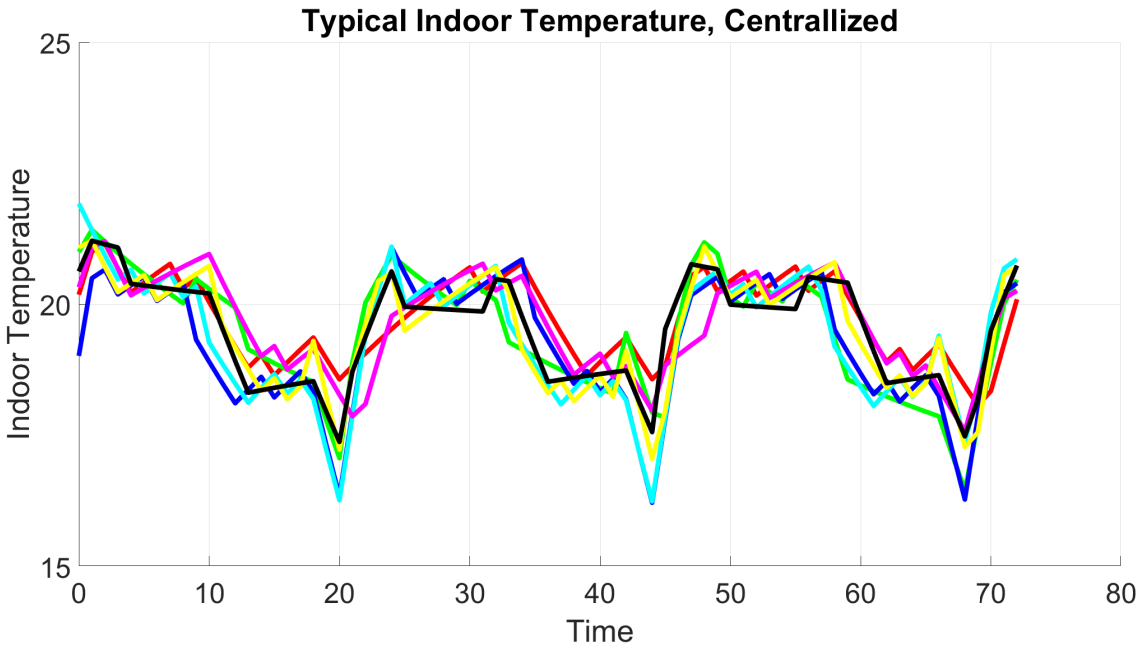


Figure 6.9: 7 episodes of indoor temperature with centrallized agent.

and smart-contract success. All 3 agents perform EV charging perfectly, where the EV is always fully charged by the departure time. The multi-advisor agents performs the strongest in terms of temperature control, while performing the worst in terms of smart contract failure. Despite this, the failure rate is relatively small at approximately 20%. For reference, a uniformly stochastic policy (randomly selected actions) has a failure rate of approximately 94%.

6.5.5 Simulation - Different Importance Weights

To further assess the multi-advisor agent, we consider the effects of changing the weight function from the prespecified case as described in section 6.3.4. In these simulations, we select the weight function to be independent of the state, and therefore each weight is a fixed value throughout the episode. To demonstrate performance, we randomly select these weights[21] prior to each episode. These experiments are performed 1000 times for each agent. A boxplot representing the newly-weighted reward distribution for each agent is provided in figure 6.22, and is rescaled in figure 6.23. Notably, the weighted reward of the multi-advisor agent exhibits more variance, contains more negative outliers, and has the (marginally) worst median performance.

To clarify the worsening performance of the multi-advisor agent, consider the indoor temperature of

[21] The weights sum to 1 for each agent.

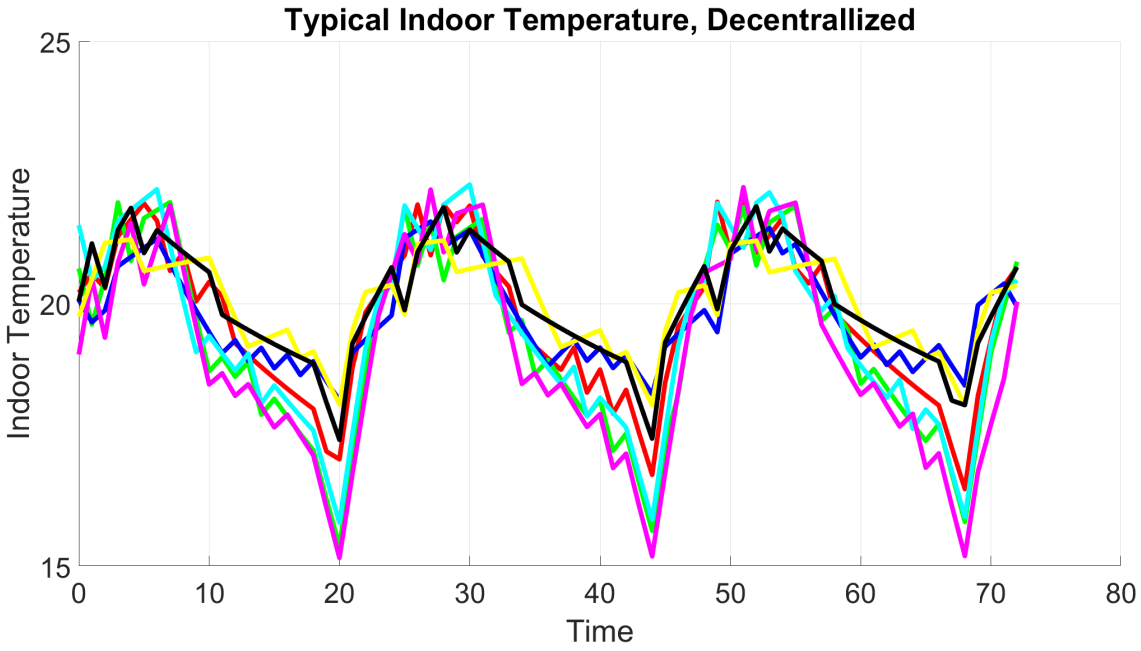


Figure 6.10: 7 episodes of indoor temperature with decentrallized agent.

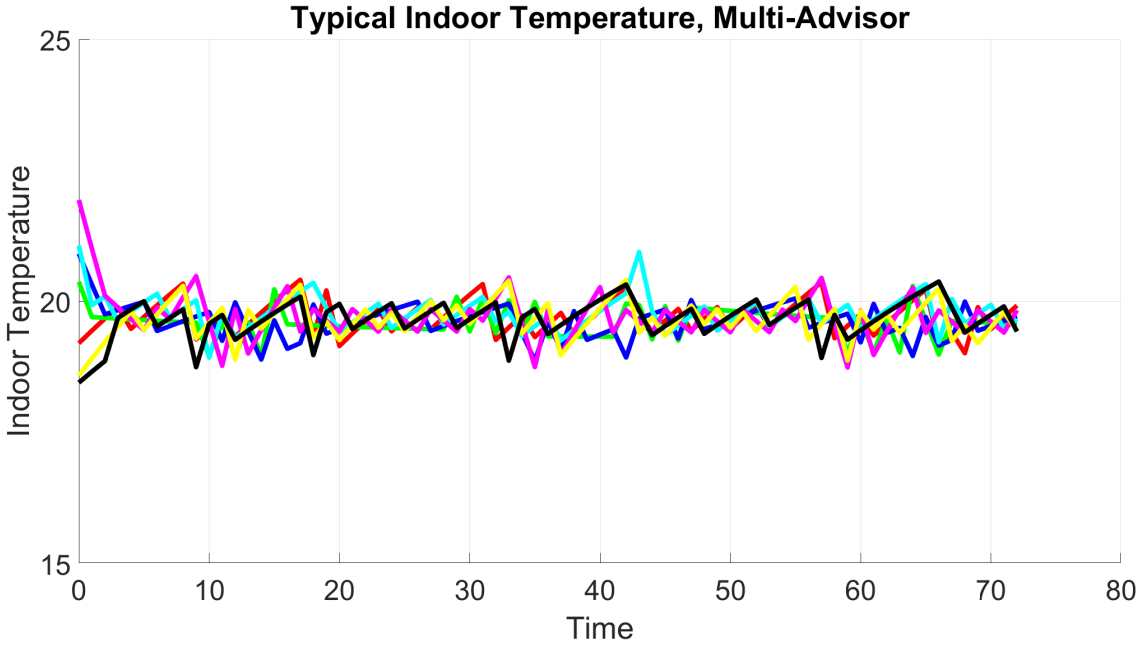


Figure 6.11: 7 episodes of indoor temperature with multi-advisor agent.

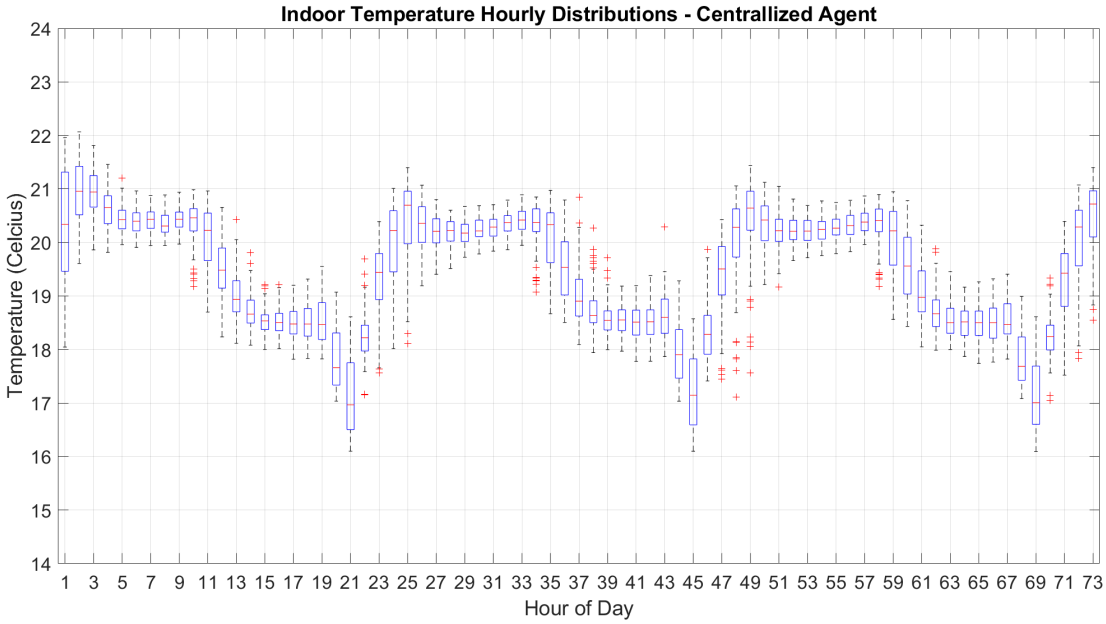


Figure 6.12: Hourly boxplots of indoor temperature with a centrallized agent

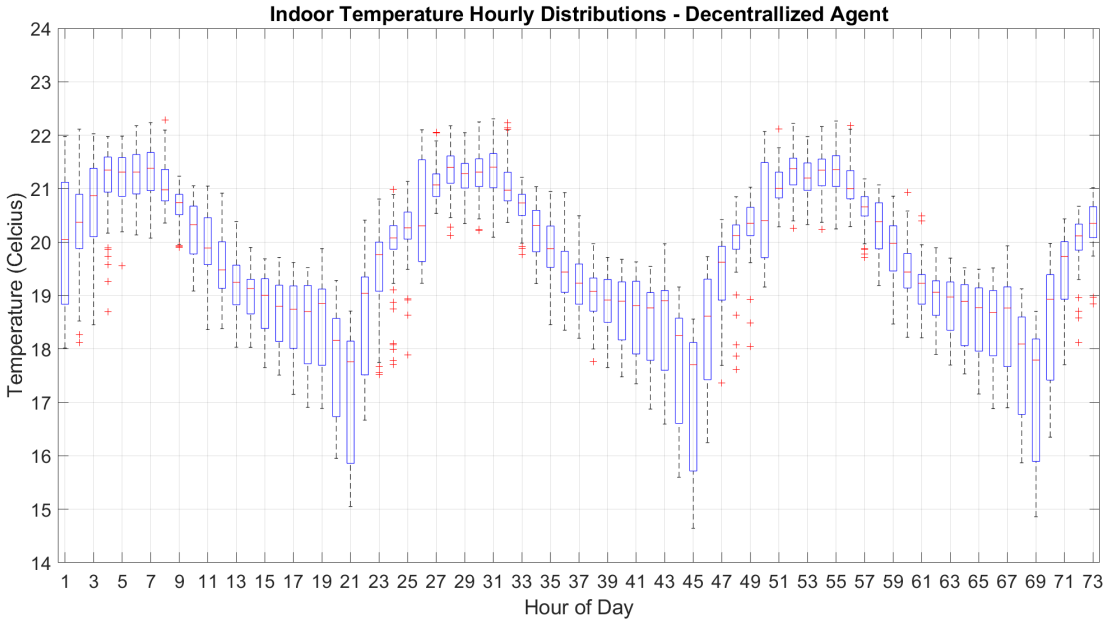


Figure 6.13: Hourly boxplots of indoor temperature with a decentrallized agent

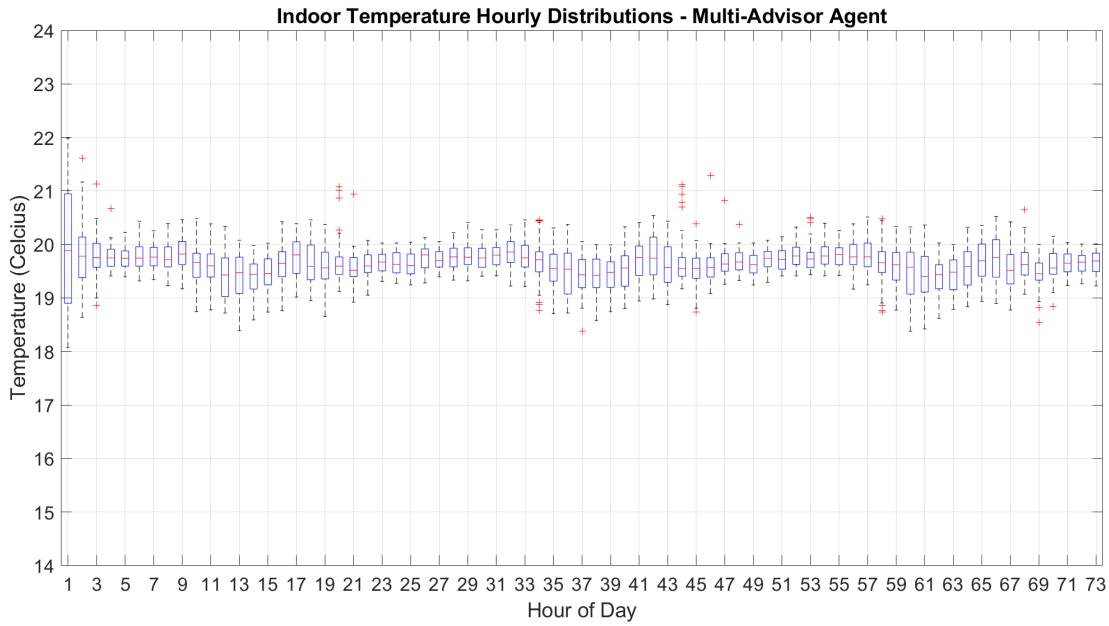


Figure 6.14: Hourly boxplots of indoor temperature with a multi-advisor agent

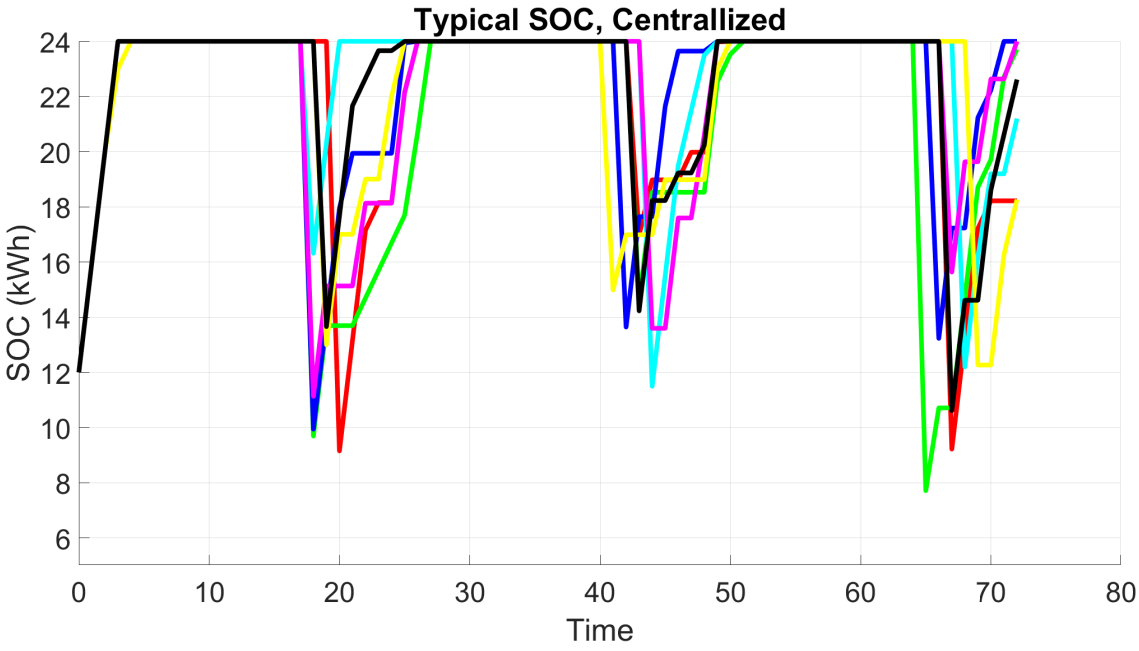


Figure 6.15: 7 episodes of SOC with centrallized agent.

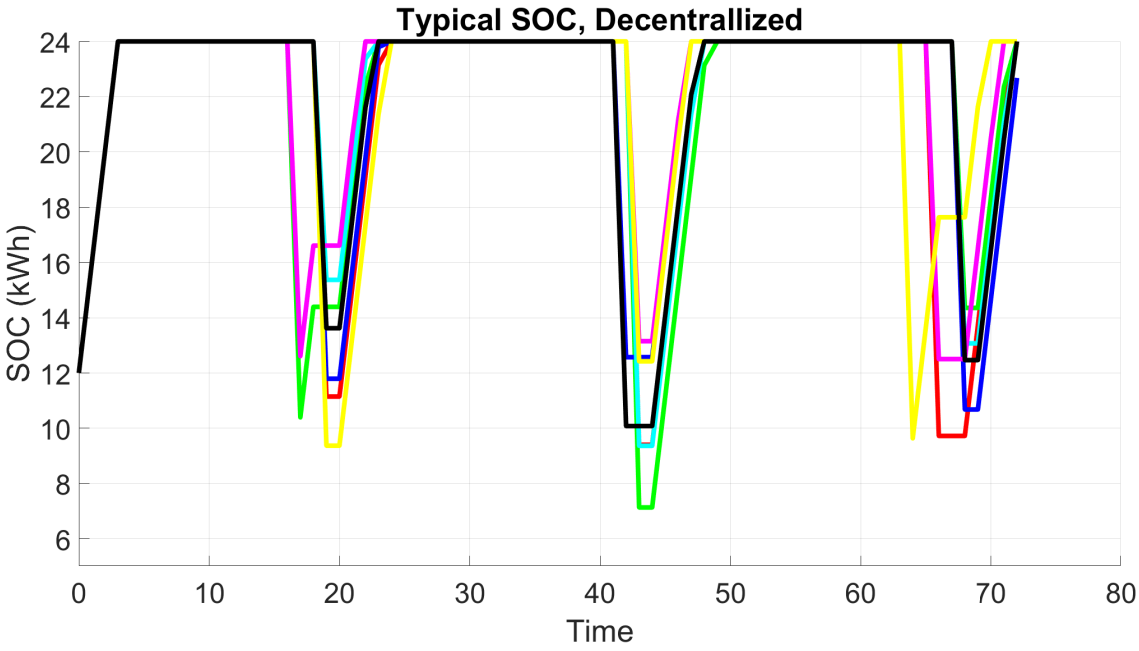


Figure 6.16: 7 episodes of SOC with decentrallized agent.

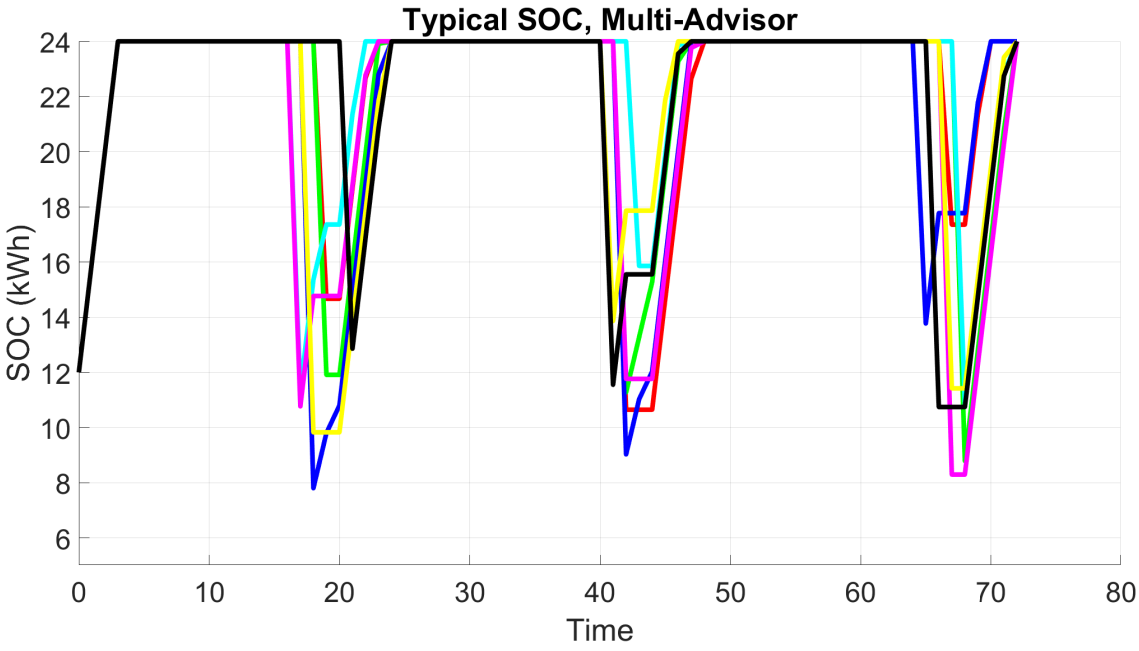


Figure 6.17: 7 episodes of SOC with multi-advisor agent.

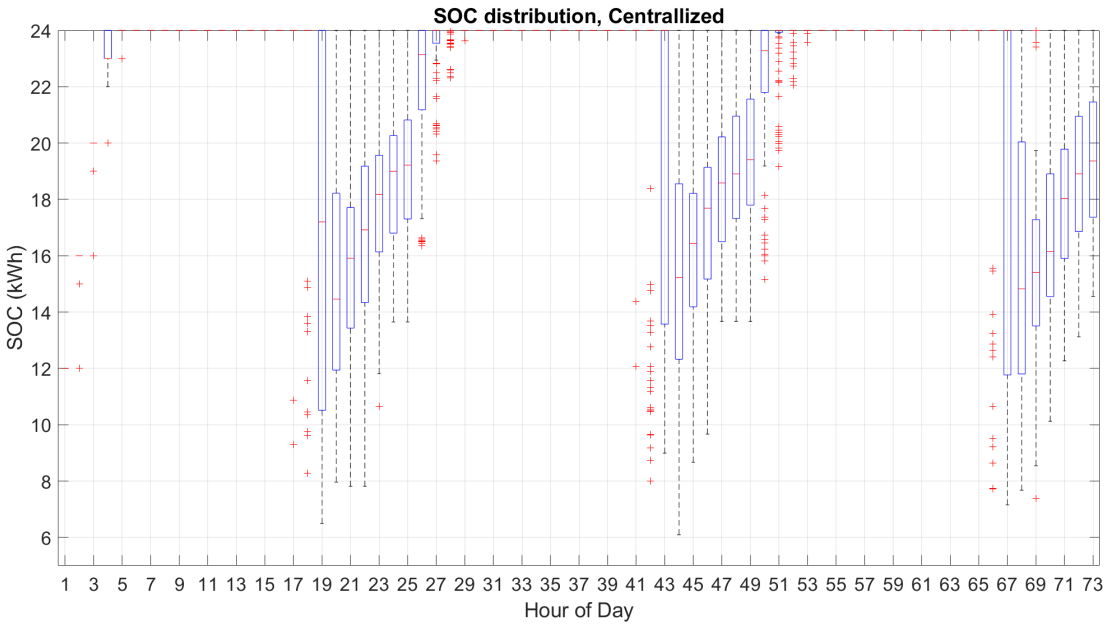


Figure 6.18: Hourly boxplots of indoor temperature with a centrallized agent

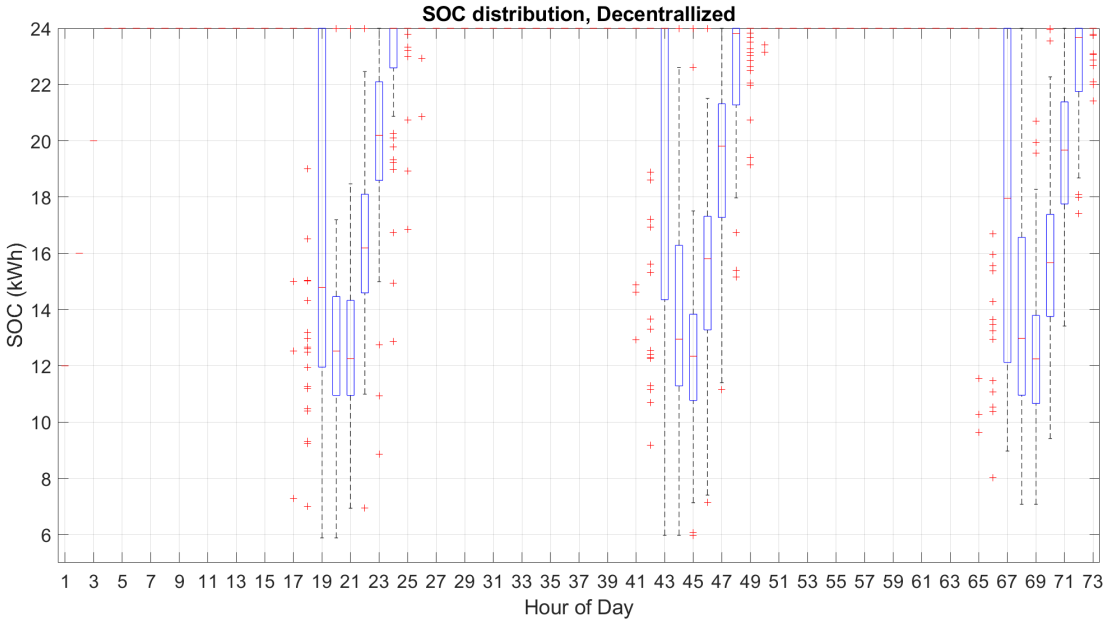


Figure 6.19: Hourly boxplots of indoor temperature with a decentrallized agent

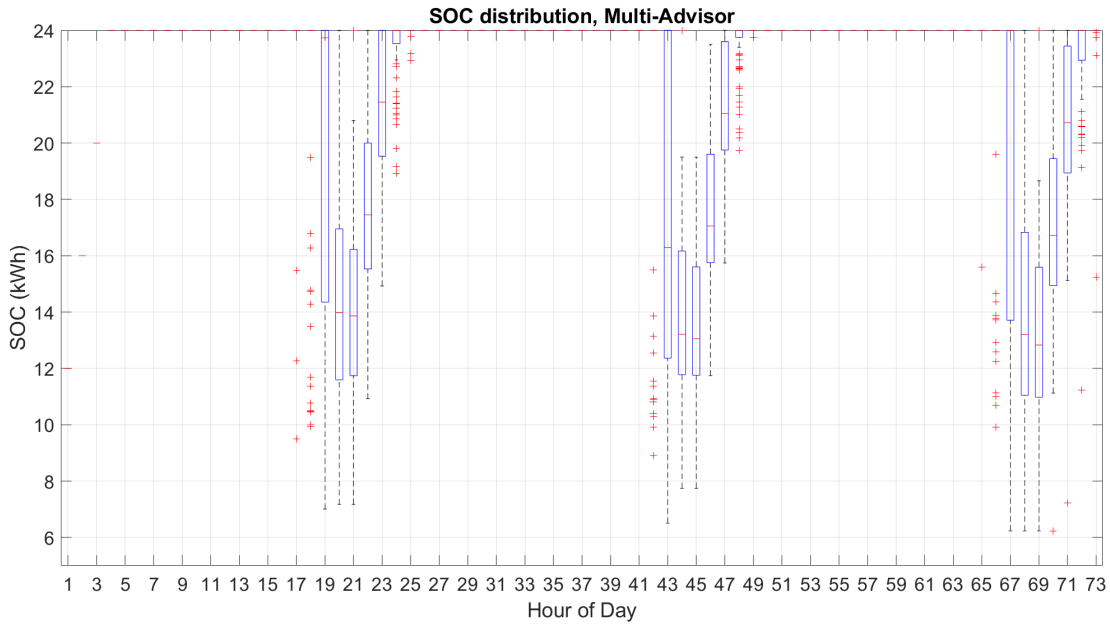


Figure 6.20: Hourly boxplots of indoor temperature with a multi-advisor agent

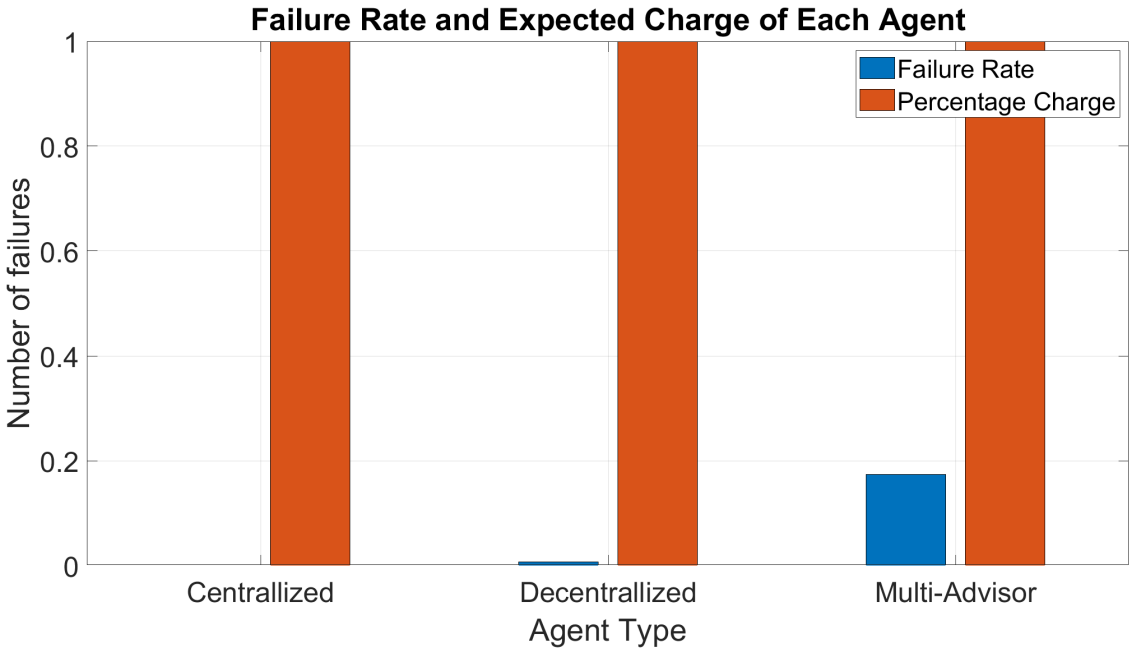


Figure 6.21: Expected SOC percentage and smart-contract failure rate of each agent type

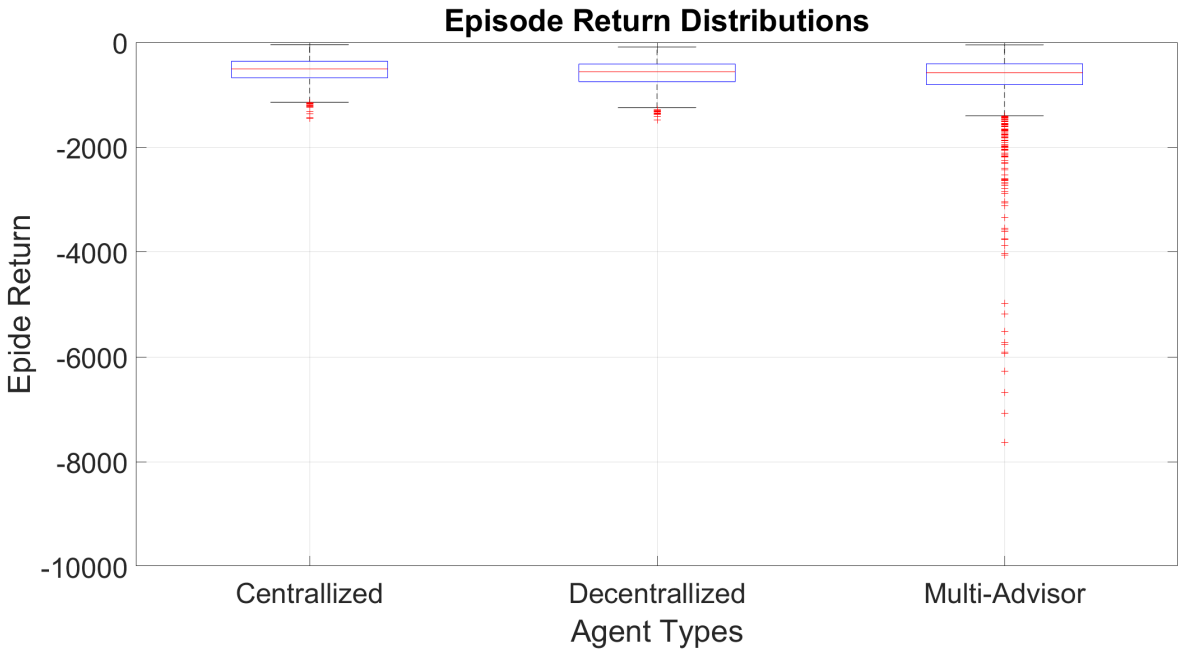


Figure 6.22: Boxplots of Episode Return for each agent type under random importance.

the 7 episodes with the lowest weighted return, plotted in figure 6.24. In each of these cases, temperature control is of notably little importance, and as such the multi-advisor agent rarely selects actions which increase the temperature. These low temperatures dramatically decrease the performance measure as displayed in figure 6.22. This indicates the need for stronger performance metrics, as the behaviour demonstrated is not necessarily undesirable, but doesn't align with the cumulative weighted reward metric used here.

6.6 Conclusions and Future Work

These experiments demonstrate the potential performance of the proposed multi-advisor agent for smart-home control. The proposed agent architecture has been shown to scale linearly with respect to both number of agents and number of objectives. Other experiments indicate that the performance of the multi-advisor agent can be similar to that of the centralized and decentralized DQN-based agents. The proposed agent has shown to be relatively successful at achieving every objective it is presented. While changing the weights seems to decrease the performance of the multi-advisor agent more dramatically relative to the benchmarks agents, the instantaneous dramatic change in behaviour indicates the potential of the proposed agent for quickly adapting to changes in user preferences.

The results are diminished by the decrease in performance of the proposed agent when the impor-

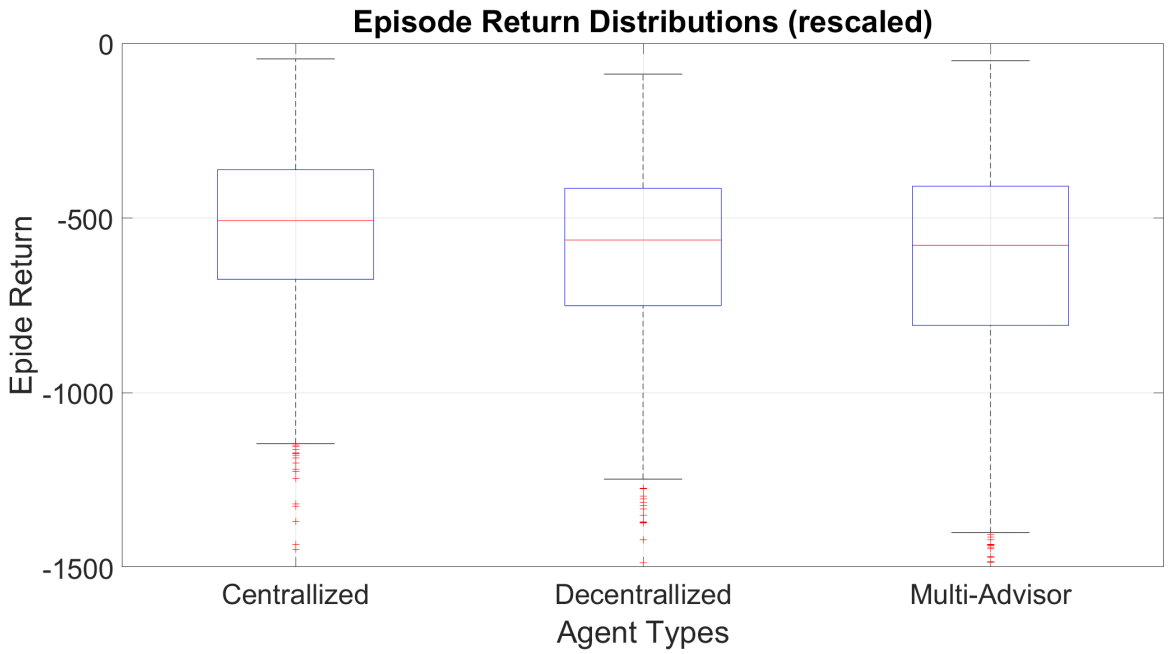


Figure 6.23: Boxplots of Episode return for each agent type under random importance, Re-scaled for clarity.

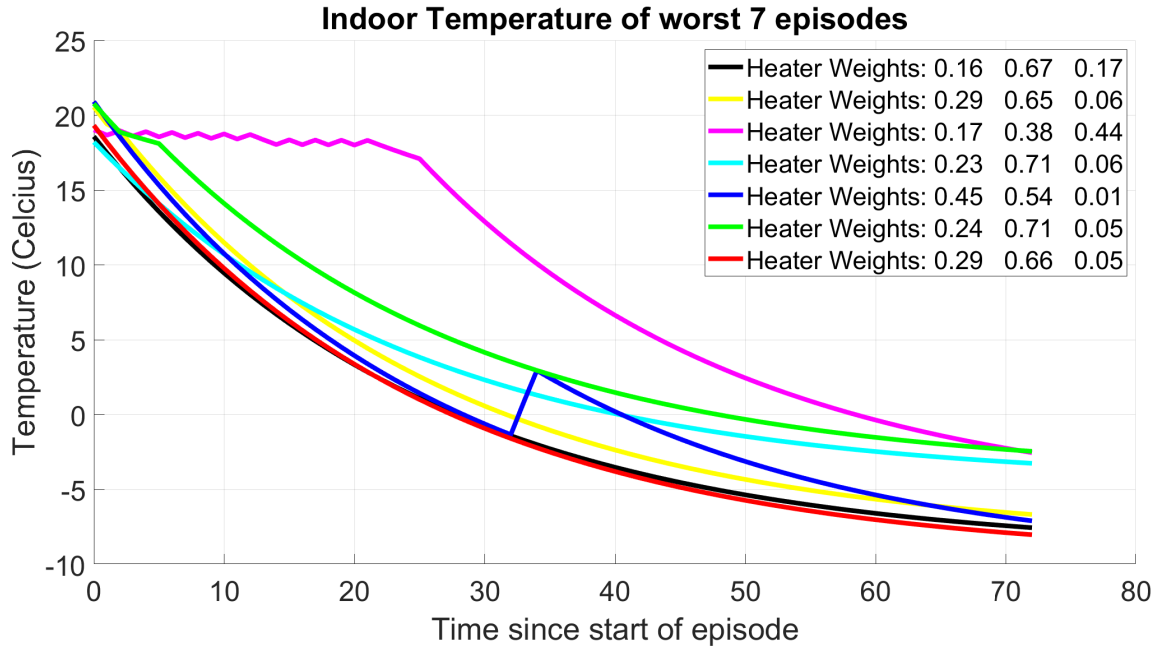


Figure 6.24: Indoor temperature of worst performing multi-advisor episodes under random importance.

tance of each objective is varied. While the selected performance metric decreases, the associated change in behaviour is not necessarily undesirable. This indicates the need for a more-specialized performance metric for such a system to reasonably compare the results with standard, single-objective agents.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

While many reinforcement learning algorithms can be described as 'model-free', they generally still require far too much training time and failed experiments to be directly applicable to the real world without a model. Nonetheless, a reinforcement learning algorithm need not be aware whether it's actions are taken in a simulator or the real world, as in the Dyna architecture [46], and therefore can be applied with relative ease when an environment model is available. Therefore, reinforcement learning algorithms can reach feasibility in real applications such as smart-homes either given a reliable model, or with advances in data-efficiency of reinforcement learning schemes. MAd-RL provides a scheme which enables the use of any standard off-policy action-value based reinforcement learning algorithm as a building block.

The potential of MAd-RL is shown by the experiments provided in chapters 5 and 6. In particular, chapter 6 demonstrated that MAd-RL is capable of making satisfying trade-offs between conflicting objectives, as well as demonstrating the desirable behaviour by switching dramatically during periods of lower preference. The work in chapter 6 further demonstrates the potential by expanding to a more complex multi-agent environment. This chapter demonstrated the performance in comparison to some reasonable benchmark agents, of which performance was comparable in terms of weighted cumulative reward. Further analysis revealed the difference in strategy undertaken by the multi-advisor system, revealing greater risk-taking behaviour by the temperature control agent in regards to smart contracts.

Upon consideration of the effect of changing the weight function arbitrarily, an issue in performance metrics for a system of this type is revealed; the behaviour of the multi-advisor agent changes dramat-

ically with the change in weights in a way that may prioritize achieving singular goals as opposed to maximizing weighted reward. Nonetheless, the new behaviour isn't strictly undesirable, and we note that future work should involve identifying more representative performance measures when comparing algorithms of this type. It seems that a state-weighted reward functions provide different outcomes than state-weighted action-value functions, notably due to the fact that action-value functions represent future rewards, which may have a much lower weight when they are actually received rather than the higher weight that may have led to this reward.

7.2 Future Work

Here we present some promising ideas to extend and improve this work, as well as some curiousities as to potential algorithm improvements which may be essential to implementing a scheme of this type in the real world.

Learning Aggregator

The aggregation function utilized in this work uses hand-selected hardcoded weights to aggregate the action-values into a decision. These weights are considered as representative of human preferences, but the mapping from weights to performance may be too complex to ask of a user. A feedback system could be developed to learn the aggregation function, which may be able to more accurately represent human preferences than the simple weight-based aggregation used in this work. This could be achieved by surveying a user about their experiences, and using the survey results to specify a reward signal/performance measure to train the aggregator.

Partial Observability

In this work, each advisor shares the same state-space. This may be of concern w.r.t. privacy, as some devices need not have knowledge of the other's input features[1]. Secondly, this enables further computational scalability as the input dimensionality of the neural networks would decrease, allowing a smaller-sized network be used for each problem. Experimentation could be performed by defining a minimum set of state-variables for each advisor/agent, and gradually including states from other agents, measuring performance as a function of observability.

[1] Eg: An automatic coffee maker doesn't need to know the user's desired indoor temperature.

Growing Agents

Related to the desire for functionality under partial observability is the task of expanding the state-space. Particularly, an answer to the question "How can I efficiently improve performance by adding a new sensor to my already functioning system?" is desired. This is notably a concern when considering that work involving the expansion of the input-space of a neural network is elusive.

Furthermore, adding an advisor (a new objective) to a previously trained agent is desirable as well because unforeseen objectives commonly manifest. Intuitively, the new advisor would require the inclusion of more sensors, which relates to the previously described partial observability concerns. This would include work involving training a newly added advisor with a new objective in a previously trained and functioning system.

Alternate Advisor Algorithms

In this work, each advisor is implemented by a DQN, a feedforward neural network which estimates $Q(s, a)$. Other algorithms such as Q-learning [58] or neural episodic control [35] could be used instead. The former is simple and requires a discrete state-space. The latter is complex, requires a detailed architecture, but learns at a much faster rate. There are many Q-function based algorithms available for experiment. Furthermore, another potentially interesting idea could be to instead approximate that objective's (stochastic) policy directly using a policy-gradient method, and have the aggregator re-weight the probabilities according to user preferences for the current state.

Advanced Multi-agent Algorithms

In chapter 6, the multi-agent component of the system is dealt with by treating each other agent as part of the environment. This technique is known as independent Q-learning, as each agent learns independently from the others. There are more dedicated algorithms for multi-agent learning available, but are outside the scope of this work. For example, in the related work, we note algorithms presented in [11] and [23] which are dedicated to dealing with the issues of multi-agent reinforcement learning. These techniques both follow the centralized learning, decentralized execution paradigm, which could be challenging to integrate when agents are allowed to have multiple objectives.

Feature Sharing

A standard approach to transfer learning in image recognition using convolutional neural networks is to treat the output from the last convolutional layer as a set of features to provide to any standard supervised learning algorithm. In this way, the features useful for identifying cats could be helpful to identify dogs. Similarly, the features that are useful to estimate $Q(s, a)$ for a single objective may be more useful to a learning aggregator than the Q-values themselves. This could essentially break the learning problem down into components which are more simple to aggregate. Furthermore, early layer features could be shared between the advisors internally, for example, a recurrent network could be used to extract features from raw sensor data, and fed into fully-connected predictors for each of the advisors.

Bibliography

[1] M. H. Albadi and E. F. El-Saadany. "Demand Response in Electricity Markets: An Overview". In: *2007 IEEE Power Engineering Society General Meeting*. June 2007, pp. 1–5. DOI: 10.1109/PES.2007.385728.

[2] S. Althaher, P. Mancarella, and J. Mutale. "Automated Demand Response From Home Energy Management System Under Dynamic Pricing and Power and Comfort Constraints". In: *IEEE Transactions on Smart Grid* 6.4 (July 2015), pp. 1874–1883. ISSN: 1949-3053. DOI: 10.1109/TSG.2014.2388357.

[3] Enda Barrett and Stephen Linder. "Autonomous hvac control, a reinforcement learning approach". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2015, pp. 3–19.

[4] Richard Bellman. "A Markovian Decision Process". In: *Indiana Univ. Math. J.* 6 (4 1957), pp. 679–684. ISSN: 0022-2518.

[5] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[6] Yu-Han Chang, Tracey Ho, and Leslie P Kaelbling. "All learning is local: Multi-agent learning in global reward games". In: (2004).

[7] *Choosing the Algorithm - MATLAB Simulink*. Available online: https://www.mathworks.com/help/optim/ug/choosing-the-algorithm.html. (Accessed: 2019-07-27).

[8] Panos Constantopoulos, Fred C Schweppe, and Richard C Larson. "ESTIA: A real-time consumer control scheme for space conditioning usage under spot electricity pricing". In: *Computers & operations research* 18.8 (1991), pp. 751–765.

[9] George Bernard Dantzig. *Linear Programming and Extensions*. Santa Monica, CA: RAND Corporation, 1963. DOI: 10.7249/R366.

[10] Damien Ernst, Pierre Geurts, and Louis Wehenkel. "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research* 6 (2005), pp. 503–556.

[11] Jakob Foerster et al. "Counterfactual multi-agent policy gradients". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[12] Vincent François-Lavet et al. "An introduction to deep reinforcement learning". In: *Foundations and Trends® in Machine Learning* 11.3-4 (2018), pp. 219–354.

[13] Arthur M Geoffrion. "Generalized benders decomposition". In: *Journal of optimization theory and applications* 10.4 (1972), pp. 237–260.

[14] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: a new learning scheme of feedforward neural networks". In: *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*. Vol. 2. 2004, 985–990 vol.2. DOI: 10.1109/IJCNN.2004.1380068.

[15] S. Gupta et al. "Multi-objective Reinforcement Learning based approach for User-Centric Power Optimization in Smart Home Environments". In: *2020 IEEE International Conference on Smart Data Services (SMDS)*. 2020, pp. 89–96. DOI: 10.1109/SMDS49396.2020.00018.

[16] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2021. URL: http://www.gurobi.com.

[17] Matthew Hausknecht and Peter Stone. "Deep recurrent q-learning for partially observable mdps". In: *arXiv preprint arXiv:1507.06527* (2015).

[18] *Historical electricity rates — Ontario Energy Board*. Available online: https://www.oeb.ca/rates-and-your-bill/electricity-rates/historical-electricity-rates. (Accessed: 2019-07-27).

[19] A. Jindal, M. Singh, and N. Kumar. "Consumption-Aware Data Analytical Demand Response Scheme for Peak Load Reduction in Smart Grid". In: *IEEE Transactions on Industrial Electronics* 65.11 (Nov. 2018), pp. 8993–9004. ISSN: 0278-0046. DOI: 10.1109/TIE.2018.2813990.

[20] Romain Laroche et al. "Multi-Advisor Reinforcement Learning". In: *CoRR* abs/1704.00756 (2017). arXiv: 1704.00756. URL: http://arxiv.org/abs/1704.00756.

[21] Y. Li et al. "Automated Residential Demand Response: Algorithmic Implications of Pricing Models". In: *IEEE Transactions on Smart Grid* 3.4 (Dec. 2012), pp. 1712–1721. ISSN: 1949-3053. DOI: 10.1109/TSG.2012.2218262.

[22] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[23] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *arXiv preprint arXiv:1706.02275* (2017).

[24] Renzhi Lu, Seung Ho Hong, and Mengmeng Yu. "Demand response for home energy management using reinforcement learning and artificial neural network". In: *IEEE Transactions on Smart Grid* 10.6 (2019), pp. 6629–6639.

[25] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[26] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[27] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[28] Elena Mocanu et al. "On-line building energy optimization using deep reinforcement learning". In: *IEEE transactions on smart grid* 10.4 (2018), pp. 3698–3708.

[29] Hossam Mossalam et al. "Multi-objective deep reinforcement learning". In: *arXiv preprint arXiv:1610.02707* (2016).

[30] John F Nash et al. "Equilibrium points in n-person games". In: *Proceedings of the national academy of sciences* 36.1 (1950), pp. 48–49.

[31] *Nest Temperature Sensor - Google Store*. Available online: https://store.google.com/ca/product/nest_temperature_sensor. (Accessed: 2021-04-02).

[32] D. O'Neill et al. "Residential Demand Response Using Reinforcement Learning". In: *2010 First IEEE International Conference on Smart Grid Communications*. Oct. 2010, pp. 409–414. DOI: 10.1109/SMARTGRID.2010.5622078.

[33] K. Ogata. *Modern Control Engineering*. Instrumentation and controls series. Prentice Hall, 2010. ISBN: 9780136156734. URL: https://books.google.ca/books?id=Wu5GpNAelzkC.

[34] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. "Optimal and approximate Q-value functions for decentralized POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 289–353.

[35] Alexander Pritzel et al. "Neural episodic control". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2827–2836.

[36] *Rates - Synergy North*. Available online: `https://synergynorth.ca/residential/billing/rates/`. (Accessed: 2019-07-27).

[37] *Reinforcement Learning Toolbox Documentation*. Available online: `https://www.mathworks.com/help/reinforcement-learning/`. (Accessed: 2021-02-22).

[38] Martin Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. Springer. 2005, pp. 317–328.

[39] H. Roh and J. Lee. "Residential Demand Response Scheduling With Multiclass Appliances in the Smart Grid". In: *IEEE Transactions on Smart Grid* 7.1 (Jan. 2016), pp. 94–104. ISSN: 1949-3053. DOI: 10.1109/TSG.2015.2445491.

[40] F. Ruelens et al. "Residential Demand Response of Thermostatically Controlled Loads Using Batch Reinforcement Learning". In: *IEEE Transactions on Smart Grid* 8.5 (2017), pp. 2149–2159. DOI: 10.1109/TSG.2016.2517211.

[41] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[42] Stuart J Russell and Andrew Zimdars. "Q-decomposition for reinforcement learning agents". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 656–663.

[43] T. Samad, E. Koch, and P. Stluka. "Automated Demand Response for Smart Buildings and Microgrids: The State of the Practice and Research Challenges". In: *Proceedings of the IEEE* 104.4 (2016), pp. 726–744. DOI: 10.1109/JPROC.2016.2520639.

[44] Lloyd S Shapley. "Stochastic games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[45] S. Singh and A. Yassine. "Mining Energy Consumption Behavior Patterns for Households in Smart Grid". In: *IEEE Transactions on Emerging Topics in Computing* 7.3 (2019), pp. 404–419.

[46] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[47] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[48] Richard S Sutton, Andrew G Barto, and Ronald J Williams. "Reinforcement learning is direct adaptive optimal control". In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 19–22.

[49] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation." In: *NIPs*. Vol. 99. Citeseer. 1999, pp. 1057–1063.

[50] Ming Tan. "Multi-agent reinforcement learning: Independent vs. cooperative agents". In: *Proceedings of the tenth international conference on machine learning*. 1993, pp. 330–337.

[51] A. Tittaferrante and A. Yassine. "Importance Scaling for Elastic Appliance for Automated Power Management in Smart Homes". In: *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT and AI (HONET-ICT)*. Oct. 2019, pp. 115–120. DOI: 10.1109/HONET.2019.8907970.

[52] A. Tittaferrante and A. Yassine. "Importance Scaling for Elastic Appliance for Automated Power Management in Smart Homes". In: *2019 IEEE 16th International Conference on Smart Cities: Improving Quality of Life Using ICT IoT and AI (HONET-ICT)*. 2019, pp. 115–120.

[53] *Topics in Reinforcement Learning by Dimitri Bertsekas*. Available online: https://mslgoee.asu.edu/Mediasite/Play/2d63c9507ef1456397ada6716e8523591d. (Accessed: 2021-04-03).

[54] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[55] Harm Van Seijen et al. "Hybrid reward architecture for reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5392–5402.

[56] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. International Series in Operations Research & Management Science. Springer US, 2007. ISBN: 9780387743882. URL: https://books.google.ca/books?id=T-BW1g69wbYC.

[57] Z. Wan et al. “Model-Free Real-Time EV Charging Scheduling Based on Deep Reinforcement Learning”. In: *IEEE Transactions on Smart Grid* 10.5 (2019), pp. 5246–5257.

[58] Christopher J.C.H. Watkins and Peter Dayan. “Technical Note: Q-Learning”. In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1023/A:1022676722315. URL: https://doi.org/10.1023/A:1022676722315.

[59] Z. Wen, D. O’Neill, and H. Maei. “Optimal Demand Response Using Device-Based Reinforcement Learning”. In: *IEEE Transactions on Smart Grid* 6.5 (2015), pp. 2312–2324.

[60] Marco Wiering and Martijn Van Otterlo. “Reinforcement learning”. In: *Adaptation, learning, and optimization* 12.3 (2012).

[61] David H Wolpert and Kagan Tumer. “Optimal payoff functions for members of collectives”. In: *Modeling complexity in economic and social systems*. World Scientific, 2002, pp. 355–369.

[62] X. Xu et al. “A Multi-Agent Reinforcement Learning-Based Data-Driven Method for Home Energy Management”. In: *IEEE Transactions on Smart Grid* 11.4 (2020), pp. 3201–3211. DOI: 10.1109/TSG.2020.2971427.

[63] A. Yassine. “Implementation challenges of automatic demand response for households in smart grids”. In: *2016 3rd International Conference on Renewable Energies for Developing Countries (REDEC)*. July 2016, pp. 1–6. DOI: 10.1109/REDEC.2016.7577546.

[64] Erik Zawadzki, Asher Lipson, and Kevin Leyton-Brown. “Empirically evaluating multiagent learning algorithms”. In: *arXiv preprint arXiv:1401.8074* (2014).

[65] D. Zhang et al. “An Optimal and Learning-Based Demand Response and Home Energy Management System”. In: *IEEE Transactions on Smart Grid* 7.4 (July 2016), pp. 1790–1801. ISSN: 1949-3053. DOI: 10.1109/TSG.2016.2552169.