TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRONIC AND COMPUTER ENGINEERING

Factored MDPs for Optimal Prosumer Decision-Making in the Smart Grid

by

Angelos Angelidakis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE M.Sc. DEGREE IN

ELECTRONIC AND COMPUTER ENGINEERING

November 2015

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Παραγοντοποιημένες Μαρκοβιανές Στοχαστικές Διαδικασίες για Βέλτιστη Λήψη Αποφάσεων ενός Παραγωγού-Καταναλωτή στο Έξυπνο Δίκτυο Ηλεκτροδότησης

Άγγελος Αγγελιδάκης

ΔΙΑΤΡΙΒΗ ΓΙΑ ΤΗ ΜΕΡΙΚΗ ΕΚΠΛΗΡΩΣΗ
ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ ΓΙΑ ΤΟ ΜΔΕ
ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΝΟΕΜΒΡΙΟΣ 2015

Abstract

Tackling the decision-making problem faced by a *prosumer* (i.e., a producer that is simultaneously a consumer) when selling and buying energy in the emerging smart electricity grid, is of utmost importance for the economic profitability of such a business entity. In this thesis, we model, for the first time, this problem as a *factored* Markov Decision process (MDP). Our model successfully captures the main aspects of the business decisions of a prosumer corresponding to a community microgrid of *any* size. Moreover, it includes appropriate sub-models for prosumer production and consumption prediction.

Employing this model, we are able to represent the problem compactly, and to provide an *exact optimal solution via dynamic programming*—notwithstanding its large size. In addition, we show how to use *approximate MDP solution methods* for taking decisions in this domain, without the need of discretizing the state space. Specifically, we employ fitted value iteration, a sampling-based approximation method that is known to be well behaved. By so doing, we generalize our factored MDP solution method to *continuous* state spaces.

Our experimental simulations verify the effectiveness of our approach. They show that our exact value iteration solution matches that of a state-of-the-art method for stochastic planning in very large environments, while outperforming it in terms of computation time. Furthermore, we evaluate our approximate solution method via using a variety of basis functions over different state sample sizes, and comparing its performance to that of our exact value iteration algorithm. Our approximation method is shown to exhibit stable performance in terms of accumulated reward, which for certain basis functions reaches 90% of that gathered by the exact algorithm.

Acknowledgements

I would like to take this opportunity to sincerely thank my thesis supervisor, Assistant Professor Dr. Georgios Chalkiadakis for his support and guidance towards the completion of my diploma. I really appreciate his valuable help and understanding, and I am deeply thankful due to the fact that he has always spared his time to discuss and solve problems that presented during my implementation. Special thanks is also extended to Associate Professor Dr. Michail G. Lagoudakis and Assistant Professor Dr. Eftichios Koutroulis for their suggestions. Furthermore I would like to express my gratitude to my parents for their continuous support, endless encouragement and confidence in me especially during my studies. Many thanks to my friends, who have been very helpful in giving me moral support throughout my studies and towards the acquisition of my diploma.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Decision-theoretic planning (DTP) [13] has attracted a considerable amount of attention recently as AI researchers seek to generalize the types of planning problems that can be tackled in computationally effective ways. DTP is primarily concerned with problems of sequential decision making under conditions of uncertainty and where there exist multiple, often conflicting, objectives whose desirability can be quantified. *Markov Decision Processes (MDPs)* [41] allow the introduction of uncertainty into the effects of actions, the modelling of uncertain exogenous events, the presence of multiple, prioritized objectives, and the solution of non–terminating process–oriented problems.

The foundations and the basic computational techniques for MDPs are well understood and in certain cases can be used directly in DTP. These methods exploit the dynamic programming principle and some of them allow MDPs to be solved in time polynomial in the size of the state and action spaces that make up the planning problem. Unfortunately, these classic dynamic programming methods are formulated so as to require explicit state space enumeration. Classic MDP solution methods are faced with the so–called "curse of dimensionality": the number of states grows exponentially with the number of variables that characterize the planning domain. However, methods have been developed that, in many instances, circumvent this problem. In classical planning one typically does not specify actions and goals explicitly using the underlying state space, but rather ”intensionally” using propositional or variable-based representations [12]. For instance, a STRIPS representation [43] of an action describes very concisely the transitions induced by that action over a large number of states. Classical planning techniques such as regression planning or nonlinear planning exploit these representations to great effect. Intuitively, such methods aggregate states that behave identically under a given action sequence with respect to a given goal [12].

Similarly, many large MDPs have significant internal structure, and can be modeled compactly if the structure is exploited in the representation. Factored MDPs [12] are one approach to representing large, structured MDPs compactly. In this framework, a state is implicitly described by an assignment to some set of state variables. A *dynamic Bayesian network* (DBN [55] can then allow a compact representation of the transition model, by exploiting the fact that the transition of a variable often depends only on a small number of other variables. DBN is a way to extend Bayesian Networks [25] to model probability distributions over a collections of random variables. Furthermore, the momentary rewards can often also be decomposed as a sum of rewards related to individual variables or small clusters of variables. Even when a large MDP can be represented compactly, for example, by using a factored representation, solving it exactly may still be intractable: Typical exact MDP solution algorithms require the manipulation of a value function, whose representation is linear in the number of states, which is exponential in the number of state variables. One approach is to approximate the solution using an approximate value function with a compact representation. A common choice is the use of linear value functions as an approximation — value functions that are a linear combination of potentially non-linear basis functions [40, 50, 32].

MDPs are a natural choice for tackling DTP problems in the Smart Grid, a fast growing field of research [6]. Electricity is the most versatile and widely used form of energy and global demand is growing continuously. Generation of electrical power, however, is an important source of carbon dioxide emissions, making a significant contribution to climate change [44]. Most of today's generation capacity relies on fossil fuels and contributes significantly to the increase of carbon dioxide in the world's atmosphere, with negative consequences for the climate and society in general [44]. To mitigate the consequences of climate change, the current electrical system needs to undergo significant adjustments. To satisfy both the increasing demand for power and the need to reduce carbon dioxide emissions, we need an electric system that can handle these challenges in a sustainable, reliable and economic way [38]. Smart grids will provide more electricity to meet rising demand, increase reliability and quality of power supplies, increase energy efficiency, be able to integrate low carbon energy sources into power networks. Smart grids also possess demand response capacity to help balance electrical consumption with supply, as well as the potential to integrate new technologies to enable energy storage devices and the large-scale use of electric vehicles [56]. Electrical systems will undergo a major evolution, improving reliability and reducing electrical losses, capital expenditures and maintenance costs. A smarter grid will provide greater control over energy costs and a more reliable energy supply for consumers using *renewable energy sources (RES)*. The environmental benefits of a smarter grid include reduced peak demand which means reduced need for conventional generators; integration of more renewable power sources; and reduced CO_2 emissions and other pollutants [33].

A major scientific and societal concern in the Smart Grid is estimating the power output of inherently intermittent and potentially distributed *renewable energy sources (RES)*. There are a few works dealing with the estimation of the power output such as [3]. Another important issue in the Smart Grid is enhancing RES effectiveness. For instance, the power output of photovoltaic systems (PVS) increases with the use of effective and efficient solar tracking techniques. A proposed solution method for this problem based on reinforcement learning (RL) [36] is described in [4]. Further, as technology evolves and electricity demand rises, the task to keep it precisely balanced with supply at all times becomes especially challenging. Maintaining *demand curve stability*, in particular, can alleviate the risk of disastrous electricity network collapses, and leads to financial and environmental benefits, as then some generators can be run on idle, or even be shut down completely. Quite a few recent works deal with this so-called *demand side management* problem(see, e.g. [8, 47].

In recent years, the term *prosumer* has been coined in order to describe an entity that both produces and consumes energy, implying that prosumers possess the ability to play a key role to the stabilization of the electricity network [38, 53]. As such, and assuming prosumers are able to adjust their behaviour according to dynamic indicators, their smooth integration into the shaping Smart Grid is of critical importance [6]. Viewed as a business entity, a prosumer could correspond to a single residence, a specific industry, or to whole neighborhoods of houses that are served by a dedicated microgrid—which may or may not be connected to the rest of the electricity Grid.

Our focus of attention in this will be optimizing the business decisions corresponding to a microgrid prosumer, which produces electricity from (mainly) renewable energy resources, and which has the option of buying and selling energy from utility companies residing in the larger electricity Grid. Paradigms of such community-oriented and renewable energy-relying microgrids are expected to be commonplace in the near future [38]. Naturally, the viability (economic and otherwise) of such an entity is tightly connected to the quality of its business decisions, whether to buy, sell, or store energy, within some decisions horizon, in order to possibly make a profit while ensuring the smooth operation of its energy-consuming units; and ensuring this viability is key to the smooth integration of prosumers into the Smart Grid.

More precisely, we created a microgrid prosumer agent, that plans electricity purchasing, storing, and selling decisions for the day-ahead, aiming to serve its electricity needs while perhaps making a profit via selling energy (via participation in some day-ahead market). In our scenario, we assume that the agent has the option to buy or sell energy from/to various utility companies operating over the Grid. The buy/sell electricity prices for the day-ahead are determined via "tariffs" issued by the utility companies at the beginning of each day, and to which the agent can subscribe to.

1.1 Contributions

To the best of our knowledge, no work to date has attacked this specific problem heads on. Thus, our main contribution lies in describing compactly and evaluating, for the first time, the decision problem faced by a microgrid prosumer planning its energy production, storage and usage strategy for the day ahead as a *factored Markov Decision Process* [13]. Our formulation enables us to provide *an exact optimal solution* (using certain discretization-related modelling decisions) for the problem faced by a prosumer corresponding to a microgrid of essentially *any* size.

The exact solution to the prosumer decision problem can be computed using standard dynamic programming techniques. In this thesis, we employed *value iteration* to this purpose. The effectiveness and efficiency of our approach is verified by comparisons to the performance of *SPUDD*, a state-of-the-art method for stochastic planning in large environments. Our value iteration method, operating over a problem horizon corresponding to twenty–four hours, is shown to produce policies that coincide with those produced by SPUDD [30]. However, as we explain in Chapter 5 below, SPUDD has to operate over a state space that is artificially larger, while, at the same time, it does not possess enough structure. This creates a need to build huge input files for SPUDD to operate on, resulting to a huge pre-processing time for the algorithm. As a result, while our method can scale to larger state spaces for our problem, SPUDD cannot produce a solution in such cases within the required twenty–four hour time–frame

In addition, we equip our consumer with specific consumption and production-predicting submodels, which provide it with the necessary input signals on which to base its decisions. As part of our work, we show that *Gaussian processes* and *Bayesian linear regression* techniques can be successfully used for consumption prediction. To obtain the production estimates of the photovoltaic systems (PVS) and wind turbine generators (WTG) of our microgrid, we employ *RENES* [3], a web-based PVS and WTG production prediction tool.

Finally, we also show how to use approximate MDP solution methods for taking decisions in this domain without the need of discretizing the state space. Specifically, we employ *fitted value iteration*, a sampling-based approximation method that is known to be well behaved. By so doing, we generalize our factored MDP solution method to continuous state spaces. We evaluate our approach using a variety of basis functions over different state sample sizes, and compare its performance to that of our original "exact" value iteration algorithm. Our generic approximation method is shown to exhibit stable performance in terms of accumulated reward, which for certain basis functions reaches 90% of that gathered by the exact algorithm. Our thesis results were presented in two scientific conferences publications [1, 2].

1.2 Thesis Layout

This thesis is organized into the following chapters: Chapter 2 provides a brief background on factored MDPs and reviews related work. Chapter 3 then describes our model, its factored representation, and the production and consumption models we employed. Chapter 4 describes our solution methods (exact and approximate). Chapter 5 presents our simulation experiments; and, finally, Chapter 6 concludes and outlines future work.

Chapter 2

Background and Related Work

In this chapter, we extensively describe *factored MDPs (FMDPs)*, which provide a compact way to represent *MDPs*. In addition, we outline several well-known (factored) MDP solution methods, some of which can operate over both discrete and continuous state spaces. Finally, we review several works on decision making in the Smart Grid.

2.1 MDPs

Markov decision processes (MDPs) [41], provide a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems solved via dynamic programming and reinforcement learning. They are used in a wide area of disciplines, including robotics, automated control, economics, and manufacturing. More precisely, a MDP is a discrete time stochastic control process. At each time step, the process is in some state s, and the decision maker may choose any action a that is available in state s. The process responds at the next time step by randomly moving into a new state s', and giving the decision maker a corresponding reward $R_a(s, s')$. The probability that the process moves into its new state s' is influenced by the chosen action. Specifically, it is given by the state transition function $P_a(s, s')$. Thus, the next state s' depends on the current state s and the decision maker's action a. But given s and a, it is conditionally independent of all previous states and actions; in other words, the state transitions of an MDP possess the Markov property.

Thus, a Markov decision process is a 4-tuple (S,A,T,R) composed of:

✓ a (finite) set S of states

- ✓ a (finite) set A of actions
- ✓ (Markov) transition function $T(s,a,s') = Pr(s'|s,a)$, specifying the probability $Pr(s'|s,a)$ of going to state s' after taking action a in state s
- ✓ reward function $R(s)$, determining the immediate reward received after transition to state s' from state s.

An MDP can be represented graphically,as shown in figure 2.1. S_t represent the current state and R_t represent the immediate reward received in that state. In addition, the edge represents the transition from state S_t to state S_{t+1}, after taking action A_t.

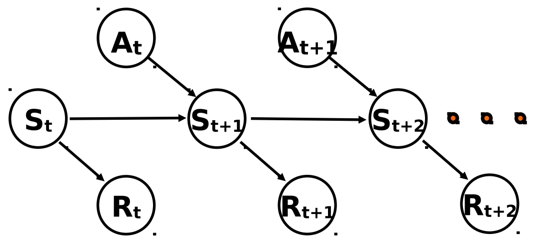


Figure 2.1: Graphically MDP representation.

The core problem of MDPs is to find a policy for the decision maker: a function π that specifies the action $\pi(s)$ that the decision maker will choose when in state s. Note that once a Markov decision process is combined with a policy in this way, this fixes the action for each state. The goal is to choose a policy π that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon: $\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}(s_t, s_{t+1})$ where γ is a discount factor that satisfies $0 \leq \gamma \leq 1$.

2.2 Factored MDPs

Factored Markov Decision Processes (FMDPs) [13] provide a *compact* alternative to standard MDP representation. Specifically, they decompose states into sets of *state variables* in order to represent the transition and model compactly—since transitions and rewards may rely on specific model aspects, corresponding to subsets of variables only. Thus, the set of states in a factored MDP representation correspond to multivariate random variables, $\boldsymbol{s} = \langle s_i \rangle$, with the s_i variables taking on values in their corresponding $DOM(s_i)$ domains. Intuitively, state variables correspond to a selection of *features* which are sufficient to describe the system state. In FMDPs, actions are also quite often described as random variables, while reward functions used are assumed to be factored into specific (usually additive) components. Furthermore, FMDP models allow for *external signals*, described by *signal variables*, affecting state variables; while *temporal Bayesian networks (TBNs)* and *influence diagrams* can be employed to represent the effects of actions on state transitions and rewards. There are two main types of structure that can simultaneously be exploited in factored MDPs: additive and context-specific structure [15]. *Additive* structure captures the fact that typical large-scale systems can often be decomposed into a combination of locally interacting components. *Context-specific* structure encodes a different type of locality of influence: Although a part of a large system may, in general, be influenced by the state of every other part of this system, at any given point in time only a small number of parts may influence it directly. A multitude of techniques that exploit the resulting representational structure can then be used to solve large problems, at least approximately (e.g., linear value functions, approximate linear programming, stochastic algebraic decision diagrams, and so on) [15, 13].

2.3 Factored MDPs Solution Methods

Many solution methods can be used to solve large problems modelled as Factored MDPs. We provide an overview of the common approaches to solving MDPs [49, 40]. In sections 2.3.1 to 2.3.3 we outline approaches that can operate only over discrete state spaces, while sections 2.3.4 to 2.3.8 present methods that can operate over continuous spaces also, via operating over approximate representations of the underlying value function.

2.3.1 Value Iteration

Value Iteration is a method of computing an optimal MDP policy and its value. Value Iteration starts at the “end” and then works backward, refining an estimate of either Q-function or V-function [49].

The expected value of a policy π is defined in terms of two interrelated functions, V^π and Q^π. Let $Q^\pi(s,a)$, where s is a state and a is an action, be the expected value of doing a in state s and then following policy π. Recall that $V^\pi(s)$, where s is a state, is the expected value of following policy π in state s. Q^π and V^π can be defined recursively in terms of each other. If the agent is in state s, performs action a, and arrives in state s', it gets the immediate reward of $R(s,a,s')$ plus the discounted future reward, $\gamma V^\pi(s')$. When the agent is planning it does not know the actual resulting state, so it uses the expected value, averaged over the possible resulting states: $Q^\pi(s,a) = \sum_s' P(s'|s,a)(R(s,a,s') + \gamma V^\pi s')$. $V^\pi(s)$ is obtained by doing the action specified by π and then acting following π: $V^\pi(s) = Q^\pi(s,\pi(s))$.

There is really no end in Value Iteration, so it uses an arbitrary end point. Let V_k be the value function assuming there are k stages to go, and let Q_k be the Q-function assuming there are k stages to go. These can be defined recursively. Value iteration starts with an arbitrary function V_0 and uses the following equations to get the functions for $k+1$ stages to go from the functions for k stages to go:

$$Q_{k+1}(s,a) = \sum_s^{'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))\ for\ k\ \geq\ 0 \tag{2.1}$$

where, S is the (finite) state set , A is the (finite) action set $P(s'|s,a)$ is the transition function, $R(s)$ the reward function and γ is a discount–factor

$$V_k(s) = max_a Q_k(s,a)\ for\ k\ \geq\ 0. \tag{2.2}$$

2.3.2 Policy Iteration

Policy Iteration starts with a policy and iteratively improves it [49]. It starts with an arbitrary policy π_0 (an approximation to the optimal policy works best) and carries out the following steps starting from $i = 0$. Policy evaluation: determine $V_i^\pi(S)$. The definition of V^π is a set of $|S|$ linear equations in $|S|$ unknowns. The unknowns are the values of $V_i^\pi(S)$. There is an equation for each state. These equations can be solved by a linear equation solution method (such as Gaussian elimination) or they can be solved iteratively. Policy improvement: choose $\pi_{i+1} = \arg\max_a Q_i^{\pi}(s,a)$, where the Q-value can be obtained from V using:

$$Q^{\pi}(s,a) = \sum_{s}^{'} P(s'|s,a)(R(s,a,s') + \gamma V^{\pi}(s')). \tag{2.3}$$

To detect when the algorithm has converged, it should only change the policy if the new action for some state improves the expected value; that is, it should set $\pi_{i+1}(s)$ to be $\pi_i(s)$ if $\pi_i(s)$ is one of the actions that maximizes $Q_i^{\pi}(s,a)$. It stops if there is no change in the policy – that is, if $\pi_{i+1} = \pi_i$ – otherwise it increases i and repeats.

2.3.3 Stochastic Planning Using Decision Diagrams

Stochastic Planning Using Decision Diagrams (SPUDD) [30], is a well-known algorithm for finding (near-)optimal policies in very large problems represented as factored MDPs, for this reason we will compare our approach to it. It is essentially a value iteration algorithm that uses *algebraic decision diagrams (ADDs) [39]* to represent value functions and policies, assuming an ADD input representation of the FMDP provided in an input script *(a)* describing the factored states and actions, and *(b)* the transition model and reward function.

In order to define ADDs, we first have to present *Binary Decision Diagrams (BDDs)*. A BBD [51] is a directed acyclic graph that consists of nodes and edges. It deals with Boolean functions. A binary decision diagram consists of a set of decision nodes, starting at the root node at the top of the decision diagram. Each decision node contains two outgoing branches, one is a high branch and the other is a low branch. These branches may be represented as solid and dotted lines, respectively. The binary decision diagram contains high and low branches that are used to connect decision nodes with each other to create decision paths. The high and low branches of the final decision nodes are connected to either a high or low terminal node, which represents the output of the function. Figure 2.2 shown an instance of a graphical representation of BDDs on the left side. *(ADDs)* then extend BDDs to represent real-valued functions. We show an instance of a graphical representation of ADDs on the right side of figure 2.2. In an ADD, there are multiple terminal nodes labelled with numeric values. ADDs can be used, along with Dynamic Bayesian Networks (DBNs), in order to provide a compact MDP representation. Specifically, they can be used in order to represent the reward function, and the so-called Conditional Probability Tables (CPTs) that describe the MDP state transition function.

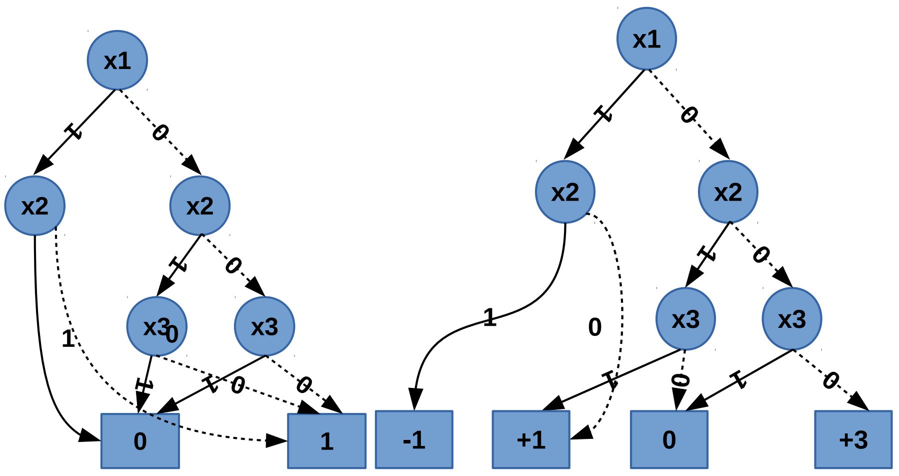


Figure 2.2: Left:BDD graphical representation. Right:ADD graphical representation.

In some detail, DBNs are probabilistic graphical models that relate variables to each other, over adjacent time steps. They can easily describe the effects that the execution of some action has on specific variables under certain conditions. A DBN for action a requires two sets of variables, one set $S = S_1, \ldots, S_n$ referring to the state of the system before action a has been executed, and $S' = S'_1, \ldots, S'_n$ denoting the state after a has been executed. Directed arcs from variables in X to variables in S' indicate direct causal influence and have the usual semantics. The conditional probability table (CPT) for each post-action variable S'_i defines a conditional distribution $P^a_{S_i}$ over S'_i. This can be viewed as a function $P^a_{S_i}\ (S_1 \ldots S_n)$, represented by an ADD, though the function value (distribution) depends only on those S_j that are parents of S_i. This representational technique allows one to describe a value function (or policy) as a function of the variables describing the domain rather than in the classic ”tabular” way. The decision graph used to represent this function is often extremely compact, implicitly grouping together states that agree on value at different points in the dynamic programming computation. As such, the number of expected value computations and maximizations required by dynamic programming is greatly reduced.

Against, this background the SPUDD algorithm itself is derived from the structured policy iteration (SPI) algorithm of [10, 11, 12], where decision trees are used to represent value functions and policies. Given a DBN action representation (with decision trees used to represent conditional probability tables), and a decision tree representation of the reward function, SPI constructs value functions that preserve much of the DBN structure. SPUDD offers considerable computational advantages in certain natural classes of problems. In addition, highly optimized ADD manipulation software can be used in the implementation of value iteration.

2.3.4 Linear Value Function Approximation

Value Function using linear regression is a popular approximate solution method [15, 40]. The space of allowable value functions is described via a set of basis functions $H = h_1, \dots, h_k$. Some coefficients **w** = (w_1, . . . , w_k) are used to define the value function as V= $\sum_{j=1}^{k} w_j h_j$ = A**w**. As a result of the definition of the value function, the projection weight is defined as:

$$\rho(s) = (V(s) - \hat{V}(s))^2 \tag{2.4}$$

where the notion of distance is weighted as $L2$ norm. The projection operation consists of computing

$$\mathbf{w} = (A^T \Lambda A)^{-1} A^T \Lambda V \tag{2.5}$$

where Λ is a weight matrix with diagonal entries equal to our projection weight ρ The key insight is that an iterative process is not required, we can find the fixed point directly by writing an approximate version. The iterative value determination equation is

$$V^{t+1} = \gamma P_\pi V^t + R \tag{2.6}$$

$$Aw^{t+1} = \gamma P_\pi A w^t + R \tag{2.7}$$

A weighted least-squares approximation to Eq. 2.7 is:

$$w^{t+1} = (A^T \Lambda A)^{-1}[\gamma A^T \Lambda P_\pi A w^{(t)} + A^T \Lambda R] \tag{2.8}$$

2.3.5 Approximate Policy Iteration

This method uses an approximate value function represented as a linear combination of basis functions [15]. It can be performed efficiently in closed form, by exploiting structure in a factored MDP. Approximate solution methods generate linear value functions which can be denoted by $H\hat{\mathbf{w}}$. In practice, the agent will define its behaviour by acting according to the greedy policy π = Greedy($H\hat{\mathbf{w}}$). Vπ of policy π compares to V^* using the Bellman error analysis [45].

The Bellman error is defined as

$$BellmanErr(V) = ||T * V - V||_\infty \tag{2.9}$$

Given the greedy policy π = Greedy(V), Bellman error analysis of Williams and Baird (1993) [45] provides the bound:

$$||V^* - V_{\hat{\pi}}||_\infty \leq \frac{2\gamma BellmanErr(V)}{1-\gamma} \tag{2.10}$$

It can efficiently compute bounds on policy quality based on the Bellman error. The exact policy iteration algorithm iterates over policies, producing an improved policy at each iteration. Starting with some initial policy π(0) , each iteration consists of two phases. Value determination computes the value function V_π(t), by finding the unique solution to the set of linear equations:

$$V_{\pi^{(t)}}(\mathbf{s}) = R(\mathbf{s}, \pi_{(t)}(\mathbf{s})) + \gamma \sum_{\mathbf{s'}} P(\mathbf{s'}|\mathbf{s}, \pi^{(t)}(s)) V_{\pi^{(t)}}(\mathbf{s'}), \forall \mathbf{s} \tag{2.11}$$

, where s is the state and s' the state after the transition. The policy improvement step defines the next policy as

$$\pi^{t+1} = Greedy(V_{\pi^{(t)}}) \tag{2.12}$$

It can be shown that this process converges to the optimal policy[21]. Denoting the reward and transition function, as well as the set of the basis functions

- $R_{\pi^{(t)}}$, $where\ R_{\pi^{(t)}}(s) = R(s, \pi^{(t)}(s))$
- $P_{\pi^{(t)}}(\mathbf{s'}|\mathbf{s}) = P(\mathbf{s'}|\mathbf{s}, \pi^{(t)\text{`}}(\mathbf{x}))$
- a set of basis functions $H = h_1, \ldots, h_k$

we can now rewrite the value determination step in terms of matrices and vectors. If we view $V_{\pi^{(t)}}$ and $R_{\pi^{(t)}}$ as N-vectors, and $P_{\pi^{(t)}}$ as an $N \times N$ matrix, we have the equations:

$$V_{\pi^{(t)}} = R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} V_{\pi^{(t)}} \tag{2.13}$$

This is a system of linear equations with one equation for each state, which can only be solved exactly for relatively small N . The goal is to provide an approximate solution, within H. More precisely, we want to find:

$$\begin{aligned} w^{(t)} = arg \min_{w} ||H\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} H\mathbf{w})|| = \\ arg \min_{w} ||(H - \gamma P_{\pi^{(t)}} H) w^{(t)} - R_{\pi^{(t)}}|| \end{aligned} \tag{2.14}$$

Thus, approximate policy iteration alternates between two steps:

$$w^{(t)} = arg \min_{w} H\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} H\mathbf{w}) \tag{2.15}$$

$$\pi^{(t+1)} = Greedy(H\mathbf{w}^{(t)}) \tag{2.16}$$

2.3.6 Factored Max-Norm Projection

The approaches in sections 2.3.4 and 2.3.5 suffer from ”norm incompatibility”. When computing the projection, they utilize the standard Euclidean projection operator with respect to the $L2$ norm or a weighted $L2$ norm. However, most of the convergence and error analyses for MDP algorithms utilize max-norm (L_∞). This incompatibility has made it difficult to provide error guarantees. The problem is defined as finding w^* such that:

$$w^* \in arg\min_w ||C\mathbf{w} - \mathbf{b}||_\infty \tag{2.17}$$

An algorithm [23] solves this problem by linear programming:

$$\begin{aligned} &Variables : w_1, \dots, w_k, \phi; \\ &Minimize : \phi; \\ &Subject\ to : \phi \geq \sum_{j=1}^{k} c_{ij} w_j - b_i\ and \\ &\qquad\qquad \phi \geq b_i - \sum_{j=1}^{k} c_{ij} w_j \end{aligned} \tag{2.18}$$

The $L\infty$ projection can be used in the context of the approximate policy iteration. When implementing the projection operation, we can use the $L\infty$ projection, where

$$C = (H - \gamma P_{\pi(t)} H)\ and\ b = R_{\pi(t)} \tag{2.19}$$

We can enhance this method to speed up computations when solving *factored MDPs* [14]. The key computational step is the solution of 2.17 using the linear program 2.18. the vectors Cw and b are vectors in $\mathbf{R}^{|S|}$, where S is our state space, a set of vectors which are assignments to the state variables $X = X_1, \dots, X_n$, where n is the total number of state variables. The size of the state space is exponential in the number of variables. The goal is to optimize 2.17 without explicitly considering each of the exponentially many states. To make the computation of 2.17 more efficient, we can view both Cw and b as functions of these state variables, and hence also their difference. Thus, we can define a function such that $F^w(x_i) = (Cw_i - b_i)$. The key for efficiency is to use the fact that F^w has a factored representation. More precisely, Cw has the form $\sum_j w_j f'_j(Z_j)$, where Z_j is a subset of X. We can express F^w as a sum $\sum_j f_j(Z_j)$, where f_j^w may or may not depend on w. We can maximize such a function F using a construction called a cost network[42], whose structure is very similar to a Bayesian network.

Hence, the goal becomes to compute [14]:

$$max_{x_1,\dots,x_n} \sum f_j(Z_j[x]).$$

where $Z_j[x]$ is the instantiation of the variables in Z_j in the assignment.

2.3.7 Approximate Linear Programming

Linear programming provides an alternative method for solving MDPs [22]. It formulates the problem of finding a value function as a linear program (LP). The LP variables are $V_1, \dots, V_N$, where V_i represents $V(s_i)$: the value of starting at the i^{th} state of the system. The LP is given by:

$$\begin{aligned} Variables :& V_1, \dots, V_N \\ Minimize :& \sum_{s_i} \alpha(s_i) V_i \\ Subject\ to :& V_i \geq [R(s_i, \alpha) + \gamma \sum_j P(s_j|s_i, \alpha) V_j] \ \forall \mathbf{s}_i \in \mathbf{S}, \alpha \in A \end{aligned}$$

where the state relevance weights α are positive The approximate formulation for the LP approach restricts the space of allowable value functions to the linear space spanned by the basis functions. In this approximate formulation, the variables are $w_1, \dots, w_k$: the weights for the basis functions. The LP is given by:

$$\begin{aligned} Varibales :& w_1, \dots, w_k \\ Minimize :& \sum_s \alpha(\mathbf{s}) \sum_i w_i h_i(\mathbf{s}) \\ Subject\ to :& \sum_i w_i h_i(s) \geq [R(\mathbf{s}, \alpha) + \gamma \sum_{\mathbf{s'}} P(\mathbf{s'}|\mathbf{s}, \alpha) \sum_i w_i h_i(\mathbf{s'})] \end{aligned}$$
$$\forall \mathbf{s} \in \mathbf{S}, \alpha \in A$$

2.3.8 Approximate Value Iteration

We now describe extensively *approximate value iteration*, which we use to solve the prosumer decision making problem. Algorithms for approximate value iteration fall into three different categories: model-based value iteration with parametric approximation, model-free value iteration with parametric approximation, and value iteration with non-parametric approximation. First, we describe the approximate value iteration with parametric approximation approaches in some detail. Specifically, we present below model-based algorithms, offline and online model-free algorithms and value iteration with non-parametric approximation.

Model-based value iteration with parametric approximation

This section considers Q-iteration with a parametric approximator. Q-iteration [49] is a model-based algorithm for approximate value iteration. Approximate Q-iteration [34] is an extension of the exact Q-iteration algorithm. Exact Q-iteration starts from a Q-function Q_0 and at each iteration i updates the Q-function:

$$Q_{i+1} = T(Q_{i+1}) \tag{2.20}$$

where T is the mapping between the states and the Q_{value}. In approximate Q-iteration, the Q-function cannot be represented exactly. Instead, an approximation is compactly represented by a parameter vector $\theta_i \ \in \ \rho^n$, using an appropriate approximation mapping F : $R^n \rightarrow Q$:

$$\hat{Q}_i = F(\theta_i) \tag{2.21}$$

This approximate Q-function replaces Q_i, as an input to the Q-iteration mapping T. So, the Q-iteration update becomes:

$$\hat{Q}_{i+1} = (T \circ F)(\theta_i) \tag{2.22}$$

The Q-function $\hat{Q}_{i+1}$ cannot be explicitly stored. Instead, it must also be represented approximately. A new parameter vector θ_{i+1} is used. This parameter vector is calculated by a projection mapping $P : \hat{Q} \rightarrow R^n$. Least-squares regression can be used to choose P, which produces:

$$P(Q) = \theta, where \tag{2.23}$$

$$\theta = argmin_\theta \sum (Q(x_i, u_i) - F(\theta)(x_i, u_i))^2 \tag{2.24}$$

The most common problem is ensuring convexity, and some care is required to ensure that θ exist. For example, when the approximator F is parametrized as a linear function, it is a convex quadratic optimization problem, and the respective techniques must be used in order solve the problem and find the θs. Figure 2.3 illustrates approximate Q-iteration, and the relations between the mappings of T and P, and Q-functions considered by the algorithm. Then, Algorithm 1 presents

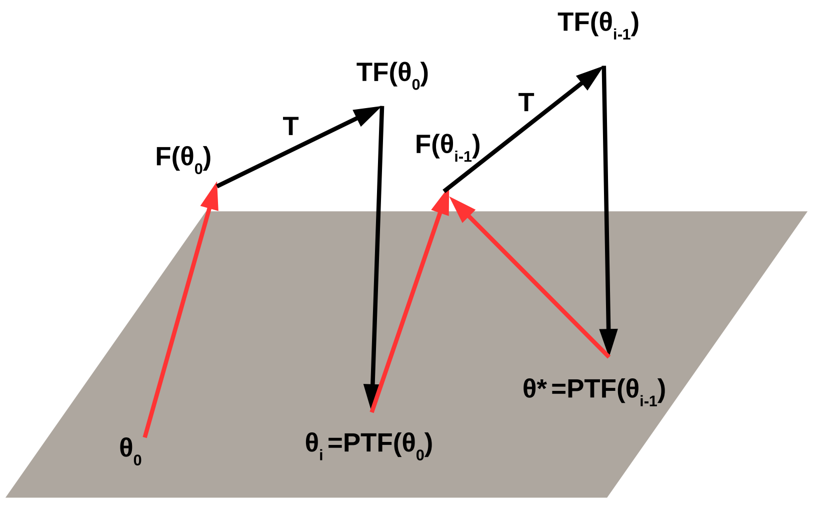


Figure 2.3: An illustration of approximate Q-iteration. At each iteration, the approximation mapping F is applied to the current parameter vector to obtain an approximate Q-function, which is then passed through the Q-iteration mapping T. The result of T is then projected back onto the parameter space with the projection mapping P. The algorithm converges to a fixed point θ^*, when passing through $P \circ T \circ F$ leads back to itself. Q-function $F(\theta^*)$ is the approximated solution [34].

an example of approximate Q-iteration for a Markov dcision process (MDP), using the least-squares projection. (We refer to [34] for more details.)

Another well known model-based value iteration algorithm is *fitted value iteration (FVI)* [27, 48]. *FVI* was the approximation algorithm of choice for us in this thesis as it is well-behaved, in the sense that by using a sufficiently large number of samples for a large class of MDPs, good performance can be achieved with high probability, as convergence rate results indicate [48].

```
i ← 0
repeat
    for $k$=1,. . . , $n_s$ samples do
        $Q_{i+1}(x_k, u_k) \leftarrow \rho(x_k, u_k) + \gamma max_{u'}\{F(\theta_i)\}$
    end
    $\theta_{i+1} \leftarrow argmin_\theta \sum(Q(x_k, u_k) - F(\theta)(x_k, u_k))^2$
    i ← i+1
until $\theta$ is satisfactory;
```

Algorithm 1: Least-squares approximate Q-iteration for deterministic MDPs [34].

Model-free value iteration with parametric approximation

Model-free algorithms for approximate value iteration do not have any prior knowledge for the transition and reward model. Algorithms from that class are can be cast as either *offline model-free approximate value iteration* or *online model-free value iteration.*

Offline model-free approximate value iteration The transition dynamics f and the reward function ρ are unknown in the case of offline model-free approximation. Only some transition samples are available:

$$(x_i, u_i, x'_i, r_i)|i = 1, ..., ns$$

where, the next state x'_i and the reward r_i are observed after taking action u_i in the state x_i. The *fitted Q-iteration* method of Algorithm 2 is an example of a model-free version of approximate Q-iteration. There are two changes wrt. the original algorithm. First, a sample-based projection mapping is taking place using only the samples (x_i, u_i), via least-squares regression. Second, due to the fact that F and ρ are not available, the updated Q-function Q_{i+1} = $(T \circ F)(\theta_i)$ cannot be computed exactly. Hence, the Q-values $Q_{i+1}(x_i, u_i)$ are approximated using some parameter variables θ_i s.t. $F(\theta_i) \approx Q_{i+1}$.

```
i ← 0
repeat
    for k=1,..., $n_s$ samples do
        $Q_{i+1}(x_k, u_k) \leftarrow r(x_k, u_k) + \gamma max_{u'}\{F(\theta_i)\}$
    end
    $\theta_{i+1} \leftarrow argmin_\theta \sum (Q_{i+1,k} - F(\theta)(x_k, u_k))^2$
    i ← i+1
until $\theta$ is satisfactory;
```

Algorithm 2: Least-squares fitted Q-iteration with parametric approximation [34].

Online model-free approximate value iteration The original Q-learning updates the Q-function with:

$$Q_{i+1}(x_i, u_i) = Q_i(x_i, u_i) + \alpha_i[r_{i+1} + \gamma max_{u'} Q_i(x_{i+1}, u') - Q_i(x_i, u_i)] \quad (2.25)$$

after observing the next state x_{i+1} and reward r_{i+1}, as a result of taking action u_i in state x_i. A straightforward way to integrate approximation in Q-learning is by using gradient descent [34]. For simplicity, we denote the approximate Q-function at time i by:

$$\hat{Q}_i(x_i, u_i) = [F(\theta_i)](x_i, u_i) \quad (2.26)$$

The algorithm aims to minimize the squared error between the optimal value Q^* and the current Q-value:

$$\theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i}\left[Q^*(x_i, u_i) - \hat{Q}(x_i, u_i)\right]^2 \quad (2.27)$$

However, in reality $Q^*(x_i, u_i)$ is not available, and it is thus replaced by an estimate derived from the Q-iteration mapping:

$$r_{i+1} + \gamma max'_u \hat{Q}_i(x_{i+1}, u')$$

The approximate Q-learning update then takes the form:

$$\theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i}\left[r_{i+1} + \gamma max_{u'} \hat{Q}_i(x_{i+1}, u') - \hat{Q}(x_i, u_i)\right]^2 \quad (2.28)$$

Actually an approximation of the temporal difference is computed. With a linearly parameterized approximator: $\phi^T(x_{i+1}, u')\theta_i$ and $\phi^T(x_i, u_i)\theta_i$, the update simplifies to:

$$\theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i} \left[r_{i+1} + \gamma max_{u'} \left(\phi^T(x_{i+1}, u')\theta_i\right) - \phi^T(x_i, u_i)\theta_i\right]^2 \quad (2.29)$$

Approximate Q-learning requires exploration. As an example, Algorithm 3 presents gradient-based Q-learning with a linear parametrization and ϵ-greedy exploration. Basically, at each time-step of this algorithm, with some small probability an exploratory action is chosen uniformly at random.

```
for $i$=1,..., $N$ time-step do
    $u_i \leftarrow \begin{cases} u \in argmax'_u(\phi^T(x_i, u_i)\theta_i), & \text{if probability } 1 - \epsilon_i. \\ \text{a uniform random action in U}, & \text{with probability } \epsilon_i. \end{cases}$
    apply $u_i$, measure next state $x_{i+1}$ and reward $r_{i+1}$
    $\theta_{i+1} \leftarrow \theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i} \left[r_{i+1} + \gamma max'_u \left(\phi^T(x_{i+1}, u')\theta_i\right) - \phi^T(x_i, u_i)\theta_i\right]^2$
end
```

Algorithm 3: Q-learning with a linear parametrization and ϵ-greedy exploration [34].

Value iteration with non-parametric approximation

In the non-parametric case, fitted Q-iteration can no longer be described using approximation and projection mappings that remain unchanged from one iteration to the next. Instead, non-parametric approximators are generated at each new iteration. Algorithm 4 outlines fitted Q-iteration with non-parametric approximation. The non-parametric regression of the algorithm is responsible for generating a new approximator Q_{i+1} that represents the updated Q-function, using information provided by the available samples.

```
i ← 0
repeat
    for k=1,..., $n_s$ samples do
        $Q_{i+1}(x_k, u_k) \leftarrow R(x_k, u_k) + \gamma max_{u'}\{Q_i(x'_k, u')\}$
    end
    find $Q_{i+1}$ using non-parametric regression on $(x_k, u_k)$,$Q_{i+1,k}$
    i ← i+1
until $Q_{i+1}$ is satisfactory;
```

Algorithm 4: Fitted Q-iteration with non-parametric approximation [34].

2.4 Decision-Making in the Smart Grid

In recent years, there have been a few works dealing with optimal decision-making in the Smart Grid. Agents are faced with decisions regarding buying and selling energy in the electric grid, increasing effectiveness of their power output, and maintaining the demand curve stability [8, 47].

Estimating the power output of inherently intermittent and potentially distributed renewable energy sources has become a major scientific and societal concern. The work [3] provides an algorithmic framework, along with an interactive web-based tool, to enable short-to-middle term forecasts of photovoltaic (PV) systems and wind generators output. We use this web-based tool to specify the current estimates about the production levels of the prosumer at a specific time step, and help the prosumer to determine its actions.

Another major problem is enhancing the effectiveness of the (RES) production methods. For instance, PV solar tracking techniques are used in solar plants. However, current techniques suffer from several drawbacks in their tracking policy: (i) they usually do not consider the forecasted or prevailing weather conditions; even when they do, they (ii) rely on complex closed-loop controllers and sophisticated instruments; and (iii) typically, they do not take the energy consumption of the trackers into account. The work [4] propose a policy iteration method (along with specialized variants), which is able to calculate near-optimal trajectories for effective and efficient day-ahead solar tracking, based on weather forecasts coming from online providers.

Besides the prediction of the power output of a Smart Grid, a main aspect of the electric grids is the maintenance of demand curve stability. As technology evolves and electricity demand rises, the task to keep it precisely balanced with supply at all times becomes especially challenging. Maintaining demand curve stability, in particular, can alleviate the risk of disastrous electricity network collapses, and leads to financial and environmental benefits as then some conventional generators can be run on idle, or even be shut down completely. The work of [8] present a directly applicable scheme for electricity consumption shifting and effective demand curve flattening. The scheme can employ the services of either individual or cooperating consumer agents alike. Agents participating in the scheme, however, are motivated to form cooperatives, in order to reduce their electricity bills via lower group prices granted for sizable consumption shifting from high to low demand time intervals.

A similar problem is faced by an alternative solution method proposed in the work [47], They propose a new scheme for efficient demand side management for the Smart Grid. Specifically, they envisage and promote the formation of cooperatives of medium-large consumers and equip them with the capability of regularly participating in the existing electricity markets by providing electricity demand reduction services to the Grid but do not deal with shifting. Based on mechanism design principles, they develop a model for such cooperatives by designing methods for estimating suitable reduction amounts, placing bids in the market and redistributing the obtained revenue amongst the member agents. The mechanism is such that the member agents have no incentive to show artificial reductions with the aim of increasing their revenues.

There are a few works that deal with the decision making of agents in market environments. For instance, TacTex [20] was the champion agent for the 2013 Power Trading Agent Competition (PowerTAC). In PowerTAC, several self-interested, autonomous agents corresponding to *brokers* compete with each other with the goal of maximizing profits through energy trading. TacTex does not model the decision making problem of a microgrid prosumer, as we do, but that of a broker simultaneously participating in tariff and wholesale markets. As such, its utility measure is the cash amount existing in a bank, while the energy amount to buy is not considered part of the decision making problem: it is simply set to the difference between predicted demand and the energy that is already procured for the targeted time period. Moreover, there are only 26 states in the MDP solved by TacTex, and a state transition leads to one of only two potential states (by contrast, we tackle MDPs with state-action spaces encompassing hundreds of thousands of elements).

Similarly to TacTex, the work of Peters *et al.* [36] also deals with optimising the long-term behaviour of broker agents during retail electricity trading. They employ the classic SARSA reinforcement learning algorithm [49] for selecting actions in a tariff market. However, it is less flexible than TacTex's tariff market strategy, which is not constrained to a finite set of actions.

We are however only aware of two papers that focus on prosumer decision-making. First, Nikovski and Zhang [19] propose a method for finding the optimal conditional operational schedule for a set of power generators, assuming stochastic electricity demand and stochastic generator output. However, in contrast to our work here, they do not tackle the problem of selling or storing the generated power. Second, Kanchev *et al.* [28] propose an energy management system which could be employed by a prosumer managing photovoltaic generators, storage units, and a gas microturbine. However, they assume a deterministic system, not accounting for uncertainty and errors that may occur during the prosumer's operation time.

Chapter 3

A Factored MDP Model for Buying and Selling Energy

The prosumer we consider in this work corresponds to a *microgrid* distributing power to a community. As such, it produces energy by means of *renewable energy sources*, and is responsible for the well-being of *residential consumers*. Moreover, the prosumer has access to *storage devices (batteries)*, which it can use to store energy for future use. Our prosumer is connected to the wider Grid, and it has to take decisions regarding the amounts of energy to purchase or sell to the Grid at pre-specified intervals during the next day. We assume that the Grid is represented by some utility company that can specify *tariffs* determining the sell and buy prices of electricity, to which the prosumer can subscribe (at any one of the aforementioned time intervals). The tariffs available to prosumers for the day-ahead are announced by the utility company at the beginning of each day. Then, the problem facing the prosumer is taking the right decisions as to which tariff to subscribe to and what amounts of energy to buy, sell, or store at any given interval of the day-ahead—so as to meet demand at a minimum cost and make a profit by selling the electricity to the utility companies.

We acknowledge that this model formulation, presented in detail below, seemingly disregards the complexity of modern and anticipated electricity markets. Indeed, prosumers could be faced with complex decisions during their simultaneous participation in markets of various types (e.g., *spot*, *forward*, *balancing*, or even *futures*). Despite this fact, we believe that solving the simpler problem of viewing the prosumer as an entity interacting with the wider electricity Grid via pre-specified tariffs determining energy prices (which, however, can be "variable" or to an extent "real-time" themselves), is key to determining behaviour in more complex business environments. In addition, ours is a model that corresponds to conceivable situations in the immediate near future, where (largely) energy self-sufficient communities will be operating their private microgrids, and only occasionally use energy from the wider Grid—essentially as a fallback strategy.

In the rest of this chapter, we first describe our factored states and actions, and present certain physical constraints they have to adhere to; and we then present the transition model, and our choices for representing the reward function so that it realistically captures the gains and costs from selling and purchasing energy. Importantly, our reward function takes into account periodic operation costs of the prosumer related with subscription to a tariff, as well as its costs because of accumulating battery life losses due to discharging. Moreover, there is nothing in the formulation below that precludes the applicability of our model and proposed solution to microgrid prosumers of a particular size or type.

3.1 Factored Representation

We now describe our problem's factored representation in detail. To begin, the factored states can be described as a multivariate random variable $\boldsymbol{s} = \langle s_i \rangle$, where each variable s_i can take a value in its domain DOM(s_i). There are three factored state variables, listed in Table 3.1. The first one, *tms*, takes as values the specific time steps at which the prosumer is able to act. Its domain is originally set to [1 ... 24] (one time step per hour in the day). However, as we later explain, we can drop this state variable altogether from the representation, and incorporate it in the problem horizon over which our value iteration method operates; moreover, we also conduct experiments that require the prosumer to act on a half-hourly basis. The second one, *bat*, corresponds to the amount of energy available in the batteries, and its domain is [0 ... $Battery_{max}$], with $Battery_{max}$ corresponding to the maximum capacity of the storage device(s). Note that *bat* is a naturally continuous state variable, but it was discretized in order to enable its processing by existing FMDP solvers (such as SPUDD). Finally, *tf* corresponds to the tariff the prosumer has assigned to at the moment, and its domain is the enumerated tariffs that the utility offers during the day. That is, DOM(*tf*)=$\{tf_1, \cdots, tf_i, \cdots, tf_K\}$, with K being the number of tariffs available on a specific day. Each tf_i tariff is characterized by a buying and a selling price, denoted $buying_i$ and $selling_i$ respectively, and communicated to the prosumer via external signals.

Features	Denoted	Description
time-step	*tms*	current time step within the operational day
battery	*bat*	amount of stored energy (at a specific time step)
tariff	*tf*	tariff currently in effect (at a specific time step)

Table 3.1: Factored states.

Then, actions can be described as a multivariate random variable $\boldsymbol{a} = \langle a_i \rangle$ where each variable a_i affects the transition from some factored state to another, and takes a value in its domain DOM(a_i). The discretization for each DOM(a_i) is performed dynamically: it is based on the discretization of the DOM(s_i) domains, in a way that from any given state, actions can lead to any other.

There are three factored actions. First, action *buy*, which describes the amount of energy bought from the electric utility. Positive values for *buy* denote the actual buying of energy from the utility, while negative values mean the prosumer sells energy to the utility. With $Load_{max}$ being the maximum total expected residential consumption load, and the nominal power generating capacity of the renewable energy sources denoted by RES_{nom}, the domain for *buy* is set to [-RES_{nom} ... $Load_{max}$]. Second, factored action *chg*, which signifies the attempt to store an amount of energy to the batteries. Its value range is [-$Battery_{max}$ ... $Battery_{max}$]. Positive values represent charging the battery, and negative values represent discharging the battery. Finally, the third action, *sel*$_{tf}$, corresponds to a selection of tariff by the prosumer. Its domain is [0 ... K]. The value 0 signifies a choice to remain attached to its current tariff, while values 1 to K signify a choice to move to some other of the K tariffs available.[1]

Actions	Denoted	Description
buy	*buy*	buy from the utility
charge	*chg*	charge battery
select tariff	*sel*$_{tf}$	select a tariff to subscribe to

Table 3.2: Factored actions.

[1]The additional ‘stay-with-current-tariff’ action is required as subscribing and resubscribing would entail a subscription cost (thus the action protects the prosumer from that cost).

Now, there are three types of external signals the prosumer receives. These are listed in Table 3.3, and can be described as multivariate random variable $\boldsymbol{sg} = \langle signal_i \rangle$ where each variable $signal_i$ can take a value in its domain DOM($signal_i$). The prosumer employs these signals to help it determine its actions.

The first signal type, *prod*, specify the production levels of the prosumer at a specific time step. Its domain are defined given the RES_{nom} values introduced above, DOM(*prod*)= [0 … RES_{nom}]. We employ *RENES* [3], a web-based PVS and WTG production prediction tool, to obtain the production estimates of the photovoltaic systems (PVS) and wind turbine generators (WTG) of our microgrid. It is extensively described in section 3.5.1.

The second signal type *cons*, specify the current estimates about the consumption levels of the prosumer at a specific time step. Their domains are defined given $Load_{max}$ values introduced above. Thus, DOM(*cons*)=[0 … $Load_{max}$]. We used regression methods to predict the load consumption of the prosumer. Specifically, we used Gaussian Process (GP) and Bayesian Linear Regression. The input data of our model, and the output data whose values we are trying to predict, correspond to the factored state *tms* and the signal variable *cons*. The prediction process is presented in section 3.5.2.

Finally, the third signal type, $price_{tf}$, specifies, once a day, the buy and sell prices ($buying_i$ and $selling_i$) for each one of the K tariffs, and for each t time step of the day ahead. This signal affects the reward representation of our model, which is described in section 3.4

Signals	Denoted	Description
production	*prod*	predicted levels of energy production
consumption	*cons*	predicted levels of energy consumption
$\{\mathrm{buying_i}, \mathrm{selling_i}\}$	$price_{tf}$	buying and selling price for tariff i

Table 3.3: Signal types.

Notice that all factored variables in our formulation are independent of the size of the prosumer microgrid—i.e., they are not affected by the number of generators or homes populating it. Moreover, despite the complexity of the problem, the temporal dependencies among the state variables in our model are in fact quite simple, as seen in the 2-stages temporal Bayesian network (2-TBN) of Fig. 3.1.

It can be seen there that a variable value at $t+1$ depends only on the variable's value at t (with value changes triggered, for *bat* and *tf*, by actions *chg* and sel_{tf} respectively).

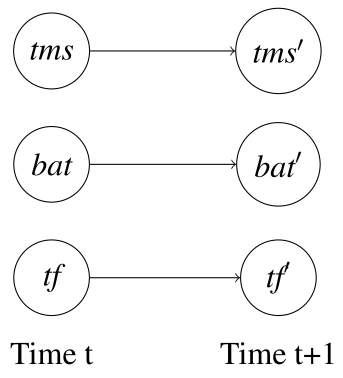


Figure 3.1: Temporal dependencies among state variables.

In what follows, we use the notation x_t to denote the value of a state, action, or signal variable at time t.

3.2 Constraints

There are certain constraints that our state and action variables must adhere to. First, in a setting involving energy exchanges, the *balance energy constraint* [18, 54] must be respected at all times. This means that, at any time step t, power produced (including that bought) should match power consumed (including that stored):

$$prod_t - cons_t - chg_t + buy_t = 0 \tag{3.1}$$

The second constraint refers to the storage unit(s) of the prosumer. A storage unit cannot be charged over its capacity:

$$chg_t \leq Battery_{max} - bat_t \tag{3.2}$$

Similarly, the energy quantity discharged from a unit cannot exceed that currently stored in the unit:

$$-chg_t \leq bat_t \tag{3.3}$$

Finally, for safety reasons, the battery storage level must be always kept between 20% and 100% [29]:

$$0.2 \leq bat_t/Battery_{max} \leq 1 \tag{3.4}$$

3.3 Transition Function

State transitions in our model will be in general stochastic, since faults may occur while taking actions like charging or discharging the storage devices and buying or selling energy to the utility. The variable *tms* is an exception to this rule—since one specific time step is always followed by the next one. That is, $Pr(tms_{t+1} = t+1|tms_t = t) = 1$. For the rest of the variables, we define certain *bounded regions* (with distinct boundaries for each variable), which include a subset of discrete factored states lying close to the factored state intended to transition to by performing a factored action taken at time t. The boundaries can be set to any values required.

Thus, (factored) actions are assumed to have the intended result with some probability p (arbitrarily set to 0.9 in our experiments); while, with probability $1-p$, they transition to some (factored) state within the bounded region (chosen uniformly at random). For instance, assuming that N *bat* states lie within a pre-specified bound_{bat} bounded region, the action of charging the battery with an energy amount c at time t (action $chg_t = c$) is successful with probability p:

$$Pr(bat_{t+1} = bat_t + c \mid chg_t = c, bat_t) = p$$

whereas with probability $1-p$ it fails, leading to any potential factored battery state within the bound_{bat} region:

$$Pr(bat_{t+1} = bat \in \text{bound}_{bat} \mid chg_t = c, bat_t) = (1-p)/N$$

Since distinct factored actions can be simultaneously utilized—i.e., the prosumer can select a new tariff, buy energy, and charge the battery at the same time step t— the overall transition probability is given by Eq. 3.5 as follows.

$$Pr(\mathit{tms}_{t+1}, \mathit{bat}_{t+1}, \mathit{tf}_{t+1} | \mathit{tms}_t, \mathit{bat}_t, \mathit{tf}_t, \mathit{chg}_t, \mathit{sel}_{\mathit{tf},t}) = Pr(\mathit{bat}_{t+1} | \mathit{bat}_t, \mathit{chg}_t) \cdot Pr(\mathit{tf}_{t+1} | \mathit{tf}_t, \mathit{sel}_{\mathit{tf},t}) \quad (3.5)$$

given that the battery level at any time step depends on the previous battery level state and on whether a *chg* action was used, while the tariff in place is affected by a tariff selection action. Notice also that, in our model, buying or selling energy does not have a direct effect on a state variable, thus no state transitions need to be defined for action *buy*. It is thus implicitly assumed that *buy* (a positive or negative energy amount) always succeeds. This assumption is quite realistic, and it is motivated from the need to respect the constraint in Eq. 3.1 above: choosing how much energy to buy/sell depends on the production and consumption estimates, and on the results of charging the battery. In practice, the latter is an action whose outcome is indeed more uncertain than that of buying/selling energy.

3.4 Factored Reward Representation

The next step is to determine the reward function for our factored MDP. The reward function is associated with *(a)* either the financial gain from selling power to the utility or the financial cost of buying power in a certain price; *(b)* the running financial costs for being subscribed to a tariff; and *(c)* the operation financial costs of using the storage devices. As such, we choose to represent the Markovian reward function as a cost function with three main components. Specifically, the function describing the immediate cost for a transition from state $\boldsymbol{s}_t$ to $\boldsymbol{s}'_{t+1}$ by executing some $\boldsymbol{a}_t$ at time-step t, is defined as follows:

$$Cost(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}'_{t+1}) = C_{\mathit{energy}}(\boldsymbol{s}'_{t+1}, \boldsymbol{a}_t) + C_{\mathit{period}}(\boldsymbol{s}'_{t+1}) + C_{\mathit{bl}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \quad (3.6)$$

We now explain its components in turn. The first component, C_{energy}, captures the cost per Wh for buying electricity (or the profits from selling it to the utility), given the buy/sell rates prescribed by the tariff in effect:

$$C_{energy}(tf_{t+1}, buy_t) = \begin{cases} buy_t \cdot \text{buying}_{tf_{t+1}} & \text{if } buy_t \geq 0 \\ buy_t \cdot \text{selling}_{tf_{t+1}} & \text{if } buy_t < 0 \end{cases} \tag{3.7}$$

The second component captures the periodic costs $C_{periodic}$ inflicted on the prosumer for being subscribed into a tariff. Naturally, one would expect that "better" tariffs for a prosumer—that is, tariffs specifying high selling prices and low buy prices—will actually incur higher periodic costs (flat rates). Due to this, in our model we make the assumption that periodic costs drop exponentially with decreasing tariff quality (i.e., as the difference between buying price and selling price increases):

$$C_{period}(tf_{t+1}, price_{tf}^{t+1}) = C_1 \exp\{-C_2 \cdot (\text{buying}_{tf}^{t+1} - \text{selling}_{tf}^{t+1})\} \tag{3.8}$$

with C_1 = 0.013, C_2 = -2.7. The function is plotted in Fig. 3.2.

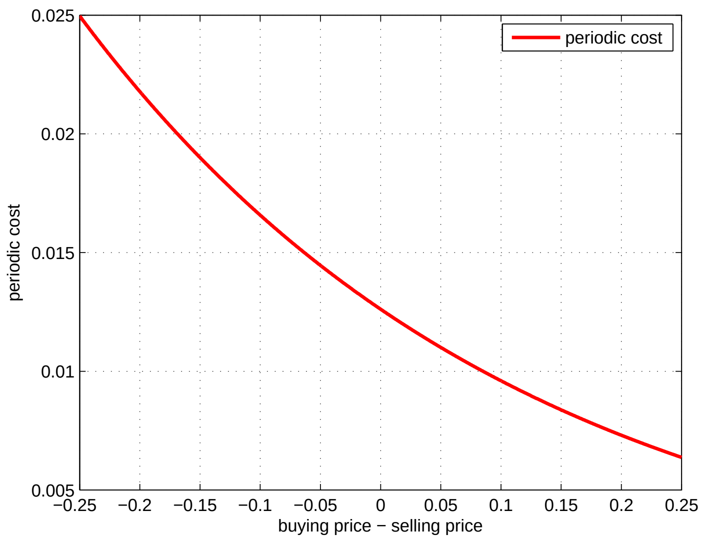


Figure 3.2: Periodic costs as a function of tariff quality.

The third component of the cost function, C_{bl}, captures the costs associated with *battery life losses*. That is, the costs inflicted from charging (or discharging) the storage devices (batteries) with a charge amount of chg_t, at a given time-step t when the stored energy amount is at bat_t. To estimate this component, we assume the use of deep-cycle batteries, which are lead-acid batteries designed to be regularly deeply discharged (using most of their capacity) [7].

The C_{bl} cost of an attempted *chg* action can then be viewed as a fraction of the $C_{init-bat}$ *initial investment cost* for the batteries:

$$C_{bl} = L_{loss} \cdot C_{init-bat} \tag{3.9}$$

The "life loss" L_{loss} factor in the above equation is affected by the *effective throughput* A_c of the battery over a certain charge period (measured in Ah) [7]:

$$L_{loss} = \frac{A_c}{A_{total}}$$

Here, A_{total} is the total cumulative throughput (in Ah) during the battery's lifetime. A battery size of $Q\ Ah$ will deliver an effective $A_{total} = 390 \cdot QAh$ over its lifetime [7].

Now, A_c above related to the operating *state of charge* (SOC) and the *actual throughput* A'_c. The latter can be calculated, given the voltage of the battery, as:

$$A'_c = \frac{chg_t}{V_{battery}}$$

To calculate A_c, we first have to define the *state of charge* (SOC) of the battery, as the fraction of its total $Battery_{max}$ capacity covered by its currently stored energy amount, bat_t:

$$SOC = \frac{bat_t}{Battery_{max}}$$

and its value has to be kept always between 0.2 and 1, for safety reasons [7]. A_c is then expressed as

$$A_c = \lambda_{soc} A'_c$$

where λ_{soc} is an *effective weighting factor* given the battery's state of charge. When SOC is between 0.2 and 1, λ_{soc} is approximately linear with SOC [7], which can be expressed as

$$\lambda_{soc} = k \cdot SOC + d$$

In our work, the values of k and d in the previous equation were set to -0.7594 and 1.43 respectively, as a result of applying linear fitting over certain empirically set (SOC, λ_{SOC}) data points reported in [7]. The resulting fitted line is depicted at Fig. 3.3.

With λ_{soc} at hand, we can then fully determine the C_{bl} component, and use it to determine the life loss cost incurred on batteries during their charge (or discharge) by the application of a chg_t action at time-step t.

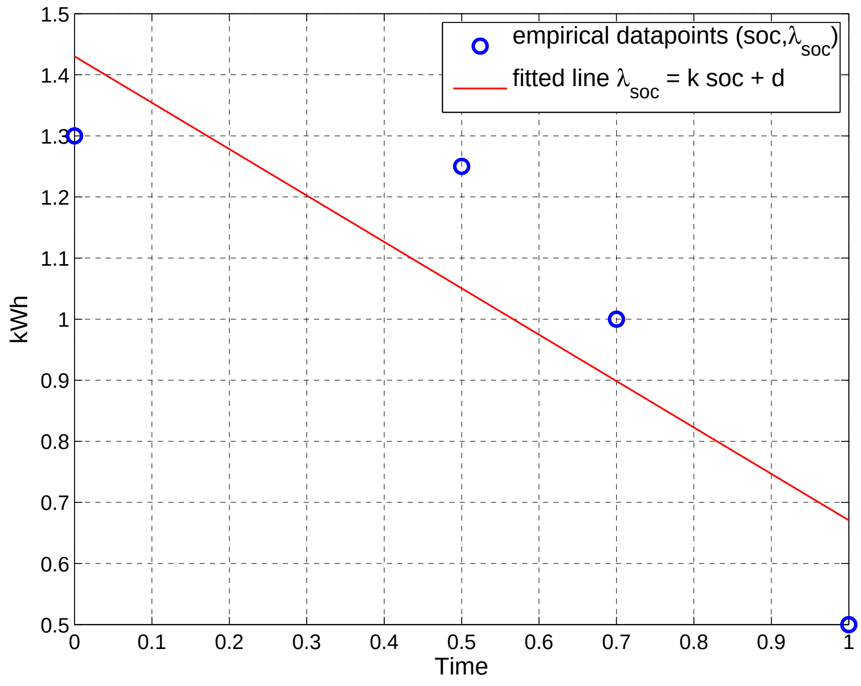


Figure 3.3: Estimating the λ_{soc} weighting factor.

3.5 Prosumer Production and Consumption Models

Naturally, the estimated production from the renewable energy sources distributed on the microgrid, and the predicted load consumption of the connected consumers, affect the policy of the prosumer. The prosumer is notified about the expected production and consumption values via the *prod* and *cons* signals. Thus, it is necessary to predict values for those signals that are as accurate as possible, to assist the decision-making process of the prosumer.

3.5.1 Production Prediction

To obtain the production estimates of the photovoltaic systems (PVS) and wind turbine generators (WTG) of our microgrid, we employ *RENES* [3], a web-based PVS and WTG production prediction tool. RENES generates PVS and WTG production estimates given time, geographical coordinates and online weather forecasts, and it comes with specific performance guarantees. For PVS production predictions, RENES utilizes non-linear approximation components for turning cloud-coverage into radiation forecasts, which are then used for production prediction. It has an interactive web-based interface, along with an API providing XML responses to prediction requests. New production estimates are provided every half an hour. RENES predictions are provided free-of-charge. The tool and API can be accessed at www.intelligence.tuc.gr/renes/ .

3.5.2 Consumption Prediction

We now we show how to employ two regression methods to predict the load consumption of the prosumer: Gaussian Process (GP) and Bayesian Linear Regression. To begin, the input data of our model, and the output data whose values we are trying to predict, correspond to the factored state *tms* and the signal variable *cons* :

$$\boldsymbol{x} = (\mathit{tms}_1, \ldots, \mathit{tms}_n) \tag{3.10}$$

$$\boldsymbol{y} = (cons_1, \ldots, cons_n) \tag{3.11}$$

that is, they are sequences of consumption data, containing information about time-steps and the respective load consumption.

Our goal in regression, is to make predictions of the target variables for new inputs. Given a set of output data

$$\boldsymbol{y} = (y_1, \ldots, y_n)^T$$

corresponding to input values (x_1, …, x_n), where n is the length of the time sequence we use, we predict the target variable y_{n+1} for a new input vector with an additional x_{n+1} value.

Bayesian Linear Regression The first method we use for prediction is *Bayesian linear regression*. To begin, we define a model parameter $\boldsymbol{w}$

$$\boldsymbol{w} = [\boldsymbol{x}\ \boldsymbol{y}]$$

with x, y as in Eqs. 3.10 and 3.11 above. For a set of training samples, $\mathcal{D} = \{(x_j, y_j), j = 1, ..., n\}$ (x_j inputs and y_j outputs) we need to predict the posterior distribution of w given the target values y.

Now, the conjugate prior of $\boldsymbol{w}$ is a *Gaussian distribution*:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\mu_0, \sigma_0^2)$$

where μ_0 is the mean and σ_0^2 the variance noise; while the *likelihood function* $p(y|w)$ is given also by a *Gaussian distribution* of the form

$$p(\boldsymbol{y}|\boldsymbol{w}) = \mathcal{N}(\boldsymbol{y}|\ \Phi\boldsymbol{w}, \beta^{-1}I)$$

where β is noise single precision parameter, and Φ is a polynomial basis function.

With conjugate prior and likelihood function at hand, the *posterior distribution* is computed using Bayes theorem for Gaussians [16]. In order to find the posterior distribution, we just require the mean and the variance:

$$p(\boldsymbol{w}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{w}|\mu_n, S_n),\ where$$

$$\mu_n = S_n(S_0^{-1}\mu_0 + \beta\Phi^T\boldsymbol{y})$$
$$S_n^{-1} = S_0^{-1} + \beta\Phi^T\Phi$$

In this work, we adopt a zero-mean isotropic Gaussian, governed by a single precision parameter α, so that:

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(w|0, \alpha^{-1}I)$$

Then, the corresponding posterior distribution $p(\boldsymbol{w}|\boldsymbol{y})$ has:

$$\mu_n = \beta S_n\Phi^T\boldsymbol{y}$$

$$S_n^{-1} = \alpha I + \beta\Phi^T\Phi$$

Evidence approximation [16] is utilised to calculate the optimal values of the hyper-parameters α and β.

Gaussian Process Regression The second regression method that we use is Gaussian Process (GP) with two form of kernels, a gaussian and a polynomial one. The use of a GP with a Gaussian kernel appears to be the better choice for our setting, as we demonstrate in Sec. 3.5.3 below. We note that Gaussian Processes have also been recently applied for consumption reduction prediction in electricity demand management settings [9, 31, 5].

Gaussian processes can be used for regression and classification without a parametric model assumption. For a set $\mathcal{D} = \{(x_j, y_j), j = 1, ..., n\}$ of training samples, with x_j inputs and y_j noisy outputs, we need to predict the distribution of the noisy output at some test locations. We assume the model:

$$y_j = f(x_j) + \epsilon_j,\ \text{where}\ \epsilon_j \sim \mathcal{N}(0, \sigma_{noise}^2)$$

with σ_{noise}^2 the variance noise.

GP regression is a Bayesian approach that assumes a priori that function values follow: $p(\mathbf{f}|x_1, x_2, ..., x_n) = \mathcal{N}(\mathbf{0}, K)$ where $\mathbf{f} = [f_1, f_2, ..., f_n]^T$ is the vector of latent function values, $f_j = f(x_j)$ and K is the covariance matrix that is computed by a "kernel" covariance function $k(\cdot, \cdot)$: $K_{jk} = k(x_j, x_k)$.

The kernel functions used in this work are given by a polynomial and Gaussian form [16] respectively:

$$k(x_j, x_k) = \theta_0 + \theta_1(x_j^T x_k) + \theta_2(x_j^T x_k)^2$$

$$k(x_j, x_k) = \theta_0 \ \exp\big(- \frac{(x_j - x_k)^T(x_j - x_k)}{2(\theta_1)^2}\big)$$

where the θ_* are the model's hyper parameters. Their optimal values can be found by maximizing the log likelihood [16], for instance using *backtracking line search* [52], as we do in this work.

Finally, in order to proceed to the inference, we must combine the joint GP prior obtained by the test values with the likelihood $p(\mathbf{y}|\mathbf{f})$, via Bayes rule. The joint GP prior and the independent likelihood are both Gaussian with mean and variance at a test point x_* as follows:

$$GP_\mu(x_*, \mathcal{D}) = K_{*,f}(K_{f,f} + \sigma^2_{noise} I)^{-1}\mathbf{y} \tag{3.12a}$$

$$GP_\sigma(x_*, \mathcal{D}) = K_{*,*} - K_{*,f}(K_{f,f} + \sigma^2_{noise} I)^{-1}K_{f,*} \tag{3.12b}$$

3.5.3 Comparing Regression Methods

To choose a Φ polynomial basis function to use for Bayesian linear regression (BLR), we performed cross-validation with random sub-sampling repeated 10 times for different polynomial functions [46]. For this, we split our consumption dataset (described in Section 5 below) to an 80% part for training, and a 20% one for testing. The results, in terms of *mean square error (MSE)*, are shown of Table 3.4. The degree of the polynomial with the minimum average MSE is D=5: thus, this was the polynomial of choice for *BLR*. We then compared the performance of *BLR* against that of a Gaussian process (GP) that employed either the polynomial (GP-poly) or the Gaussian (GP-G) kernel mentioned above. The prediction performance of the methods is depicted in Fig. 3.4; and Table 3.5 contains the methods' MSE. Results show that *GP-G* does much better than *GP-poly* and *BLR* in terms of MSE, achieving a quite low MSE value. Moreover, the prediction mean of the *GP-G* method apparently follows more closely the actual consumption pattern, as emerging from the actual (x, y) data points. Hence, we choose the Gaussian process with the Gaussian kernel method to obtain the load consumption estimates of the prosumer.

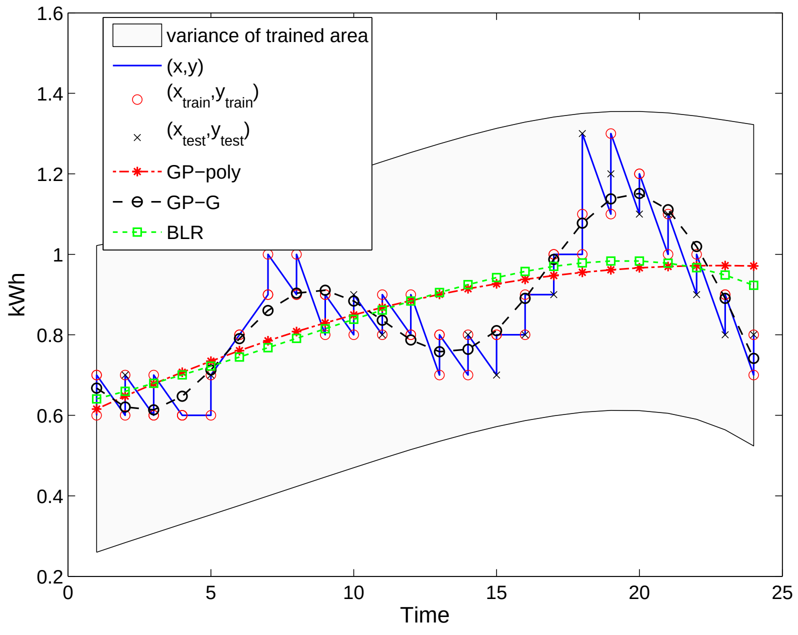


Figure 3.4: Prediction performance of *GP-poly*, *GP-G*, and *BLR*. The (x, y) input-target pairs are actual consumption data points. The *GP-G* mean matches the (typical) daily electricity demand curve of our dataset, with two consumption peaks.

Degree of Polynomial	MSE
1	0.022372
2	0.021312
3	0.020175
4	0.017679
5	0.016861
6	0.017329
7	0.017355
8	0.017167
9	0.017399
10	0.017611

Table 3.4: MSE of Bayesian linear regression Φ functions.

GP with polynomial kernel (GP-poly)	0.0173
GP with Gaussian kernel (GP-G)	0.006943
Bayesian linear Regression (BLR)	0.0169

Table 3.5: MSE of GP & Bayesian Linear Regression.

Chapter 4

Solving the Factored MDP

With the above FMDP at hand, the optimal policy can be derived by solving the corresponding Bellman equations. Dynamic programming (DP) methods can be used to obtain the optimal solution [49]. In our work here we used *value iteration (VI)* as the DP method of choice, for discrete state MDP, while for continuous state MDP we used *approximate value iteration (AVI)*.

4.1 Solving the Discrete-State MDP

Our problem is naturally a finite-horizon problem, thus we employed a finite-horizon VI method. By setting the horizon T to be equal to the number of time steps at which the prosumer is required to act, we can incorporate the *tms* factored state into the problem's horizon, thus effectively reducing the size of the state space. In addition, we used a sel_{tf} action, which corresponds to a selection of tariff by the prosumer. Tariffs can be key to group together a range of consumer preferences, that would have had to be represented by distinct state or action variables otherwise. For instance, one would have wished to represent preferences to consume when buying prices are low, e.g. at night, and sell when selling prices are high—and distinct sell and buy variables would have been required to allow this. Tariffs could potentially incorporate more information, such as special discounts, and so on. Thus, the use of tariffs can be key at reducing the state-action space in such problems. These choices allow us to represent the problem compactly, and provide an exact optimal solution via dynamic programming.

Then, with $\boldsymbol{s}'_t$ denoting the potential successor states of $\boldsymbol{s}_t$; with $Pr(\boldsymbol{s}'_{t+1}\,|\boldsymbol{a}_t, \boldsymbol{s}_t)$ denoting the probability of state transitions from $\boldsymbol{s}_t$ to possible successor states $\boldsymbol{s}'_{t+1}$, given that action $\boldsymbol{a}_t$ was taken; and $R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}'_{t+1}) = -Cost(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}'_{t+1})$ denoting the corresponding immediate reward (the negative immediate cost), the

```
for all instantiations of $\boldsymbol{s}$ do
    set $V_{T+1}(\boldsymbol{s}) = 0$
end
for all time-steps $t$ in descending order
    (i.e., with $1, \cdots, T$ stages-to-go) do
    for all instantiations of $\boldsymbol{s}_t$ do
        $V_t(\boldsymbol{s}_t) \leftarrow \min_{\boldsymbol{a}_t} \sum_{\boldsymbol{s}'_{t+1}} Pr(\boldsymbol{s}'_{t+1} \,|\boldsymbol{a}_t, \boldsymbol{s}_t) \cdot \big(R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}'_{t+1}) + V_{t+1}(\boldsymbol{s}'_{t+1})\big)$
    end
end
for all instantiations of $\boldsymbol{s}$ and all time-steps $t$ do
    $\pi(\boldsymbol{s}, t) = \arg\min_{\boldsymbol{a}} \sum_{\boldsymbol{s}'} Pr(\boldsymbol{s}' \,|\boldsymbol{a}, \boldsymbol{s})\,(R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + V_{t+1}(\boldsymbol{s}'))$
end
```

Algorithm 5: Value iteration for solving the FMDP.

VI algorithm iteratively estimates the value function for the factored states, and outputs an optimal policy π, as shown in Alg. 5. Interestingly, our experiments confirm that our formulation permits VI to provide us with the solution within a reasonable time, when run on everyday desktops or laptops. While, at the same time, SPUDD sometimes fails to compute a solution within a reasonable time, when taking its pre-processing requirements into account. We will discuss our experimental results in length in Chapter 5.

4.2 Solving the Continuous-State MDP

The above solution method is used to solve decision making problems with discrete state spaces. However, approximate solution methods are used for continuous state spaces. We took some samples of the state space and we used regression in order to approximate the value function of the model. More specifically, we solve the prosumer decision problem using a model-based value iteration approximation method. Model-based value iteration is appropriate for the task, due to our prior knowledge of the transition and reward models. Specifically, our method of choice is *fitted value iteration (FVI)* [27, 48, 1], a sampling-based approximation method that is known to be well-behaving, as mentioned in Section 2.3.8 above. In principle, approximating the value function, and thus generalizing the MDP solution to continuous spaces, could lead to improved decisions quality. We now describe the method in some detail.

The decision problem of the prosumer has a continuous state space $S = R^n$, but we will assume that the action space A is discrete. In traditional value iteration operating over continuous state spaces, one needs to perform the following update:

$$V(\boldsymbol{s}) \leftarrow \min_{\boldsymbol{a}} \int_{\boldsymbol{s}'} Pr(\boldsymbol{s}' \,|\boldsymbol{a}, \boldsymbol{s}) \cdot (-R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + V(\boldsymbol{s}')) \,.$$

The main idea of fitted value iteration is to approximately carry out this step over a finite sample of states $s^{(1)}$,..., $s^{(m)}$. Specifically, we can use a supervised learning algorithm–linear regression in our description below–to approximate the value function as a linear function of the states:

$$V(\boldsymbol{s}) = \theta^T \phi(\boldsymbol{s})$$

Thus, to approximate the value function, one needs to obtain the parameters θ and the basis functions ϕ, where ϕ is some appropriate feature mapping of the states. For each state $\boldsymbol{s}$ in our finite sample of m states, fitted value iteration will first compute a quantity y , which will be our approximation to

$$\int_{\boldsymbol{s}'} Pr(\boldsymbol{s}' \,|\boldsymbol{a}, \boldsymbol{s}) \cdot (-R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + V(\boldsymbol{s}'))$$

Then, it employs some supervised learning algorithm, for instance linear regression, to get $V(\boldsymbol{s})$ close to y. In detail, the method is as described in Algorithm 6.

Fitted value iteration does not provably always converge. However, in practice, it often does converge (or at least approximately converge) [48]. If one uses a deterministic MDP model, then fitted value iteration can be simplified by setting $k = 1$ in the above algorithm. This is because the expectation becomes an expectation over a deterministic distribution, and so a single iteration is sufficient to exactly compute that expectation.

```
Randomly sample m states $s^{(1)}$,..., $s^{(m)} \in S$
for k iterations do
    for each horizon h do
        for each sampled state s do
            for each action a do
                sample state transitions $s'$
                q(a) = $\frac{1}{m} \sum ( -R_h$(s,a) + $\gamma$ $V_{h-1}(s')$ )
            end
            $y_h$(s) = min(q(a))    % min because reward corresponds to costs
        end
        $\theta \leftarrow argmin_\theta \frac{1}{2} \sum (\theta^T \phi_h(s,a) - y_h(s))^2$
        $V_h$(s) = $\theta^T \phi_h(s,a)$
    end
end
```

Algorithm 6: Fitted Value Iteration with finite horizon. Algorithm description based on the pseudocode in Andrew Ng's lecture notes in http://cs229.stanford.edu/notes/cs229-notes12.pdf.

Now, in order to find the optimal parameters θ, we have to solve the equation:

$$\theta \leftarrow argmin_\theta \frac{1}{2} \sum (\theta^T \phi_h(s,a) - y_h(s))^2$$

This is an optimization problem and we employ least linear square optimization to this purpose. IBM CPLEX provides us with a high performance optimizer to solve such optimization problems. Selecting the basis functions ϕ, on the other hand, can require extensive experimentation, in order to choose the ones whose use results to the best performance in a given setting. In our case, we evaluated several candidate basis functions, as we report in Section 5 below.

Chapter 5

Experiments and Results

We evaluate our model by examining a residential prosumer at New Hampshire, New England, northeastern United States. The data used in our prediction of residential load consumption for the area, comes from the Public Service Company of New Hampshire, and is freely available in their website (http://www.psnh.com/). Our simulated prosumer serves 30 households and includes 20 photovoltaic arrays with nominal power 60kW, 2 windturbines with nominal power 1000kW and 24 deep cycle 12Volts batteries 212AH C20 / FMD200 – VRLA/AGM, with cost of each battery 269,00 €. Estimated battery lifetime is 10-12 years. As mentioned earlier, we employ RENES (www.intelligence.tuc.gr/renes/) to obtain predictions regarding the power production of the prosumer's renewable energy generators; the services provided by RENES are also free of charge. Our simulations were conducted with data regarding a specific day-ahead (24 / 10 / 2014), at which date the predicted electricity consumption and electricity production profile of the particular residential prosumer was as presented in Fig. 5.1. All experiments were conducted on a 2.10 GHz x 4 Intel Core i3-2310M processor, with 8GB of memory. We now proceed to present our experiments over discrete and continuous state spaces.

We adopted the following discretisation for our state and action variables (signals are not discretised, but simply communicate the production and consumption predictions, and tariff characteristics to the prosumer). The discretisation step size is shown inside the range of the factored state *bat* (corresponding to the prosumer's batteries' array), and the action *chg* below:

$$\textit{bat} = [0kWh \; : \; 1kWh \; : \; 60kWh]$$

$$\textit{chg} = [-60kWh \; : \; 1kWh \; : \; 60kWh]$$

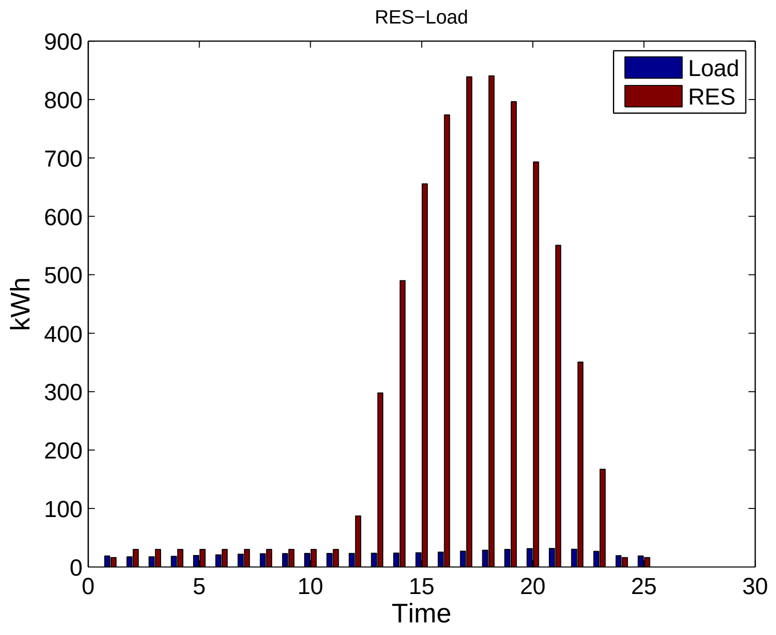


Figure 5.1: Predicted production of renewable energy sources (RES) and predicted load consumption of prosumer (Load).

We also defined nine tariffs, which are as follows:

$$tf_1 = \{0.1€, 0.1€\} \quad tf_4 = \{0.2€, 0.1€\} \quad tf_7 = \{0.3€, 0.1€\}$$

$$tf_2 = \{0.1€, 0.2€\} \quad tf_5 = \{0.2€, 0.2€\} \quad tf_8 = \{0.3€, 0.2€\}$$

$$tf_3 = \{0.1€, 0.3€\} \quad tf_6 = \{0.2€, 0.3€\} \quad tf_9 = \{0.3€, 0.3€\}$$

which thus give rise to 10 possible sel_{tf} tariff selection actions (9 corresponding to choosing one of the tariffs+1 for choosing to stay with their current one).

The transition boundaries for our state variables were initially set to boundary$_{bat}$=1kWh and boundary$_{tf}$=0.1€. Given those boundaries, the maximum number of transitions leading from one state to another are ~ 15.

5.1 Evaluation of our exact Value Iteration Method

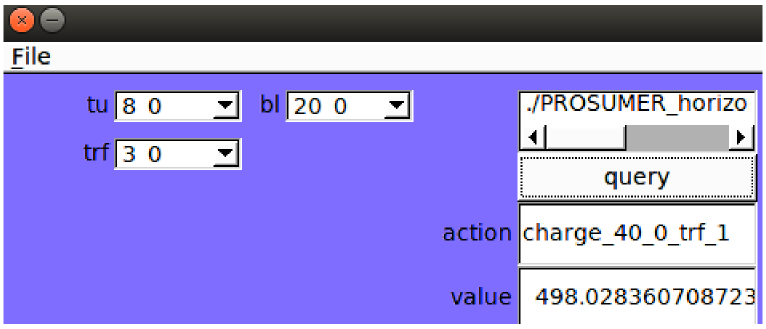
Now, the discretisation above resulted to a state-action space size of $|S \times A| = 664290$. Notice, that in order for SPUDD to be able to model our problem, we need to include state variable, *tms* , due to the fact that SPUDD does not allow us to incorporate the time-step into the problem horizon, but requires a complete representation for all states. Simply put, formulating the problem requires the states to be "stamped" by the time-step, so as to keep them distinct from each other, for the SPUDD solver to be able to operate upon the representation. This results to increased setup generation pre-processing time, as shown in Table 5.1. By contrast our exact VI algorithm can operate without this variable as it is incorporated into the problem's horizon (as explained in section 4.1). Thus, in the case of SPUDD, state-action spaces shown in Table 5.1 are expanded by a factor of $|DOM(tms)|$ (i.e., 24 or 48, for our experiments). We also note that a policy extracted by SPUDD can be presented through policy diagrams and the *pquery* SPUDD GUI tool. Figure 5.2 provides an insight on how such diagrams look like, for a toy example (a smaller instance of our problem).

Horizon	$\lvert S \times A \rvert$	bounded region size	value iteration (hours)	SPUDD (hours)	
				Script	Runtime
24	664290	15	1.76	13.4992	0.184
		90	15.84	46.9188	1.19
	2624490	15	8.7603	36.98	0.73975
48	664290	15	3.5	16.8221	0.4271

Table 5.1: Running time of value iteration and SPUDD for four different scenarios. "Script" refers to the pre-processing time required for the SPUDD input files to be generated, while "Runtime" denotes the subsequent SPUDD execution time.

We compared SPUDD to value iteration for our discretisation, and observed that the optimal policy computed through value iteration and SPUDD for the day-ahead coincide with each other. Nevertheless, value iteration produced the optimal policy in approximately 15% of the required time for SPUDD to extract the same policy. The exact running times are presented in Table 5.1.

Following our initial experiments, we increased the size of the transition boundaries so as to contain 90 state variables instead of 15. The boundaries used for the



tu 8 0
bl 20 0
trf 3 0
./PROSUMER_horizo
action charge_40_0_trf_1
value 498.028360708723

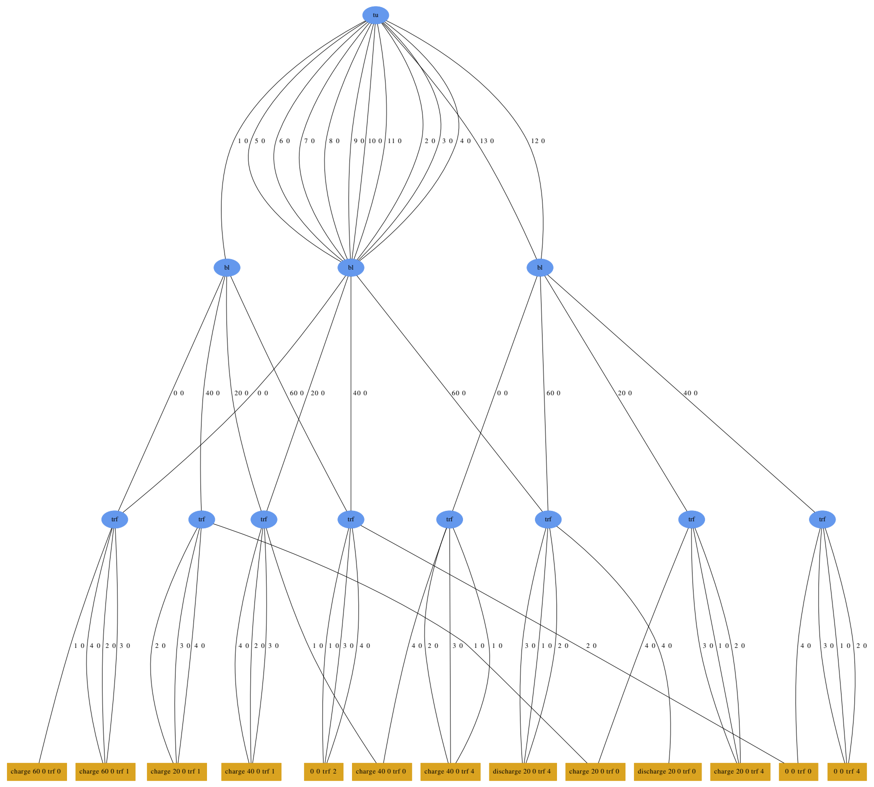


Figure 5.2: SPUDD's optimal policy (below), for a toy example with $|S| \cdot |A| = 63000$, and 15 factored states at most within any bounded region. Running time to create the script was: 29.7 sec and to execute it: 1.46 sec. Variables tu, bl and trf presented in pquery correspond to the factored states *tms*, *bat* and *tf* respectively. Variables are presented in blue bubbles, with factored actions in yellow squares, e.g. $charge\_40\_0\_trf\_1$ represents the actions *chg*= 40kWh and sel_{tf} = 1. The pquery GUI tool is shown above.

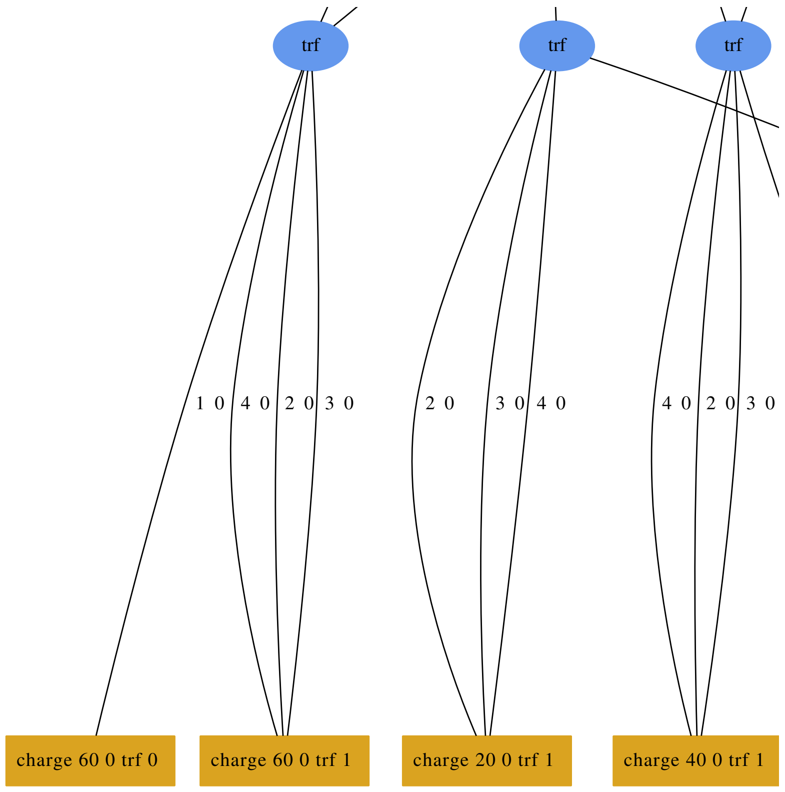


Figure 5.3: Part of SPUDD's optimal policy 5.2.

transitions from one state to another in this case are: $boundary_{bat}$=10kWh and $boundary_{tf}$=0.2€. SPUDD could not produce a solution within the time that our "planning-for-the-day-ahead" problem must be solved (maximum 24 hours). This is largely due to the fact that SPUDD has to operate on large input scripts and requires more than 46 hours for pre-processing. By contrast, the running time for the simple value iteration method was 15.84 hours, as shown in Table 5.1.

We then increased the size of state and action spaces to $|S \times A| = 2624490$ (by reducing the discretisation step sizes for our factored variables), but kept the bounded regions for state transitions quite small ($boundary_{bat}$=1kWh and $boundary_{tf}$=0.1€). SPUDD, once more, was not able to produce a solution within 24 hours, and could not generate a final policy with the available memory,in contrast to our value iteration method (Table 5.1).

Finally, we also experimented with a scenario involving 48 (half-hour) time steps at which the prosumer is required to act (as is usually the case in electricity markets). In this case, we had $|S \times A| = 664290$, $boundary_{bat}$=1kWh , and $boundary_{tf}$=0.1€. Once again, value iteration provided us with the same (optimal) policy as SPUDD, but in approximately 25% of the time (Table 5.1).

The experiments above demonstrate the limitations of SPUDD when used for problems that do not possess enough structure to allow for a compact enough representation of the required transitions in its input files. Both SPUDD and value iteration provide us with the same optimal policies in all experiments–that is, policies which intuitively maximize profits from selling/buying decisions while ensuring that consumer needs are satisfied. Nevertheless, value iteration required a fraction of SPUDD's total required time to produce the solution. We note that this is despite the fact that we took special care to make our factored representation as compact as possible for SPUDD to operate upon.

We also report that the average actual reward when running the exact value iteration (EVI) method of [2] is *1850€* for the entire finite state space. While, the average actual reward when running the exact value iteration (EVI) method with random policy is *1150€*. This experiment demonstrates a loss of *700€* for the prosumer by not following the optimal policy extracted from the exact value iteration.

5.2 Evaluation of the Fitted Value Iteration Method

In order to learn the approximate value function for this problem and generalise to the continuous state spaces, we use (progressively increasing) fractions of the aforementioned finite state space as the m samples required by the *FVI* method of Algorithm 6 for learning the $V_h(s)$. Specifically, we learned 11 different approximate value functions, using sample sizes of 5% and $\{10\%, 20\%, 30\% \dots 100\%\}$ of the finite state space, and then we evaluated the performance of their corresponding resulting policies, by observing the actual rewards they accumulate, and by calculating their *root mean squared error (RMSE)* [35] with respect to the rewards accumulated by the exact value iteration algorithm of [2]. In order to assess the effect of different basis functions on approximation quality, we tested 9 different basis functions at each one of our 11 approximation settings. These functions are the well-known *sigmoid*, *gaussian*, *inverse quadratic*, *thin plate spline*, and five *polynomials* of 1^{st} to 5^{th} degree.

All the *FVI* variants thus obtained, compute value functions that constitute *generalised* solutions; these can then be used to provide the optimal (greedy wrt. the value function) policies, given any particular state space discretisation. Here we assume a discretisation that is as the one presented above, and evaluate the performance of each *FVI* variant as follows. Once the approximate value functions and their corresponding approximate optimal policies are calculated, we execute the policies $1,000$ times each—and compute their accumulated rewards over a complete horizon, and its average value over the $1,000$ runs. We can thus assess the various variants in terms of average performance *wrt.* accumulated rewards. We present our findings in Table 5.2 for discount factor γ=0.9, and in Table 5.3 for discount factor γ=1. Moreover, we calculate the *RMSE* of the rewards derived from policies corresponding to the approximate and non–approximate value function. The *RMSE* values are presented in Table 5.4 for discount factor γ=0.9 and in Table 5.5 for discount factor γ=1.

We see in those tables that, with the exception of the *inverse quadratic* variants, all methods exhibit good performance, which is also quite stable across most sample sizes used for learning. The *gaussian* and the *polynomial* variants, in particular, are doing very well, often exhibiting performance that reaches or exceeds 90% of that of our exact value iteration (EVI for short) algorithm, when it operates in the $|S \times A| = 664290$ environment. Moreover, they appear to be able to do quite well even with small sample sizes. By contrast, the sigmoid method does exhibit stable performance, regularly at 80% of that of EVI, but does not do very well for small sample sizes. The *thin plate spline* variant also reaches an average performance of 83%, but does not do as well as the *gaussian* or the *polynomial* variants. The fact that most variants, and the polynomial variants in particular, are good approximations of the exact value function is further exhibited in Figures 5.4 to 5.12, where blue line presents the approximate value function (estimated with a discount factor γ=1), while the red line presents the EVI value

		Percentage of Sampling											
		5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
	sigmoid	1012	1482	1557	1634	1578	1499	1491	1507	1575	1517	1551	1491
	gaussian	**1685**	1737	1618	1625	**1668**	1633	1562	1583	1648	1748	1600	1646
	inverse quadratic	581	585	580	576	578	580	585	586	585	586	576	581
	thins plate spline	1551	1487	1318	1375	1658	1658	1589	1596	1590	1577	**1669**	1551
Function	1st polynomial	**1690**	**1714**	1529	1564	1568	1633	1678	1615	1632	1617	1625	1624
	2nd polynomial	1617	**1808**	1647	1697	1598	1632	1584	1622	1640	1608	**1679**	1648
	3rd polynomial	**1756**	**1697**	1662	1650	1687	**1669**	**1692**	**1703**	1609	**1800**	**1679**	**1691**
	4th polynomial	**1695**	1621	**1685**	1651	1532	1592	1648	1594	**1718**	**1711**	**1725**	1652
	5th polynomial	**1750**	1586	1661	1526	1578	**1745**	1607	**1692**	1646	1587	1568	1631

Table 5.2: Accumulated reward when using different basis functions and different sample sizes of the finite state space for learning the approximate value function. All numbers in the 5% to 100% columns are averages over 1000 runs and discount factor γ=0.9. We also report that the average actual reward when running the exact value iteration (EVI) method of [2] is *1850* for the entire finite state space. Values shown in **bold** are those that are over $1665 = 90\% \cdot 1850$€.

		Percentage of Sampling											
		5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
	sigmoid	1321	1641	1641	1641	1641	1641	1641	1641	1641	1641	1641	1613
	gaussian	1483	**1778**	**1805**	**1807**	**1816**	**1817**	**1818**	**1817**	**1818**	**1818**	**1818**	1781
	inverse quadratic	**1839**	**1822**	**1837**	**1839**	**1839**	**1837**	**1837**	**1837**	**1837**	**1837**	**1837**	1837
	thins plate spline	1469	636	1205	1205	1206	1207	1206	1207	1206	1206	1207	1179
Functions	1st polynomial	**1821**	**1824**	**1824**	**1824**	**1824**	**1825**	**1825**	**1826**	**1826**	**1826**	**1827**	1825
	2nd polynomial	**1824**	**1825**	**1824**	**1826**	**1823**	**1826**	**1826**	**1826**	**1826**	**1826**	**1829**	1826
	3rd polynomial	1350	1241	1141	1346	1347	1346	1346	1346	1347	1347	1268	1312
	4th polynomial	1350	1241	1143	1347	1346	1347	1347	1347	1347	1347	1268	1312
	5th polynomial	**1821**	**1824**	**1824**	**1826**	**1823**	**1826**	**1827**	**1827**	**1827**	**1826**	**1830**	1826

Table 5.3: Accumulated reward when using different basis functions and different sample sizes of the finite state space for learning the approximate value function. All numbers in the 5% to 100% columns are averages over 1000 runs and discount factor γ=1. We also report that the average actual reward when running the exact value iteration (EVI) method of [2] is *1850* for the entire finite state space. Values shown in **bold** are those that are over $1775 = 95\% \cdot 1850$€.

		Percentage of Sampling											
		5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
	sigmoid	958	471	491	540	496	532	518	526	477	488	522	547.6
	gaussian	640	639	624	636	622	668	678	633	614	604	685	640.8
	inverse quadratic	1264	1262	1267	1277	1272	1267	1266	1262	1264	1266	1277	1268.1
	thins plate spline	612	715	576	516	628	602	658	663	594	662	503	612.2
Function	1^{st} polynomial	596	615	685	652	672	629	640	648	667	614	647	642.7
	2^{nd} polynomial	452	559	611	595	677	644	623	641	622	638	567	603.1
	3^{rd} polynomial	335	555	642	629	641	601	625	637	642	576	611	591
	4^{th} polynomial	383	678	638	617	685	650	631	671	588	632	578	614.1
	5^{th} polynomial	484	690	635	657	674	596	644	634	619	644	673	632.4

Table 5.4: RMSE with respect to the EVI [2] policy reward for discount factor γ=0.9.

		Percentage of Sampling											
		5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
Functions	sigmoid	1050	458	340	407	236	275	275	275	275	275	236	373
	gaussian	1371	648	109	68	132	135	135	135	135	135	189	291
	inverse quadratic	1488	1487	1488	1488	1488	1488	1488	1488	1488	1488	1488	1488
	thins plate spline	$3.9\ 10^5$	$7.2\ 10^6$	$2.8\ 10^6$	$9.8\ 10^5$	$1.9\ 10^5$	$1.2\ 10^5$	$1.2\ 10^5$	$1.2\ 10^5$	$1.2\ 10^5$	$1.2\ 10^5$	$6.1\ 10^4$	$1.1\ 10^6$
	1st polynomial	24	26	24	143	229	206	206	206	206	206	226	155
	2nd polynomial	18	23	23	144	230	202	202	202	202	202	227	153
	3rd polynomial	2341	7829	7585	4694	4249	4254	4254	4254	4254	4254	4358	4757
	4th polynomial	2341	7829	7585	4694	4249	4254	4254	4254	4254	4254	4358	4757
	5th polynomial	27	26	24	144	230	203	203	203	203	203	227	154

Table 5.5: RMSE with respect to the EVI [2] policy reward for discount factor γ=1.

function.[1] We observe there that the graphs of their approximate value functions in general follow closely those of EVI for a large part of the state space, even though the expected values calculated do not match those calculated by EVI. Indeed, what is important for a good approximation is that the graph slope and the *relative ranking* of the state values are as those in the EVI value function graph, while the actual values do not matter. The graphs for the value functions of the polynomial variants, and, to some extent, of the *gaussian*, exhibit this behaviour. By contrast, the graph of the *thin plate spline* and the *inverse quadratic* variants depart quite a bit from that of EVI, which is consistent with the fact that their performance wrt. *RMSE* and accumulated rewards is not as satisfactory as that of the rest of our methods. In conclusion, the variants that exhibit the strongest and more stable performance are those employing a *gaussian* or a *polynomial* basis function—with the *polynomial* variants and, in particular, the *3rd degree polynomial* variant, doing equally well, regularly reaching a performance that is at about 90% of that achieved by EVI, or even more than 95% in the case of γ=1.

[1]States on the x axis in these figures are ranked in reverse order wrt. steps-to-go in the horizon: states with small indices occur early in the day-ahead, and the ones to the right late.

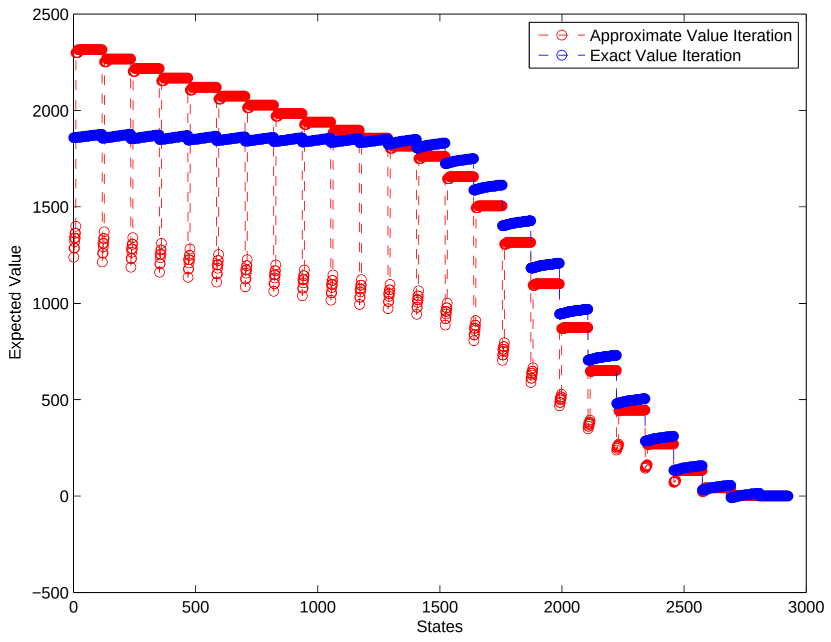


Figure 5.4: Approximate Value Function with a Sigmoid Basis Function.

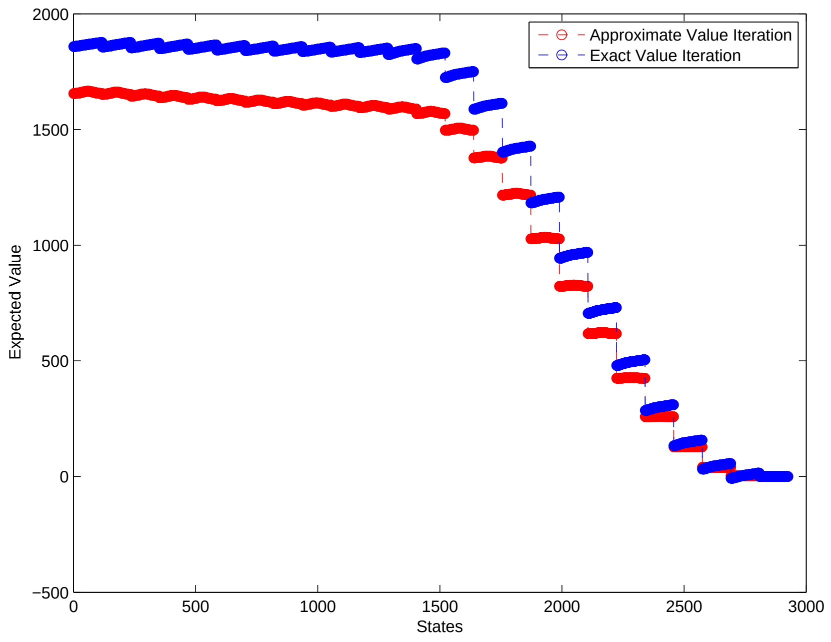


Figure 5.5: Approximate Value Function with a Gaussian Basis Function.

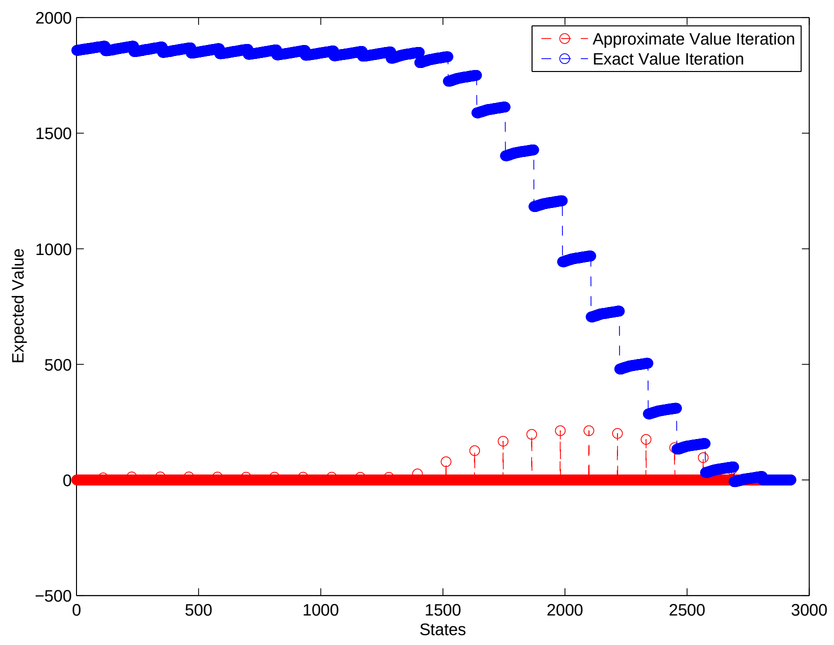


Figure 5.6: Approx. Value Function with an $Inverse\ Quadratic$ Basis Function.

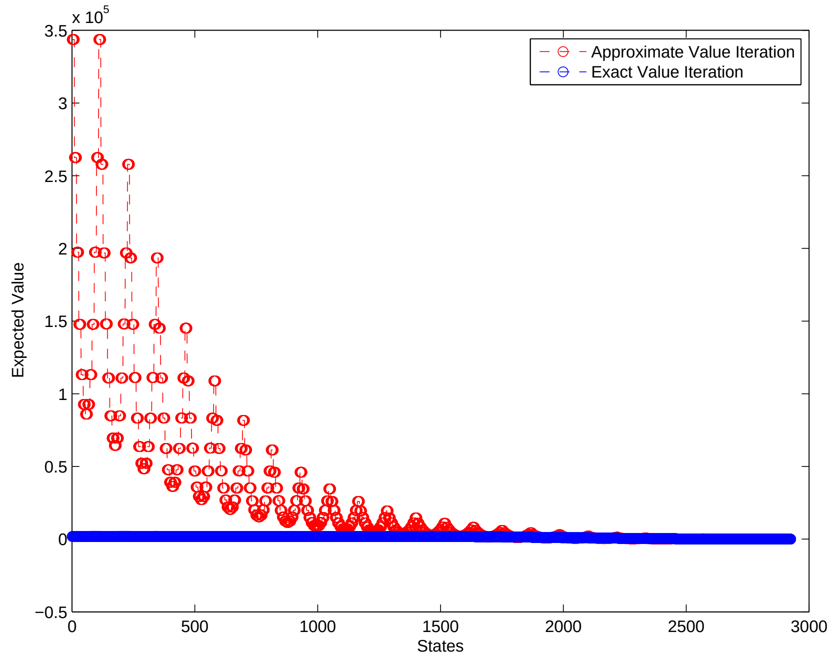


Figure 5.7: Approx. Value Function with a $Thin\ Plate\ Spline$ Basis Function.

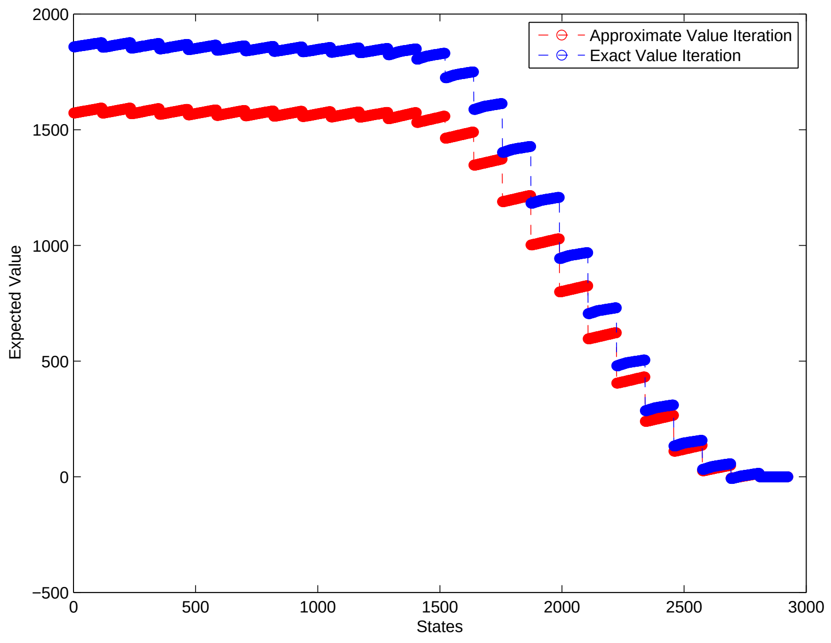


Figure 5.8: Approximate Value Function with a $1^{st}degree\ poly$ Basis Function.

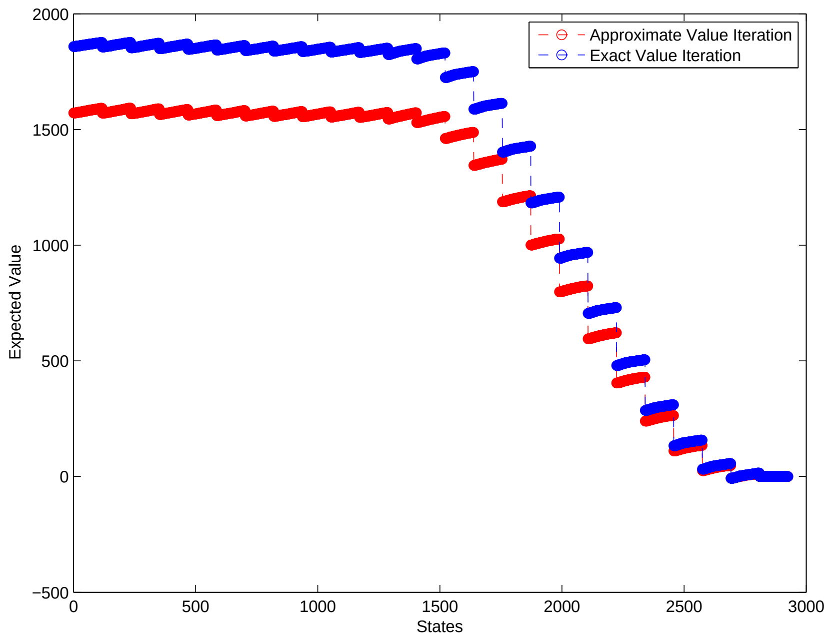


Figure 5.9: Approximate Value Function with a $2^{nd}degree\ poly$ Basis Function.

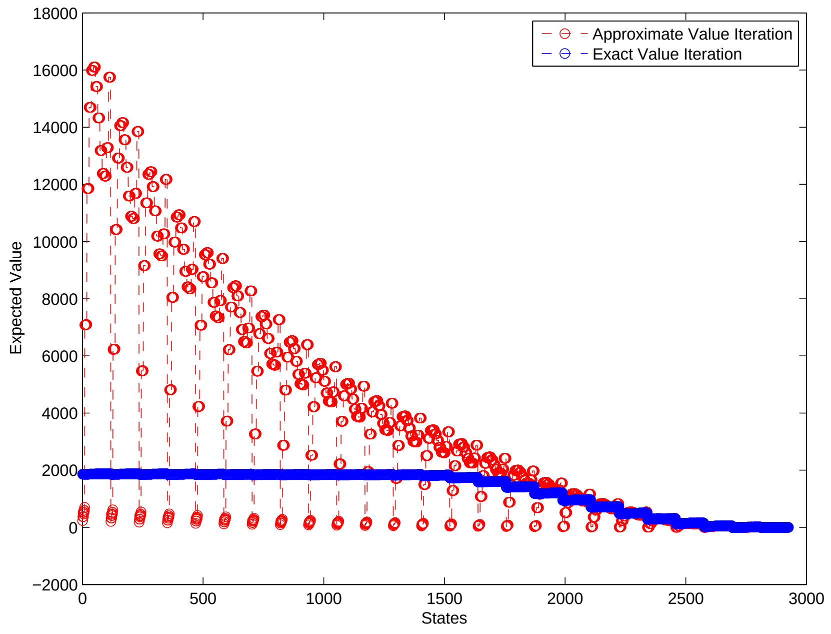


Figure 5.10: Approximate Value Function with a $3^{rd} degree\ poly$ Basis Function.

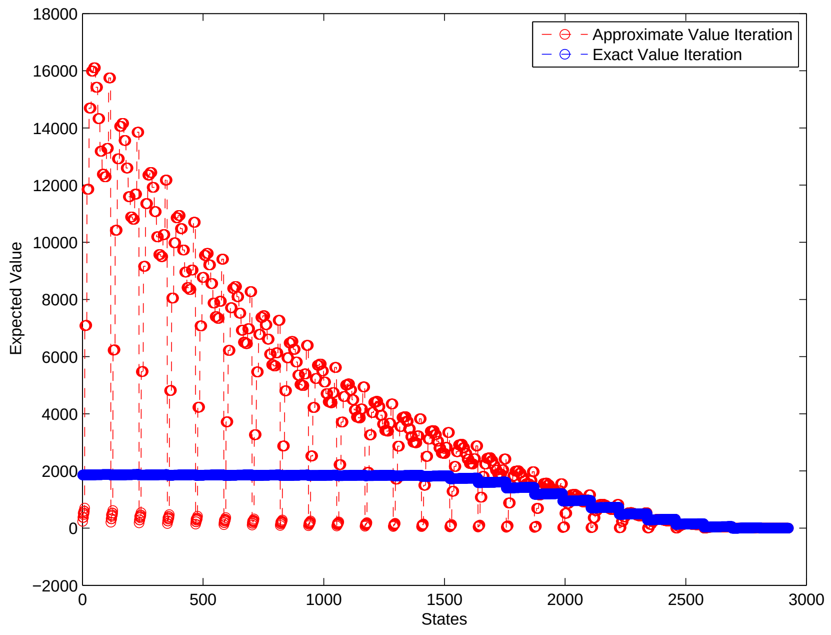


Figure 5.11: Approximate Value Function with a $4^{th} degree\ poly$ Basis Function.

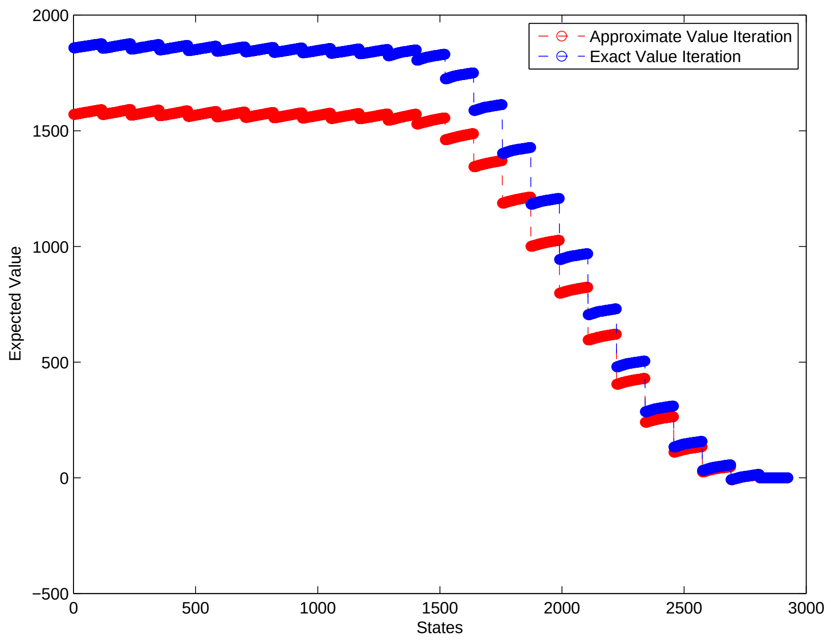


Figure 5.12: Approximate Value Function with a $5^{th} degree\ poly$ Basis Function.

Chapter 6

Conclusions

This thesis employs, for the first time, factored MDPs to model the decision problem faced by a prosumer planning its energy flow management for the day-ahead. Our model incorporates the key factors responsible for the effective operation of a microgrid prosumer, regardless of its size; and allows us to obtain the exact optimal solution to the problem. We used a simple value iteration algorithm to compute the solution to this sequential decision making problem, and demonstrated our method's effectiveness and efficiency by comparing it to the performance of *SPUDD*. By so doing, we exposed the limitations of this particular FMDP solver. While our model enables the simple VI method to compute the optimal solution within a reasonable time, the problem does not have enough structure to allow the creation of a compact input file for SPUDD to operate on, resulting to poor performance.

We also show how to use approximate MDP solution methods for taking decisions in this domain *without* the need of discretizing the state space. More specifically, we solve the prosumer decision problem using FVI in order to generalize our factored MDP solution method to continuous state spaces. Our method is shown to exhibit stable performance in terms of accumulated reward, which for certain basis functions reaches 90% of that gathered by the exact algorithm. Furthermore, a distinct contribution of our work is the evaluation of two regression methods (GPs and Bayesian Linear Regression) that can be used for obtaining the prosumer load consumption estimates.

Our model and solution technique allow the determination of optimal policies regarding the main prosumer activities. However, additional state and action variables can be added to the model, to allow for additional operations to take place (e.g., choosing to alter the projected production and consumption levels for increased economic benefits). Moreover, one can use these ideas in order to smoothly incorporate prosumers within *cooperatives* that are fast emerging in the Smart Grid [26, 24]. Finally, it would be interesting to analyse the performance if we have two or more microgrids using this kind of decision making, and to take the interaction between microgrids into account.

Bibliography

[1] A. Angelidakis and G. Chalkiadakis. Factored MDPs for Optimal Prosumer Decision-Making in Continuous State Spaces. In *Proc. of EUMAS-2015*, 2015.

[2] A. Angelidakis and G. Chalkiadakis. Factored MDPs for Optimal Prosumer Decision-Making. In *Proc. of AAMAS-2015*, 503–511, 2015.

[3] A. Panagopoulos, G. Chalkiadakis, and E. Koutroulis. Predicting the power output of distributed renewable energy resources within a broad geographical region. In *Proc. of ECAI /PAIS 2012*, pages 981–986, 2012.

[4] A. Panagopoulos, G. Chalkiadakis, and N. Jennings. Towards optimal solar tracking: a dynamic programming approach. In *Proc. of AAAI-2015*, 2015.

[5] A. Rogers, S. Maleki, S. Ghosh, and N. Jennings. Adaptive home heating control through gaussian process prediction and mathematical programming. In *Proc. of ATES 2011*, pages 71–78, 2011.

[6] A. Rogers, S. Ramchurn, and N. Jennings. Delivering the Smart Grid: Challenges for autonomous agents and multi-agent systems research. In *Proc. of AAAI-2012*, pages 2166–2172, 2012.

[7] B. Zhao, X. Zhang, J. Chen, C. Wang, and L. Guo. Operation optimization of standalone microgrids considering lifetime characteristics of battery energy storage system. *Sustainable Energy, IEEE Transactions on*, pages 934–943, 2013.

[8] C. Akasiadis and G. Chalkiadakis. Agent Cooperatives for Effective Power Consumption Shifting. In *Proc. of AAAI-2013*, 2013.

[9] C. Akasiadis and G. Chalkiadakis. Stochastic filtering methods for predicting agent performance in the Smart Grid. In *Proc. of ECAI/PAIS-2014*, pages 1205–1206, 2014.

[10] C. Boutilier Correlated action effects in decision theoretic regression. Proc. of UAI-1997, pages 30–37, 1997.

[11] C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. *Intl. Conf. on Machine Learning*, pages 54–62, 1996.

[12] C. Boutilier, R. Dearden and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Journal of Artificial Intelligence Research (JAIR)*, pages 49–107, 2000.

[13] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)*, pages 1–94, 1999.

[14] C. Guestrin, D. Koller and R. Parr. Max-norm projections for factored MDPs. In *Proc. of IJCAI-2001*, pages 673–682, 2001.

[15] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, pages 399–468, 2003.

[16] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] D. Ernst and P. Geurts and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, pages 503–556, 2005.

[18] D. Kirschen and G. Strbac. *Fundamentals of Power System Economics*. J. Wiley & Sons, 2005.

[19] D. Nikovski and W. Zhang. Factored markov decision process models for stochastic unit commitment. In *IEEE Conference on Innovative Technologies for an Efficient and Reliable Electricity Supply (CITRES)*, pages 28–35, 2010.

[20] D. Urieli and P. Stone. Tactex'13: a champion adaptive power trading agent. In *Proc. of AAMAS-2014*, pages 1447–1448, 2014.

[21] D P. Bertsekas and J N. Tsitsiklis. Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3). In *Proc. of Journal of Athena Scientific*, 15–23, 1996.

[22] D P. De Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate dynamic. In *Proc. of Journal of Mathematics of Operations Research*, 462–478, 2004.

[23] E. Stiefel Note on Jordan elimination, linear programming and Tchebycheff approximation. *Journal of Numerische Mathematik*, pages 1–17, 1960.

[24] Federation of groups and cooperatives of citizens for renewable energy in Europe. http://www.rescoop.eu.

[25] F. V. Jensen. An introduction to Bayesian networks. UCL press London, 1996.

[26] G. Chalkiadakis, V. Robu, R. Kota, A. Rogers, and N. Jennings. Cooperatives of distributed energy resources for efficient virtual power plants. In *Proc. of AAMAS-2011 - Volume 2*, pages 787–794, 2011.

[27] G. J. Gordon. Stable function approximation in dynamic programming. In *Proc. of the 12th International Conference on Machine Learning*, 261–268, 1995.

[28] H. Kanchev, D. Lu, F. Colas, V. Lazarov, and B. Francois. Energy management and operational planning of a microgrid with a PV-based active generator for Smart Grid applications. *Industrial Electronics, IEEE Transactions on*, pages 4583–4592, 2011.

[29] J. Chiasson and B. Vairamohan. Estimating the state of charge of a battery. *IEEE Transactions on Control Systems Technology*, pages 465–470, 2005.

[30] J. Hoey, R. St-Aubin, A. J. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. Proc. of UAI-1999, pages 279–288, 1999.

[31] J. Kolter and J. Ferreira. A large-scale study on predicting and contextualizing building energy usage. In *AAAI*, pages 1349–1356, 2011.

[32] J N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. Springer, 1996.

[33] J. Veerapen and M. Beerepoot. Co-generation and Renewables. OECD/IEA, Paris, 2011.

[34] L. Busoniu and R. Babuska and B. De Schutter and D. Ernst. Reinforcement learning and dynamic programming using function approximators. CRC press, 2010.

[35] M. DeGroot and J. Schervish (2002). *Probability and Statistics* Addison-Wesley, 2002.

[36] M. Peters, W. Ketter, M. Saar-Tsechansky, and J. Collins. A reinforcement learning approach to autonomous decision-making in smart electricity markets. *Machine learning*, 92(1), pages 5–39, 2013.

[37] N. Nisan and A. Ronen Algorithmic mechanism design. In *Proc. of the annual ACM symposium on Theory of computing*, 129–140, 1999

[38] P. Asmus. Microgrids, virtual power plants and our distributed energy future. *The Electricity Journal*, pages 72–82, 2010.

[39] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, pages 171–206, 1997.

[40] R. Bellman, R. Kalaba and B. Kotkin. Polynomial approximation–A new computational technique in dynamic programming: Allocation processes. *International Journal of Mathematics and Computation*, pages 155–161, 1963.

[41] R. Bellman. A Markovian decision process. In *Proc. of Journal of Mathematics and Mechanics*, 1957.

[42] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Journal of Artificial Intelligence Research (JAIR)*, pages 41–85, 1999.

[43] R. E. Fikes, N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Journal of Artificial Intelligence Research (JAIR)*, pages 189–208, 1972.

[44] R. E. Sims and G. K. Hans-Holger. Carbon emission and mitigation cost comparisons between fossil fuel, nuclear and renewable energy resources for electricity generation. In *Proc. of Journal of Energy Policy*, 1315–1326, 2003.

[45] R. J. Williams and L. C. Baird. *Tight performance bounds on greedy policies based on imperfect value functions*. Discussion Paper, 1993.

[46] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of IJCAI-1995*, pages 1137–1143, 1995.

[47] R. Kota, G. Chalkiadakis, V. Robu, V. Rogers and N. Jennings. Cooperatives for demand side management. In *Proc. of ECAI /PAIS 2012*, 2012.

[48] R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration In *Proc. of Journal of Machine Learning Research*, 815–857, 2008.

[49] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction, 1998. MIT press Cambridge, 1998.

[50] R. Sutton. Learning to predict by the methods of temporal differences. Springer, 1988.

[51] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 509–516, 1978.

[52] S. Boyd and L. Vandenberghe. *Convex Optimization*. 2004.

[53] S. D. Ramchurn, P. Vytelingum, A. Rogers, and N. Jennings. Putting the 'smarts' into the Smart Grid: A grand challenge for Artificial Intelligence. *Commun. ACM*, 55(4), pages 86-97, 2012.

[54] T. Ackermann (ed.). *Wind power in power systems*. J. Wiley & Sons, 2005.

[55] T. Dean and K. Kanazawa. *A model for reasoning about persistence and causation*. Brown University, Department of Computer Science, 1989.

[56] W. Kempton and J. Tomić. Vehicle-to-grid power implementation: From stabilizing the grid to supporting large-scale renewable energy. In *Proc. of Journal of Power Sources*, 280-294, 2005.