Factored MDPs for Optimal Prosumer Decision-Making in Continuous State Spaces

Angelos Angelidakis and Georgios Chalkiadakis

School of Electronic and Computer Engineering, Technical University of Crete, Greece

Abstract. The economic profitability of Smart Grid *prosumers* (i.e., producers that are simultaneously consumers) depends on their tackling of the decision-making problem they face when selling and buying energy. In previous work, we had modelled this problem compactly as a factored Markov Decision Process, capturing the main aspects of the business decisions of a prosumer corresponding to a community microgrid of any size. Though that work had employed an exact value iteration algorithm to obtain a near-optimal solution over discrete state spaces, it could not tackle problems defined over continuous state spaces. By contrast, in this paper 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. 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.

Keywords: energy; smart grid; factored MDPs; decision-making; approximation methods; continuous state spaces;

1 Introduction

In the emerging Smart Grid, prosumers are entities that both produce and consume energy, and which could be of utmost importance for the effectiveness and the stabilization of the electric networks [3,14,15]. A prosumer could correspond to a single residence, a specific industry, or to whole neighbourhoods of houses, that may or may not be connected to the rest of the electricity Grid. Already today, in Europe alone, there exist dozens of *energy cooperatives* encompassing hundreds of thousands of electricity prosumers [18]. The prosumer we consider in this work corresponds to a *microgrid* distributing power to a community. As such, he produces energy by means of *renewable energy sources*, and is responsible for the needs of *residential consumers*. Moreover, the prosumer has access to *storage devices (batteries)*, which he can use to store energy for future use. Our prosumer is connected to the wider Grid, and he has to take decisions regarding the amounts of energy to purchase from or sell to utility companies. The economic viability of such an entity therefore relies entirely on its business

decisions regarding buying, selling, or storing energy; the prosumer aims to maximise profits, while covering the electricity needs of its consumers. Prosumers like the one we consider here are set to become commonplace in the near future [3].

There has been much recent interest in decision making regarding selling and buying electricity in the Smart Grid. There are many competitors in the well-known Power-TAC research competition, for instance, pitting autonomous brokers that compete with each other in order to maximize profits through energy trading.[1] Most of published work in the domain, however, does not deal with prosumers; and papers that do, do not focus on micro-grid consumers. As an example, Nikovski and Zhang [13], 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, they do not tackle the problem of selling or storing the power generated. Kanchev *et al.* [10], on the other hand, propose an energy management system for prosumers managing photovoltaic generators, storage units, and gas micro-turbines; but their system is deterministic, and does not consider any uncertainties or perceivable failures.

To the best of our knowledge, our recent work in [2] is the only one tackling the decision-making problem faced by a micro-grid prosumer, modelling him as a *factored Markov Decision Process (FMDP)* [4]. In this way, the problem can be represented compactly, and an exact solution can be provided via dynamic programming, notwithstanding its large size. Our FMDP representation enabled a simple *value iteration* [16] method to outperform a state-of-the-art method for stochastic planning in very large environments [2]. However, the state space in [2] was finite, and as a result its *generalisation* to large state spaces is problematic [16]. In our work here we remove this limitation, and adopt an alternative solution method that approximates the value function, and can thus be employed in continuous state spaces.

Specifically, in this paper we employ *fitted value iteration*, a sampling-based approximation method that is known to perform well in a large class of MDPs [8,12]. This enables us to provide a near-optimal solution to the problem faced by a prosumer corresponding to a microgrid of *any* size, when the problem is modelled as a continuous-state FMDP. Representing the approximate value function requires the use of certain parameters and basis functions [12,5]. We use some well known polynomial and non-polynomial basis-functions, and estimate the optimal approximate value function parameters via employing off-the-shelf optimization algorithms.

In a nutshell, our contribution in this paper lies in proposing and evaluating, for the first time, the use of an approximate factored MDP solution method to continuous state decision problems faced by Smart Grid prosumers. We test-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 rewards, reaching about 90% of the performance of the exact algorithm in some cases.

The rest of this paper is organized as follows.[2] Section 2 provides a brief background on factored MDPs and reviews related work; Section 3 then describes our model, while

[1] See http://www.powertac.org/node/11 for a list of related publications.

[2] This is a slightly extended version of the paper that appears in the proceedings of EUMAS-2015. In particular, the Appendix section does not appear in the EUMAS-2015 article.

our approximate value iteration algorithm is described in Section 4; Section 5 presents our evaluation; and, finally, Section 6 concludes this paper and outlines future work.

2 Background

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 value iteration with parametric approximation approaches in some detail. Specifically, in Section 2.1 we present model-based algorithms, and in Section 2.2 we describe offline and online model-free algorithms. Then, in Section 2.3, we present value iteration with non-parametric approximation.

2.1 Model-based value iteration with parametric approximation

This section considers Q-iteration with a parametric approximator. Q-iteration [16] is a model-based algorithm for approximate value iteration. Approximate Q-iteration [5] 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:

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

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 a appropriate approximation mapping F : $R^n \rightarrow Q$:

$$\hat{Q}_i = F(\theta_i) \tag{2}$$

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

$$\hat{Q}_{i+1} = (T \circ F)(\theta_i) \tag{3}$$

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:

$$p(Q) = \theta, where \tag{4}$$

$$\theta = argmin_\theta \sum (Q(x_i, u_i) - F(\theta)(x_i, u_i))^2 \tag{5}$$

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 1 illustrates approximate Q-iteration, and the relations between the mappings of T and P, and Q-functions. Then,

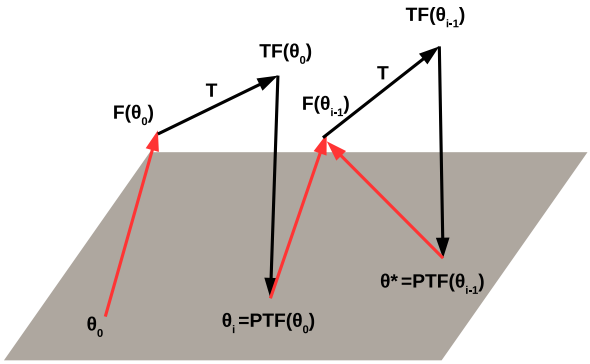

Fig. 1. An illustration of approximate Q-iteration. At each iteration, the approximation mapping F is applied to the current parameter vector to obtain an approximate Q-function, which is then passed through the Q-iteration mapping T. The result of T is then projected back onto the parameter space with 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 [5].

i $\leftarrow$ 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 $\leftarrow$ i+1
until θ *is satisfactory*;

Algorithm 1: Least-squares approximate Q-iteration for deterministic MDPs [5].

Algorithm 1 presents an example of approximate Q-iteration for a Markov decision process (MDP), using the least-squares projection. (We refer to [5] for more details.)

Another well known model-based value iteration algorithm is *fitted value iteration (FVI)* [8,12]. This algorithm is known to be well-behaving, 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 [12]. *FVI* was the algorithm of choice for us in this paper.

2.2 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:

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

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 *θ is satisfactory*;

Algorithm 2: Least-squares fitted Q-iteration with parametric approximation [5].

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

$$Q_{i+1}(x_i, u_i) = Q_i(x_i, u_i) + \alpha_i[r_{i+1} + \gamma max'_u Q_i(x_{i+1}, u') - Q_i(x_i, u_i)] \quad (6)$$

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 [5]. For simplicity, we denote the approximate Q-function at time i by:

$$\hat{Q}_i(x_i, u_i) = [F(\theta_i)](x_i, u_i) \quad (7)$$

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

$$\theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i}\Big[Q^*(x_i, u_i) - \hat{Q}(x_i, u_i)\Big]^2 \quad (8)$$

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:

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

The approximate Q-learning update then takes the form:

$$\theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i}\Big[r_{i+1} + \gamma max'_u \hat{Q}_i(xi + 1, u') - \hat{Q}(x_i, u_i)\Big]^2 \quad (9)$$

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

$$\theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i}\left[r_{i+1} + \gamma max'_u\left(\phi^T(xi+1,u')\theta_i\right) - \phi^T(xi,u_i)\theta_i\right]^2 \quad (10)$$

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}

$$\theta_{i+1} \leftarrow \theta_{i+1} = \theta_i - \frac{1}{2}\alpha_i \frac{\partial}{\partial \theta_i}\left[r_{i+1} + \gamma max'_u\left(\phi^T(xi+1,u')\theta_i\right) - \phi^T(xi,u_i)\theta_i\right]^2$$

end

Algorithm 3: Q-learning with a linear parametrization and ϵ-greedy exploration [5].

2.3 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 $\leftarrow$ 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 $\leftarrow$ i+1

until Q_{i+1} *is satisfactory*;

Algorithm 4: Fitted Q-iteration with non-parametric approximation [5].

3 A Factored MDP Model for Buying and Selling Energy

Factored Markov Decision Processes (FMDPs) [4] 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. 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) [9,4].

The FMDP model in our work here is as in [2], with the only difference that the state space is not finite. We provide a summarized description of this model here, and refer to [2] for more details. The model assumes that the Grid is represented by some utility company that can specify time–specific tariffs determining the sell and buy prices of electricity, to which the prosumer can subscribe to. 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 utilities. We note 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.

Factored states, actions, and signals The factored states are 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). The first one, *tms*, represents the time steps at which the prosumer takes decisions. Its domain is set to [1 ... 24], one for each hour in the day-ahead (for which we perform our actions scheduling). The second variable, *bat*, represents the amount of energy available in the batteries, and its domain is [0 ... $Battery_{max}$], with $Battery_{max}$ refers to the maximum capacity of the batteries. This state variable takes on continuous values. Finally, *tf* represents the tariff the prosumer subscribes to at each *tms* and its domain is the tariffs that the utility announces during the day. Its domain is DOM(*tf*)=$\{tf_1, \cdots, tf_i, \cdots, tf_K\}$, with K corresponds to the number of tariffs available on a specific day. Each tf_i tariff is described by a $buying_i$ and a $selling_i$ price.

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 a factored state to another, and takes a value in its domain DOM(a_i). The discretization for each DOM(a_i) 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 corresponds to the amount of energy bought or sold to the electric utility. $Load_{max}$ denotes the maximum total predicted residential consumption load of the prosumer, and RES_{nom} denotes the nominal power generating capacity of the renewable energy sources. The domain for *buy* is set to [-RES_{nom} ... $Load_{max}$]. Negative *buy* values signify the selling of energy. Second, factored action *chg*, represents the attempt to charge or discharge the batteries. Its value range is [-$Battery_{max}$ ... $Battery_{max}$]. Finally, the third action, sel_{tf}, corresponds to a selection of tariff by the prosumer. Its domain is [0 ... K]. The value 0 corresponds to the choice of the prosumer to remain to its current tariff, while values 1 to K corresponds to a choice to select one of the other K tariffs.

Now, there are three types of external signals received by the prosumer and can be described as multivariate random variable $\boldsymbol{sg} = \langle signal_i \rangle$. Each variable $signal_i$ can take a value in its domain DOM($signal_i$). *prod* and *cons* signify the prediction about the production and consumption of the prosumer at a specific time step *tms*. Their domains are DOM(*prod*)= [0 ... RES_{nom}] and DOM(*cons*)=[0 ... $Load_{max}$], and corresponds to the variables RES_{nom} and $Load_{max}$ respectively. The third signal, $price_{tf}$, represents, the buy and sell prices ($buying_i$ and $selling_i$) for each one of the K tariffs.

Finally, we note that there exist certain *physical constraints* that need to be respected at all times [2]. A battery cannot be charged over its capacity, and there should not be an attempt to discharge an energy quantity higher than that currently stored in the unit. Moreover, the battery storage level must always be kept between 20% and 80%. Finally, it is crucial that the *balance energy constraint* [11,1] 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):

$$prod_t - cons_t - chg_t + buy_t = 0 \tag{11}$$

Transition Model State transitions in our model are stochastic, since faults may occur while taking actions. The variable *tms* is an exception to this rule–since one specific time step is always followed by the next one. For the rest of the variables, we define certain *bounded regions* which include a subset of the state space 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. Actions are successful with some probability p while, with probability $1 - p$, they transition (uniformly at random) to some state within the bounded region.

Factored Reward Representation The reward function is associated with *(a)* either the gain from selling power to the utility or the cost of buying power in a certain price; *(b)* the running costs for being subscribed to a tariff; and *(c)* the operation costs of using the storage devices. As such, we choose to represent the reward function as a cost function with three main components. 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:

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

C_{energy}, captures the cost per Wh for buying electricity given the buy/sell rates prescribed by the tariff in effect:

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

C_{period} captures the periodic cost for being subscribed into a tariff:

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

where $C_1 = 0.013$, $C_2 = -2.7$ [2]. C_{bl}, captures the costs associated with *battery life losses*. That is, the costs of charging (or discharging) the storage devices (batteries). The C_{bl} cost of an attempted *chg* action can then be viewed as a fraction of the C_{init} *initial investment cost* for the batteries:

$$C_{bl} = L_{loss} \cdot C_{init}$$

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 (see [2,17] for details):

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

A_{total} is the total cumulative throughput (in Ah) during the battery's lifetime.

4 Solving the Factored Continuous-State MDP

With the above model at hand, 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)* [8,12], a sampling-based approximation method that is known to be well-behaving, as explained in Section 2.1 above. 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, exact value iteration, one needs to perform the following update:

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

The main idea of fitted value iteration is to approximately carry out this step, over a finite sample of states $s^{(1)}, \ldots, s^{(m)}$. 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 5.

```
Randomly sample m states s^(1),..., s^(m) ∈ 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) = (1/m) Σ ( R_h(s,a) + γ V_{h-1}(s') )
            end
            y_h(s) = min(q(a))      % min because reward corresponds to costs
        end
        θ ← argmin_θ (1/2) Σ (θ^T φ_h(s,a) − y_h(s))^2
        V_h(s) = θ^T φ_h(s,a)
    end
end
```

Algorithm 5: Fitted Value Iteration with finite horizon. Algorithm description based on the pseudo-code in Andrew Ng's lecture notes in `http://cs229.stanford.edu/notes/cs229-notes12.pdf`.

Fitted value iteration does not provably always converge. However, in practice, it often does converge (or at least approximately converge) [12]. 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.

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.

5 Evaluation

We evaluate our model by examining a residential prosumer at New Hampshire, New England, north-eastern United States. Our simulated prosumer serves 30 households and includes 20 photovoltaic modules with nominal power 60kW, 2 wind-turbines 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. All experiments were conducted on a 2.10 GHz x 4 Intel Core i3-2310M processor, with 8GB of memory.

We initially adopted the following discretisation for our state and action variables. 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:

$$bat = [0kWh \ : \ 1kWh \ : \ 60kWh]$$

$$chg = [-60kWh \ : \ 1kWh \ : \ 60kWh]$$

We also defined nine tariffs, which are as follows:

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

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

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

which thus give rise to 10 possible sel_{tf} tariff selection actions (nine corresponding to choosing one of the tariffs, plus one for staying with their current one). The transition boundaries for our state variables were set to bound_{bat}=1kWh and bound_{tf}=0.1€.

The discretisation above results to a state space size of $|S| = 13,176$, when the values of the state variable *tms* are also taken into account. However, *tms* can be incorporated into the problem's horizon, by setting the horizon to be equal to the number of time steps at which the prosumer is required to act; this effectively reduces the size of the finite state space. Without *tms*, the *state* space contains 549 discrete states, while the size of *state-action space* is $|S \times A| = 664290$. This is the finite space upon which the exact value iteration algorithm of [2] operates.

Now, 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 5 for learning the $V_h(s)$. Specifically, we learned 11 different approximate value functions, using sample sizes of 5% and $\{10\%, 20\%, 30\% \ldots 100\%\}$ 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)* [7] 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 1st to 5th 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. 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.

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 the exact value iteration (EVI for short) algorithm used in [2]. 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

		Percentage of Sampling											
		5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
Function	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
	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 1. 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. 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
Function	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
	1st polynomial	596	615	685	652	672	629	640	648	667	614	647	642.7
	2nd polynomial	452	559	611	595	677	644	623	641	622	638	567	603.1
	3rd polynomial	335	555	642	629	641	601	625	637	642	576	611	591
	4th polynomial	383	678	638	617	685	650	631	671	588	632	578	614.1
	5th polynomial	484	690	635	657	674	596	644	634	619	644	673	632.4

Table 2. RMSE with respect to the EVI [2] policy reward.

approximations of the exact value function is further exhibited in Figures 2 to 10, where blue line presents the approximate value function while the red line presents the EVI value function.[3] 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 slightly better, regularly reaching a performance that is at about 90% of that achieved by EVI.

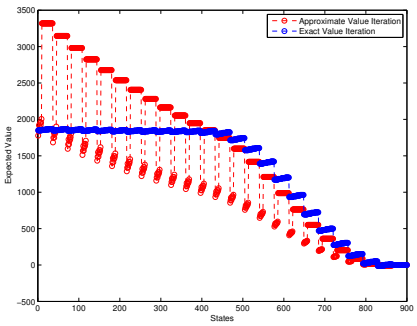

Fig. 2. Approximate Value Function with a Sigmoid Basis Function.

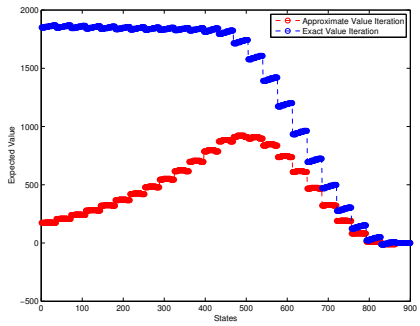

Fig. 3. Approximate Value Function with a Gaussian Basis Function.

6 Conclusions

This paper proposes and evaluates, for the first time, an approximate value iteration method for solving the decision problem facing Smart Grid prosumers when operating within an environment modelled as a continuous-state factored MDP. We provided a thorough evaluation of various functions that might form the basis of the approximate value function, and demonstrated that our approach performs very well for several such functions. As such, our model and solution technique allow the determination of approximately optimal policies regarding the main prosumer activities.

[3] 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.

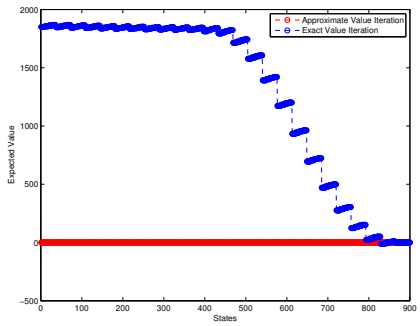

Fig. 4. Approx. Value Function with an $Inverse\ Quadratic$ Basis Function.

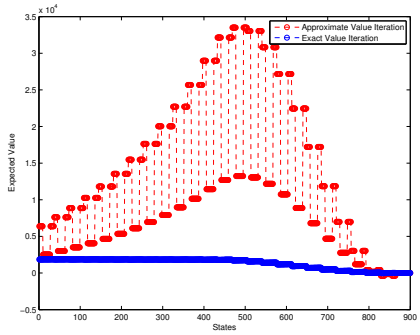

Fig. 5. Approx. Value Function with a $Thin\ Plate\ Spline$ Basis Function.

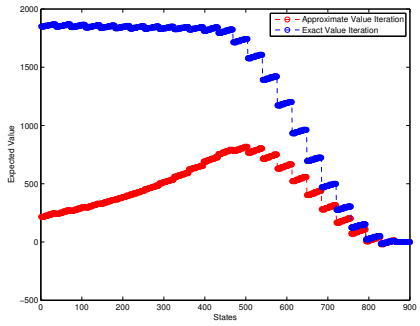

Fig. 6. Approximate Value Function with a $1^{st} degree\ poly$ Basis Function.

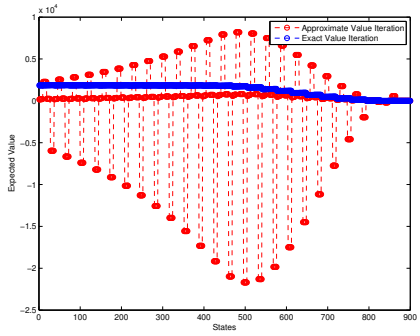

Fig. 7. Approximate Value Function with a $2^{nd} degree\ poly$ Basis Function.

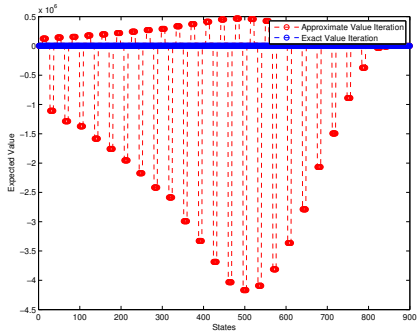

Fig. 8. Approximate Value Function with a $3^{rd} degree\ poly$ Basis Function.

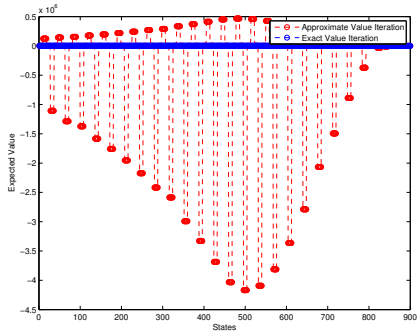

Fig. 9. Approximate Value Function with a $4^{th} degree\ poly$ Basis Function.

Future work includes incorporating more prosumer actions into our model, such as altering the projected production and consumption levels to increase the economic

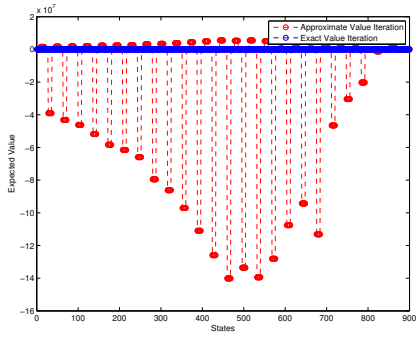

Fig. 10. Approximate Value Function with a $5^{th} degree\ poly$ Basis Function.

turnouts; and testing alternative MDP solution methods. Finally, we plan to incorporate our model within *renewable energy sources cooperatives*, the emergence of which is of extremely high economic, social, and environmental importance [6,18].

Acknowledgements The work presented in this paper was supported by the Greek General Secretariat for Research and Technology (GSRT) through the funding of research project "AFORMI – Reconfigurable Systems for scientific research" with proposal code 2427 within the context of action "ΑΡΙΣΤΕΙΑ" of the Lifelong Learning Program.

References

1. T. Ackermann (ed.). *Wind power in power systems*. Wiley, 2005.
2. A. Angelidakis and G. Chalkiadakis. Factored MDPs for Optimal Prosumer Decision-Making. In *Proc. of AAMAS-2015*, 503–511, 2015.
3. P. Asmus. Microgrids, virtual power plants and our distributed energy future. *The Electricity Journal*, 72–82, 2010.
4. C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)*, 1–94, 1999.
5. L. Busoniu and R. Babuska and B. De Schutter and D. Ernst Reinforcement learning and dynamic programming using function approximators. CRC press, 2010.
6. 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*, 787–794, 2011.
7. DeGroot, M. and J. Schervish (2002). *Probability and Statistics* Addison-Wesley, 2002.
8. G. J. Gordon Stable function approximation in dynamic programming. In *Proc. of the 12th International Conference on Machine Learning*, 261–268, 1995.
9. C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 399–468, 2003.
10. 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*, 4583–4592, 2011.
11. D. Kirschen and G. Strbac. *Fundamentals of Power System Economics*. Wiley, 2005.

12. R. Munos and C. Szepesvári Finite-time bounds for fitted value iteration In *Proc. of Journal of Machine Learning Research*, 815–857, 2008.
13. 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)*, 28–35, 2010.
14. 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), 86-97, 2012.
15. 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*, 2166–2172, 2012.
16. R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction, 1998.
17. 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*, 934–943, 2013.
18. Federation of European renewable energy cooperatives. `http://www.rescoop.eu`

7 Appendix

The results in the main body of the paper were obtained with a discount factor (used in Algorithm 5) of γ=0.9. We now present in Tables 3 and 4 further results obtained with a discount factor of γ=1. Moreover, Figures 11 to 19 show the approximation of the exact value function when γ=1.

We also report that the average actual reward when executing a random policy is *1150€*. By comparison, it gains *1850€* when executing the optimal policy.

		Percentage of Sampling											
		5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Average
Functions	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
	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 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 $1775 = \mathbf{95}\% \cdot 1850$€.

		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	$3.9\ 10^9$	$7.2\ 10^6$	$2.8\ 10^6$	$9.8\ 10^9$	$1.9\ 10^9$	$1.2\ 10^9$	$1.2\ 10^9$	$1.2\ 10^9$	$1.2\ 10^9$	$1.2\ 10^9$	$6.1\ 10^4$	$1.1\ 10^6$
	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 4. RMSE with respect to the EVI [2] policy reward for discount factor γ=1.

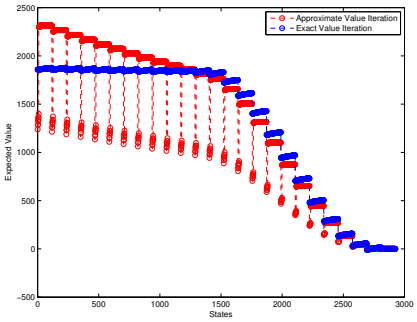

Fig. 11. Approximate Value Function with a Sigmoid Basis Function.

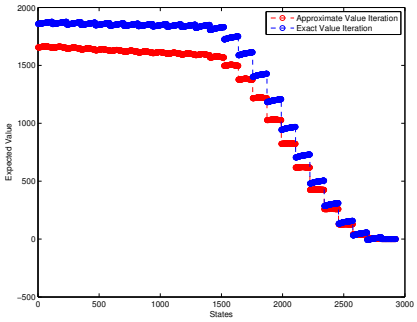

Fig. 12. Approximate Value Function with a Gaussian Basis Function.

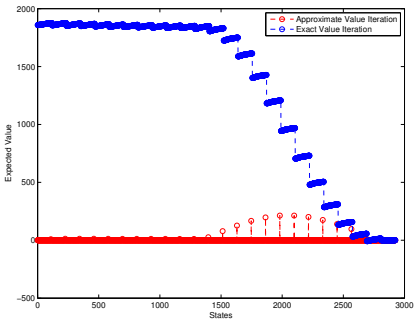

Fig. 13. Approx. Value Function with an *Inverse Quadratic* Basis Function.

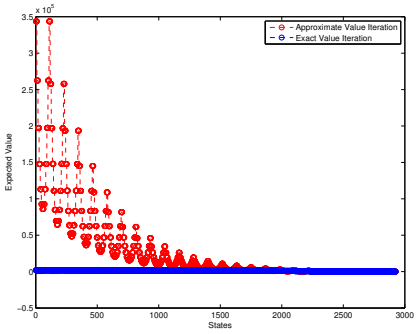

Fig. 14. Approx. Value Function with a *Thin Plate Spline* Basis Function.

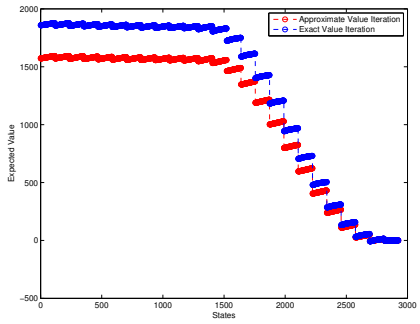

Fig. 15. Approximate Value Function with a $1^{st} degree\ poly$ Basis Function.

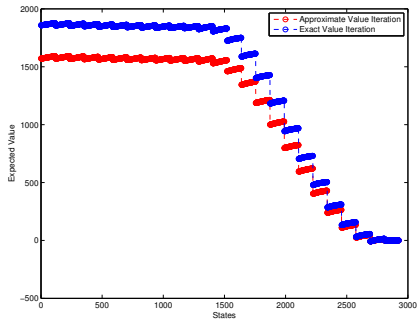

Fig. 16. Approximate Value Function with a $2^{nd} degree\ poly$ Basis Function.

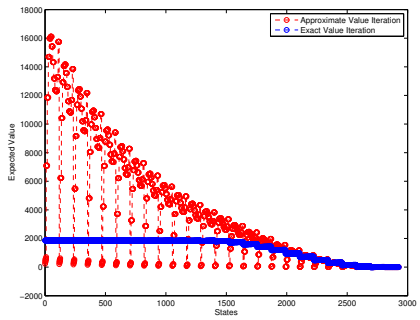

Fig. 17. Approximate Value Function with a $3^{rd} degree\ poly$ Basis Function.

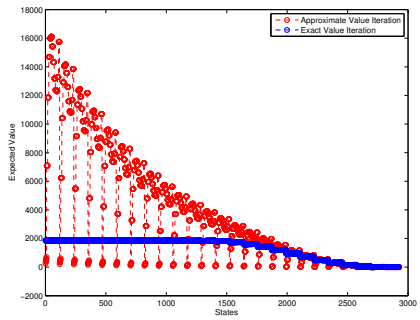

Fig. 18. Approximate Value Function with a $4^{th} degree\ poly$ Basis Function.

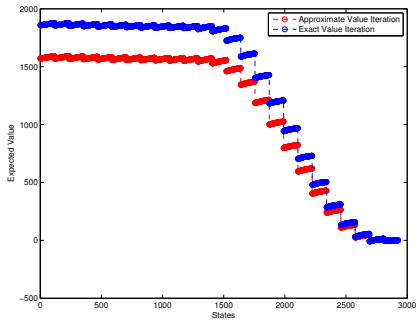

Fig. 19. Approximate Value Function with a $5^{th} degree\ poly$ Basis Function.