in: Proc. 2011 IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL 2011), pp. 84-90, IEEE 2011

Agent Self-Assessment: Determining Policy Quality Without Execution

Alexander Hans
Neuroinformatics and Cognitive Robotics Lab
Ilmenau University of Technology
Ilmenau, Germany
alexander.hans.ext@siemens.com

Siegmund Duell
Machine Learning Group
Berlin Institute of Technology
Berlin, Germany
duell.siegmund.ext@siemens.com

Steffen Udluft
Intelligent Systems and Control
Siemens AG, Corporate Technology
Munich, Germany
steffen.udluft@siemens.com

***Abstract*—With the development of data-efficient reinforcement learning (RL) methods, a promising data-driven solution for optimal control of complex technical systems has become available. For the application of RL to a technical system, it is usually required to evaluate a policy before actually applying it to ensure it operates the system safely and within required performance bounds. In benchmark applications one can use the system dynamics directly to measure the policy quality. In real applications, however, this might be too expensive or even impossible. Being unable to evaluate the policy without using the actual system hinders the application of RL to autonomous controllers. As a first step toward agent self-assessment, we deal with discrete MDPs in this paper. We propose to use the value function along with its uncertainty to assess a policy's quality and show that, when dealing with an MDP estimated from observations, the value function itself can be misleading. We address this problem by determining the value function's uncertainty through uncertainty propagation and evaluate the approach using a number of benchmark applications.**

***Index Terms*—reinforcement learning, robustness, autonomous agent, self-assessment, policy quality, Markov decision processes, uncertainty propagation**

I. Introduction

Reinforcement learning (RL) [1] is the machine learning answer to the optimal control problem. In the last decades RL algorithms have evolved to a viable alternative to classical control approaches. Since they are data-driven, using actual observations of the environment to be controlled, they are also applicable to problems where an analytical description of the system is unavailable or inaccurate. Moreover, they are able to adapt the solution to changing characteristics of the environment, which might be introduced by wear. Arguably the most important step on the way of making RL applicable to complex technical control tasks, e.g., combustion process tuning [2] and gas turbine control [3], was the development of data-efficient methods such as least-squares policy iteration [4], tree-based [5] and neural [6] fitted Q-iteration (FQI), neural rewards regression [7], or the recurrent control neural network [8]. Data-efficiency is crucial as observations are expensive because they are time-consuming to generate and arbitrary exploration is usually not permitted. Most methods attain data-efficiency through re-use of observation tuples [9], which is easiest to do in batch-mode, where not every observations is used immediately to update the policy currently executed, but the full batch of all observations is used offline to generate a new policy. For many technical control tasks it is adequate to use batch-mode RL—observations of the system from operation with previous controllers are often available and the policy is not updated continuously but only when a sufficient number of new observations leading to a substantially different policy are available.

However, data-efficiency is not sufficient. In order to implement RL for an autonomously working controller, more is needed. First, a suitable algorithm must be selected. Many algorithms are influenced by a number of parameters. So as a second step, the parameters have to be tuned for the problem at hand. Third, for algorithms employing approximators with non-convex error functions and using stochastic optimization, e.g., neural networks, it is often advisable to monitor the learning process, tune parameters, and possibly re-run the learning algorithm. When finally a policy is found, before actually applying it to the real system, it must be evaluated to ensure its quality. In benchmark applications, this evaluation can easily be done using the environment itself. For a real-world application, however, this is usually not possible, as the policy might turn out to be insufficient, leading to an undesirable decrease of performance or even damage the system while being evaluated. As an alternative, one could use a simulation, but therefore such a simulation must be available and model the real system accurately enough to allow conclusions about the policy's performance on the real system.

Instead of actually executing the policy to evaluate it, we are looking for methods to inspect a policy without requiring execution. The obvious indicator of policy performance is the value function. It should give the expected discounted future reward when following that policy. However, if the knowledge of the environment is limited, the estimates of the underlying Markov decision process's (MDP) parameters might lead to wrong conclusions and thus a flawed value function that does not reflect the true performance of the policy on the real MDP. We will illustrate that problem and propose the usage of the value function's uncertainty as a remedy. The uncertainty can be determined by uncertainty propagation, which in the context of RL has previously been used for quality assurance [10], [11] and exploration [12].

To the best of our knowledge, so far only few works related

978-1-4244-9888-8/11/$26.00 ©2011 IEEE

to the issue of self-assessment in RL exist. There are works concerned with the selection of a suitable policy. Gabel and Riedmiller [13] address the problem of policy degradation in NFQ by calculating a sample of the optimal Q-function tabularly and comparing that with the neural representation. They conclude that the closer the match, the better the policy. Migliavacca et al. [14] propose *fitted policy search*, a direct policy search method that uses an FQI-like approach to evaluate candidate policies. Instead of evaluating the policy on the real system or a simulation, they use *fitted policy evaluation* to determine the value function of the candidate policy using that policy and a set of observations of the system. Surprisingly, the resulting value function was sufficient to select good policies in this application. According to our experience with neural FQI, it is in general not possible to reliably reason about the quality of a policy using solely a single value function.

As a first step, this paper deals with discrete MDPs. We show that the naïve approach of comparing value functions can fail and how the incorporation of uncertainty can help to overcome this problem.

II. Preliminaries

In RL one is interested in finding a policy $\pi : S \mapsto A$ that moves an agent optimally in an environment assumed to be a Markov decision process (MDP) $M := (S, A, P, R)$ with a state space S, a set of possible actions A, the system dynamics, defined as probability distribution $P : S \times A \times S \mapsto [0, 1]$, which gives the probability of reaching state s' by executing action a in state s, and a reward function $R : S \times A \times S \mapsto \mathbb{R}$, which determines the reward expectation for a given transition.

Moving the agent optimally means maximizing the value function

$$V^\pi(s) = \mathbf{E}_{s'}\left[R(s,a,s') + \gamma V^\pi(s')\right] \quad (1)$$
$$= \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^\pi(s')\right], \quad (2)$$

where $\gamma \in [0, 1]$ is the discount factor. Often a so-called Q-function $Q^\pi(s, a)$ is utilized that gives the expected discounted reward when choosing action a in state s and afterward following policy π. The Q-function for the optimal policy $Q^{\pi^*} = Q^*$ is given by a solution of the Bellman optimality equation [15]

$$Q^*(s,a) = \mathbf{E}_{s'}\left[R(s,a,s') + \gamma V^*(s')\right] \quad (3)$$
$$= \mathbf{E}_{s'}\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right] \quad (4)$$
$$= \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]. \quad (5)$$

From Q^* the optimal policy follows as $\pi^*(s) = \arg\max_a Q^*(s, a)$, where π^* is a deterministic policy.

The Bellman equation can be used as an update equation that allows finding the value function for a specific policy (V^π) or the optimal policy (V^*) using dynamic programming [15]. If the parameters of the MDP are not known, they can be estimated from observations. This is the setting we deal with in this paper.

When we talk about the performance of a policy, we are referring to the mean reward the agent receives per step when executing the policy, i.e., when evaluating a policy for n steps and collecting immediate rewards $r_t, t = 1, 2, \ldots, n$, the mean reward of that evaluation is $\bar{r} = 1/n \sum_{t=1}^{n} r_t$.

III. Value Function Based Self-Assessment

If the true value function of a policy is available, the obvious solution for self-assessment is the usage of the value function as indicator of policy quality. The expected return of a policy π is then given as

$$J^\mu(\pi) = \sum_{s \in S} \mu_0(s) V^\pi(s), \quad (6)$$

where $\mu_0(s)$ is the probability of starting in s and V^π the value function of π. We found the mean value, i.e.,

$$J(\pi) = \bar{V}^\pi = \frac{1}{|S|} \sum_{s \in S} V^\pi(s), \quad (7)$$

to be a good alternative and use that for our experiments (Sec. V). Given a set of policies, with $J(\pi)$ it is possible to select the best m policies. Likewise, the user can specify a minimum required return $J_{\min}$. In an autonomous system a new policy π is then only applied if $J(\pi) \geq J_{\min}$.

Unfortunately, calculating the true value function requires exact knowledge of the MDP's state-transition probabilities and reward function. Usually those have to be estimated from observations. When dealing with stochastic MDPs, the estimates can be flawed. As an example, consider the simple two-state MDP illustrated in fig. 1. The optimal policy would in state 1 execute action 1 to remain in that state and receive a reward of 1, while in state 2 it would execute action 2 to go to state 1. If, however, only a limited set of observations of that MDP is available, e.g., generated by random exploration, chances are that in state 2 only the self-transition giving a reward of 2 was observed. When using the simple mean as reward estimator, this would lead to an estimate of $\hat{R}(2, 1, 2) = 2$. A policy determined using those estimators would always go to state 2 and execute action 1 there, assuming $\hat{V}^\pi(2) = 2/(1-\gamma)$, while the true value is $V^\pi(2) = -5/(1-\gamma)$. When comparing policies based on the value function, the bad policy would appear better than the optimal one. The problem here is that the estimators are used without considering their uncertainty. Having observed the transition leading to a reward of -10 not even once implies only few observations of that state-action pair in general, which in turn leads to a high uncertainty. In this example the incorrect reward estimate leads to a bad policy; incorrect estimates of transition probabilities can have similar effects. Knowing the uncertainty σV^π, which stems from both, transition probability and reward uncertainties, one can reformulate equation (6) to

$$J_u^\mu(\pi) = \sum_s \mu_0(s)\left[V^\pi(s) - \xi \sigma V^\pi(s)\right]. \quad (8)$$

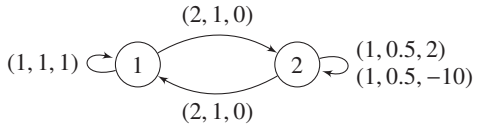

Fig. 1: Simple two-state MDP. In the description (a, b, c) of a transition a is the action, b the probability for that transition to occur, and c the reward.

ξ is a parameter weighting the uncertainty. Likewise, equation (7) becomes

$$J_u(\pi) = \frac{1}{|S|} \sum_s V^\pi(s) - \xi \sigma V^\pi(s). \tag{9}$$

We call the uncertainty incorporating value function $V_u^{\pi,\xi}(s) = V^\pi(s) - \xi \sigma V^\pi(s)$ *quantile value function*.

IV. Determining the Value Function's Uncertainty

To get the value function's uncertainty σV, we apply uncertainty propagation to the Bellman equation.

A. Uncertainty Propagation

Uncertainty propagation (UP), also known as Gaussian error propagation (see, e.g., [16]), is a common method in statistics to propagate the uncertainty of measurements to the results. It is based on a first-order Taylor expansion. Given a function $f(x)$ with $f : \mathbb{R}^M \mapsto \mathbb{R}^N$ and the uncertainty of the function arguments as covariance matrix $\mathrm{Cov}(x)$, the uncertainty of the function values $f(x)$ is determined as

$$\mathrm{Cov}(f) = \mathrm{Cov}(f, f) = D\mathrm{Cov}(x)D^\mathrm{T}. \tag{10}$$

D is the Jacobian matrix of f w.r.t. x consisting of the partial derivatives of f w.r.t. to each component of x, i.e., $D_{i,j} = \frac{\partial f_i}{\partial x_j}$.

When neglecting correlations of the arguments x as well as correlations of the components of $f(x)$, the argument's covariance matrix and the resulting covariance matrix $\mathrm{Cov}(f)$ are diagonal. In this case, a simplified expression for determining the uncertainty σf_i of values $f_i(x)$ can be used:

$$(\sigma f_i)^2 = \sum_j \left(D_{i,j}\right)^2 (\sigma x_j)^2 \tag{11}$$

$$= \sum_j \left(\frac{\partial f_i}{\partial x_j}\right)^2 (\sigma x_j)^2. \tag{12}$$

Here, $(\sigma f_i)^2$, $i = 1, 2, \ldots, N$ corresponds to the diagonal elements of $\mathrm{Cov}(f)$.

In the following, we will only use diagonal UP.

B. Applying UP

To determine the uncertainty of the value function, we apply uncertainty propagation to the Bellman equation

$$Q(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V(s')\right], \tag{13}$$

where $V(s) = Q(s, \pi(s))$ for policy evaluation, $V(s) = \max_a Q(s,a)$ for value iteration [1]. The Bellman equation naturally becomes an update equation when replacing the left hand Q with Q^{k+1} and the right hand V with V^k. Iterating this update equation, one arrives at the optimal Q-function Q^* (or Q^π for policy evaluation) for the given estimators P and R. The Bellman update equation can be regarded as a function mapping arguments P, R, and Q^k to Q^{k+1}. When applying diagonal UP (equation (12)), we get

$$\begin{aligned}(\sigma Q^m(s,a))^2 &:= \sum_{s'} (d_{QQ})^2 (\sigma V^{m-1}(s'))^2 + \\ &\quad \sum_{s'} (d_{QP})^2 (\sigma P(s'|s,a))^2 + \\ &\quad \sum_{s'} (d_{QR})^2 (\sigma R(s,a,s'))^2, \end{aligned} \tag{14}$$

$$\begin{aligned} d_{QQ} &= \gamma P(s'|s,a), \\ d_{QP} &= R(s,a,s') + \gamma V^{m-1}(s'), \\ d_{QR} &= P(s'|s,a), \end{aligned}$$

where σP and σR are the uncertainties of the respective estimator.

The resulting algorithm is called the *diagonal approximation of uncertainty incorporating policy iteration* (DUIPI) [11] and is summarized in Alg. 1.

Algorithm 1: Diagonal Approximation of Uncertainty Incorporating Policy Iteration

Input: P, σP, R, σR, γ
Result: Q, σQ
begin
  $k \leftarrow 0$
  $\forall (s,a) : Q^0(s,a) \leftarrow 0,\ \sigma Q^0(s,a) \leftarrow 0$
  while *desired precision not reached* **do**
    // update Q
    for $\forall (s,a) \in S \times A$ **do**
      $Q^{k+1}(s,a) \leftarrow$
      $\sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q^k(s',a')\right]$
    end
    // update σQ
    for *all* $(s,a) \in S \times A$ **do**
      $\sigma Q^{k+1}(s,a) \leftarrow \sum_{s'} \left[\gamma P(s'|s,a) \sigma V^k(s')\right]^2 +$
      $\left[R(s,a,s') + \gamma V^k(s') \sigma P^k(s'|s,a)\right]^2 +$
      $[P(s'|s,a) \sigma R(s,a,s')]^2$
    end
    $k \leftarrow k + 1$
  end
  return Q^k, $\sqrt{\sigma Q^k}$
end

C. Bayesian Estimators and Uncertainty

UP for the Bellman iteration can be combined with any suitable estimator. A popular choice is the frequentist estimator, using relative frequency for the transition probabilities and the sample mean for the reward expectation. In this paper, however, we choose a Bayesian approach for the transition

probabilities, because this way the uncertainties are properly accessible.

Assuming all transitions from different state-action pairs to be independent of each other and the rewards, we can model the transitions as multinomial distributions. We assume the Dirichlet distribution as a prior over the parameter space $P(s_k|s_i,a_j)$ for each $(s_i,a_j) \in S \times A$, with density

$$\Pr(P(s_1|s_i,a_j),\ldots,P(s_{|S|}|s_i,a_j))_{\alpha_{ij1},\ldots,\alpha_{ij|S|}} = \frac{\Gamma(\alpha_{ij})}{\prod_{k=1}^{|S|}\Gamma(\alpha_{ijk})}\prod_{k=1}^{|S|}P(s_k|s_i,a_j)^{\alpha_{ijk}-1}, \quad (15)$$

$\alpha_{ij} = \sum_{k=1}^{|S|}\alpha_{ijk}$, which is a conjugate prior with posterior parameters $\alpha^d_{ijk} = \alpha_{ijk} + n(s_i,a_j,s_k)$, $\alpha^d_{ij} = \sum_{k=1}^{|S|}\alpha^d_{ijk}$. $n(s_i,a_j,s_k)$ denotes the number of observed transitions from state s_i with action a_j to state s_k. Since we have no prior knowledge about the distribution of particular transitions, we set $\alpha_{ijk} = \alpha$ equally for all distributions. We choose the expectation of the posterior distribution as the estimator, i.e., $\hat{P}(s_k|s_i,a_j) = \alpha^d_{ijk}/\alpha^d_{ij}$. The uncertainty of $\hat{P}$ then is

$$(\sigma\hat{P}(s_k|s_i,s_j))^2 = \frac{\alpha^d_{ijk}(\alpha^d_{i,j}-\alpha^d_{ijk})}{(\alpha^d_{ij})^2(\alpha^d_{ij}+1)}. \quad (16)$$

As estimator for the reward expectation, the mean of all observed reward samples of a transition (s_i,a_j,s_k) is used. As uncertainties of that estimator, we set

$$(\sigma\hat{R}(s_i,a_j,s_k))^2 = \begin{cases} \frac{\mathrm{var}(\hat{R}(s_i,a_j,s_k))}{n(s_i,a_j,s_k)-1}, & \text{if } n(s_i,a_j,s_k) > 1 \\ R^2_{\max}, & \text{otherwise.} \end{cases} \quad (17)$$

$R_{\max}$ is the maximum reward.

Alg. 2 summarizes the estimation of the transition probabilities and the reward expectations together with their respective uncertainties. Plugging those into Alg. 1, we can determine the Q-function with its uncertainty, from which the value function follows trivially as $V^*(s) = \max_a Q^*(s,a)$ for policy iteration and $V^\pi(s) = Q^\pi(s,\pi(s))$ for policy evaluation of π.

Algorithm 2: Estimation of transition probabilities and rewards

Input: transition counts n, sum of rewards r, sum of squared rewards r^2, α
Result: $\hat{P}$, $\sigma\hat{P}$, $\hat{R}$, $\sigma\hat{R}$
begin
  for *all* $(s,a) \in S \times A$ **do**
    $n_{sa} \leftarrow \sum_{s'} n(s,a,s')$
    $\alpha_0 \leftarrow n_{sa} + |S|\alpha$
    for *all* $s' \in S$ **do**
      $\alpha_i \leftarrow \alpha + n(s,a,s')$
      $\hat{P}(s'|s,a) \leftarrow \alpha_i/\alpha_0$
      $\sigma\hat{P}(s'|s,a) \leftarrow (\alpha_i(\alpha_0-\alpha_i)/(\alpha_0^2(\alpha_0+1))$
      if $n_{sa} > 1$ *and* $n(s,a,s') > 0$ **then**
        $\hat{R}(s,a,s') \leftarrow r(s,a,s')/n(s,a,s')$
        $\sigma\hat{R}(s,a,s') \leftarrow \frac{(r^2(s,a,s')/n(s,a,s') - r(s,a,s')/n(s,a,s'))^2}{n_{sa}-1}$
      else
        $\hat{R}(s,a,s') \leftarrow 0$
        $\sigma\hat{R}(s,a,s') \leftarrow r^2_{max}$
      end
    end
  end
  return $\hat{P}$, $\sqrt{\sigma\hat{P}}$, $\hat{R}$, $\sqrt{\sigma\hat{R}}$
end

V. Experiments

To evaluate the possibilities of determining policy quality using either the standard value function or the quantile value function, we conducted a number of experiments using the archery [10], wet-chicken [17], and trap [18] benchmark domains.[1]

A. Setup

For each domain, we generated a number of policies and tried to select the best m policies without additional information, i.e., without running the policy on the real MDP or a simulation. Likewise, no additional observations were used. To assess the selection quality, we evaluated each policy on the real MDP and determined its performance (mean reward per step) as a measure of its true quality.

[1]Source code allowing to reproduce the experimental results is available at `http://ahans.de/publications/adprl2011`

In particular, for each experiment we did the following:

1) Generate N observations using random exploration, estimate the MDP from the observations, use dynamic programming (policy iteration) [1] to determine the optimal policy π_i for the estimated MDP, and finally evaluate the policy 100 times for 1,000 steps each to determine its performance r_i. This is repeated 25 times, resulting in policies $\pi_{1,2,\ldots,25}$, value functions and uncertainties $(V,\sigma V)_{1,2,\ldots,25}$, and true performances $r_{1,2,\ldots,25}$.
2) Use $J(\pi)$ and $J_u(\pi)$ to create ranking vectors g^J and g^{J_u} of the policies. E.g., g^J_1 gives the index of the best policy according to $J(\pi)$, g^J_{25} the worst.
3) From the true performances r_i and the rankings g^J and g^{J_u} create vectors l^J and l^{J_u} containing the mean performance of the best m policies, $m = 1,2,\ldots,25$, i.e., $l^J = \left(r_{g^J{}_1}, \frac{1}{2}\sum_{i=1}^{2} r_{g^J_i}, \ldots, \frac{1}{m}\sum_{i=1}^{m} r_{g^J_i}, \ldots, \frac{1}{25}\sum_{i=1}^{25} r_{g^J_i}\right)$. Obviously, $l^J_{25} = l^{J_u}_{25}$ gives the mean performance of all 25 policies.
4) Steps 1–3 are repeated 400 times, allowing to generate vectors $\bar{l}^J$ and $\bar{l}^{J_u}$ containing the mean of the individual l^J and l^{J_u} vectors and $\sigma\bar{l}^J$ and $\sigma\bar{l}^{J_u}$ containing the uncertainty of the mean (standard error).

Note that while we generated the policies and their value functions and according uncertainty in one step, one could as well use a given policy and set of observations to generate the value function and uncertainty (policy evaluation).

We set the discount factor $\gamma = 0.975$. For all experiments we used the Bayesian estimator with $\alpha = 0.01$ for the transition

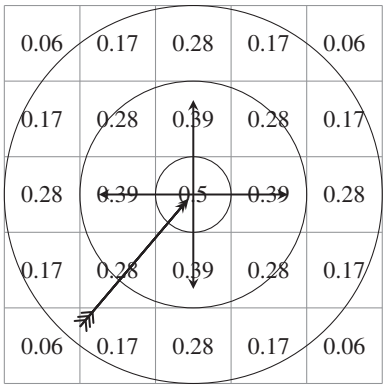

Fig. 2: Visualization of the archery benchmark showing the 25 states with their hitting probabilities.

probabilities and the sample mean to estimate the reward (Sec. IV-C). As weighting of the uncertainty we used a value of $\xi = 3$.

The figures in this section show the mean rewards given a number of selected best policies (vectors $\bar{l}^J$ and $\bar{l}^{J_u}$). E.g., the very left point gives the mean performance of the best selected policy, the very right point gives the mean performance over all policies, i.e., the performance expectation when selecting a policy randomly. For each experiment, the 10,000 policies were divided into 400 distinct sets of 25 policies each, a ranking was performed for each of those sets of 25 policies. Therefore, each point in a figure is the average of 400 values.

B. Archery

In the archery benchmark [10], the state space represents an archer's target (fig. 2). Starting in the target's middle, the archer has the possibility to move the arrowhead in all four directions and to shoot the arrow. The exploration was performed randomly with short episodes of 25 transitions. The arrowhead's moves are stochastic (probability 0.25 of moving in another direction) as well as the event of making a hit after shooting the arrow. The highest probability for a hit is with the arrowhead in the target's middle. Every exploration episode starts in the middle as well. The border is explored quite rarely, such that a hit there can misleadingly cause the respective estimator to indicate a high reward and thus the agent to finally shoot from this place.

We performed experiments according to the general setup using various numbers of random exploration observations. Results for a representative selection of numbers of observations are given in fig. 3.

When estimating an MDP from 300 observations, the standard value function does help in selecting a policy performing better than average. When selecting only the presumably best policy from a set of 25 policies, the mean performance of the policies applied to the real problem is 0.43, while random selection gives policies with a mean performance of 0.39. However, when considering the uncertainty for the selection as well, the performance of the policies selected as best increases to 0.48. When using 500 observations, the gain achievable with standard value function based policy selection further decreases, while the selection quality of the quantile value function remains constant. This is because of the increasing number of misleading observations at the border of the target. Although the probability of hitting the target from a specific border state is quite small, since there are many border states, observing a hit from one of the border states is quite likely, leading to the assumption that shooting from this state the probability of hitting the target is high. This assumption leads to policies that move to such a border state and always shoot from there. With 1,000 and 2,000 observations, the problem becomes even more pronounced—selecting a policy based only on the standard value function leads to the selection of badly performing policies, as those are the ones with massively overestimated value functions. This is expected in domains exhibiting the "border-phenomenon" [10], where most observations are focused in a favorable area of the state space. The border is only explored rarely but relatively large; it is therefore likely to observe a positive reward by chance. With increasing dimensionality of the state space, the border increases.

To illustrate the overestimation, fig. 4 shows histograms of the estimated mean values for different true policy qualities (exemplary for 2,000 observations). In the left column the standard value functions are depicted, the right column shows the histograms of the quantile values. The top row contains values for the best policies (mean reward greater than 0.4), the second and third row intermediate policies, the bottom row shows bad policies (mean reward less than 0.2). While the value function itself does not allow the selection of good policies (all histograms lie in the same range), the quantile value function reflects the true value more clearly (histograms move from lower to larger values with increasing true policy performance).

C. Wet-Chicken

In the wet-chicken benchmark [17] a canoeist paddles on a one-dimensional river with length $l = 20$ and flow velocity $v = 1$. At position $x = l$ of the river there is a waterfall. Starting at position $x = 0$, the canoeist has to try to get as near as possible to the waterfall without falling down. If he falls down, he has to restart at position $x = 0$. The reward increases linearly with the proximity to the waterfall and is given by $r = x$. The canoeist has the possibility to drift ($x - 0 + v = x + 1$), to hold the position ($x - 1 + v = x$), or to paddle back ($x - 2 + v = x - 1$). River turbulences of size $s = 2.5$ cause the state transitions to be stochastic. Thus, after having applied the canoeist's action to his position (also considering the flow of the river), the new position is finally given by $x' = x + n$, where $n \in [-s, s]$ is a uniformly distributed random value.

Here an exploration run consists of one continuous trajectory. Due to the stochasticity of the turbulences, there are situations when the canoeist is very near the waterfall without falling down. Although the probability of falling down from a point like this is high, limited observations can cause the estimator to misleadingly indicate a high probability of not falling down. Since the reward close to the waterfall is high,

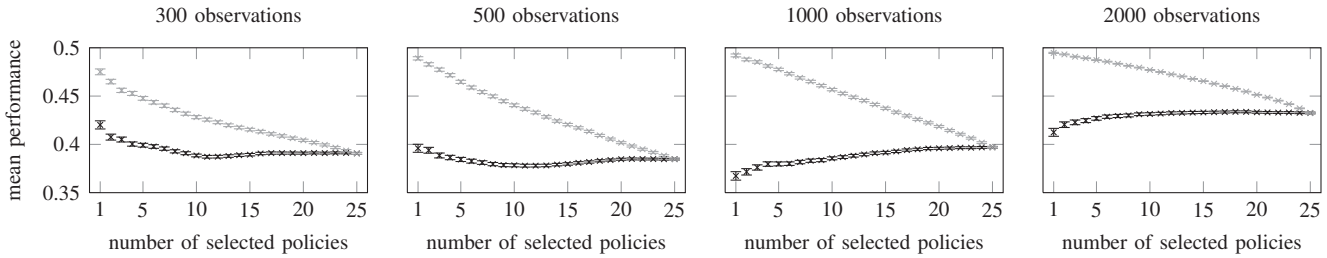

Fig. 3: Results of experiments using the archery benchmark. Shown are the performance of policies ranked either using $J(\pi)$ (black) or $J_u(\pi)$ (gray). The very left point in each plot shows the expected performance of the policy ranked best, the very right point gives the mean performance of all policies.

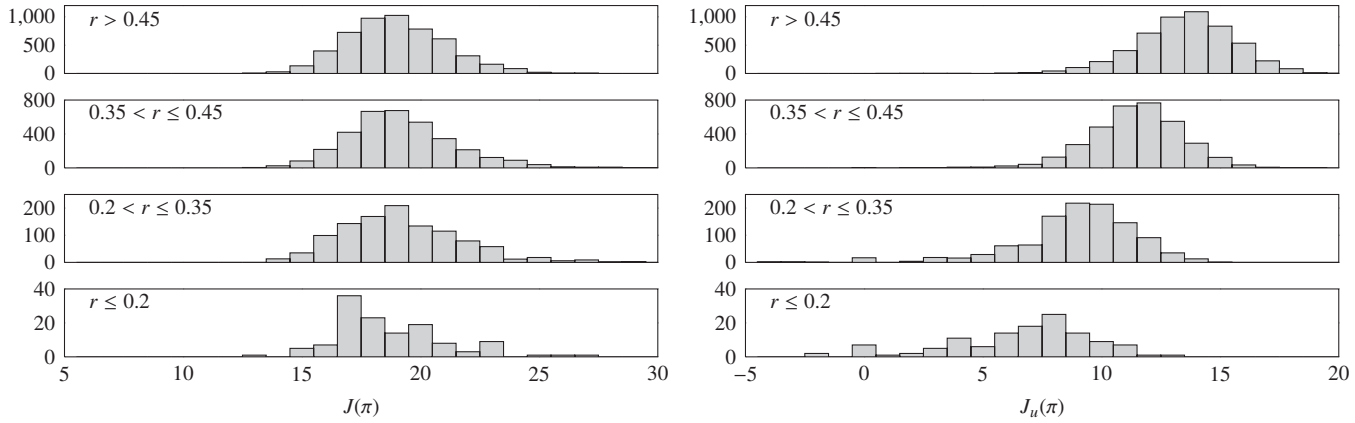

Fig. 4: Histograms of $J(\pi)$ and $J_u(\pi)$ for different true policy performances for the experiment using the archery benchmark with 2,000 observations. The left column shows $J(\pi)$ (ignoring uncertainty), the right column shows $J_u(\pi)$ values considering the uncertainty. The policies are ordered by true performance with the best policies in the top row ($r > 0.45$), intermediate policies in the second and third rows, and the worst policies in the fourth row ($r \leq 0.2$).

a policy generated using those estimators would try to reach a point near the waterfall, expecting to stay there without falling down and to receive a high reward.

Fig. 5 shows the results for different numbers of observations. In the wet-chicken domain the policy selection using the value function systematically selects bad policies. In this setting it is better to pick a policy randomly than choosing the one with the best value function. E.g., for 2,000 observations most policies are near-optimal, but the over-optimistic value function of some policies leads to the selection of bad policies. When considering the uncertainty (with the quantile value function), the situation changes, since the overestimated values are affected by a high uncertainty.

D. Trap

The trap domain [18] is a maze containing 18 states and four possible actions. The agent must collect flags and deliver them to the goal. For each flag delivered the agent receives a reward. However, the maze also contains a trap state. Entering the trap state results in a large negative reward. With probability 0.8 the agent's action has the desired effect, with probability 0.2 the agent moves in perpendicular direction (chosen randomly with equal probability). See fig. 7 for an illustration.

The results from this domain are shown in fig. 6. For 300 and 500 observations we see the same effects as with the other domains—while the value function based approach systematically selects bad policies, considering the uncertainty of the value function as well it is possible to overcome this problem and select good policies. However, for 1,000 and 2,000 observations in the setting chosen here with $\xi = 3$ also the uncertainty aware approach tends to systematically select bad policies, albeit not as extreme as the value function only approach. Setting ξ to a higher value would help here, but could lead to a dominance of the uncertainty for cases with fewer observations.

The optimal policy for this domain tries to stay away from the trap state. After collecting the flag, it goes back to the start state, then two fields down, and finally two fields right to deliver the flag. If an observation set does not contain the event of entering the trap state accidentally from the state left of it, the resulting policy will try to take the shortest path from the flag state to the goal, closely passing the trap state. Since the path is shorter, the corresponding estimated value function will misleadingly contain larger values than those of a more defensive (and in fact better) policy.

VI. Conclusion

In this paper, we made a first attempt at comparing policies without executing them on the real MDP. We showed that the value function can be misleading and largely overestimate the

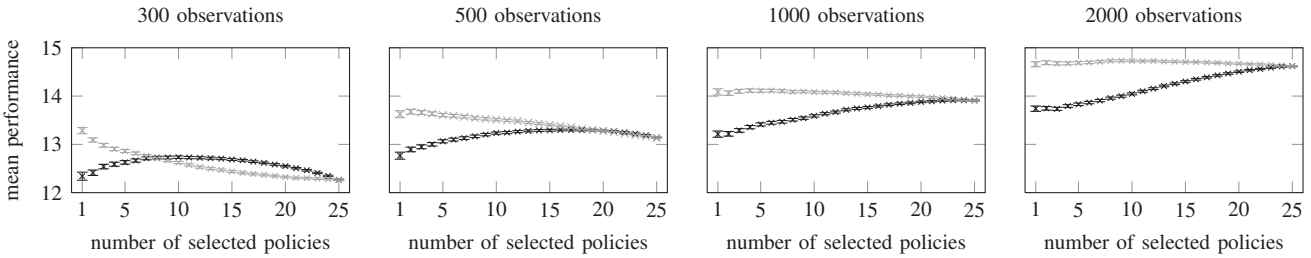

Fig. 5: Performance of ranked policies for the wet-chicken benchmark. The ranking using the standard value function is marked black, the ranking according to the quantile value function gray.

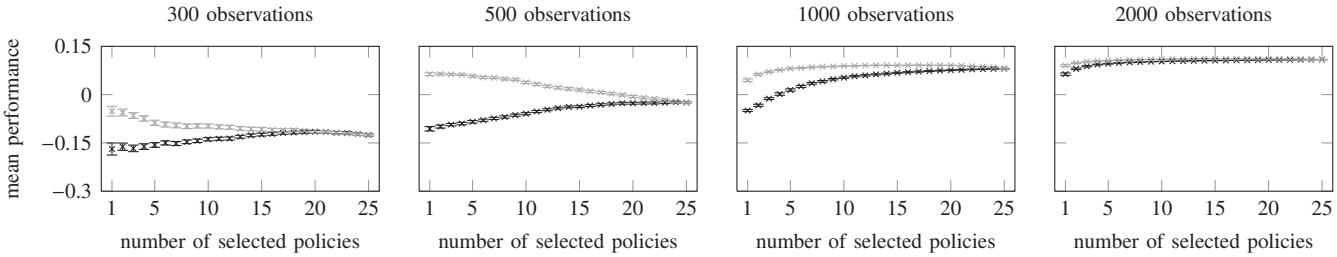

Fig. 6: Performance of ranked policies for the trap domain. Again, the result of the standard value function based ranking is marked black, the ranking also incorporating the uncertainty is marked gray.

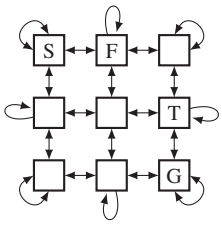

Fig. 7: Illustration of the trap domain. Starting in state S the agent must collect the flag from state F and deliver it to the goal state G while avoiding the trap state T.

quality of the policy. To address this problem, we used uncertainty propagation to determine the uncertainty of the value function as well. Considering the uncertainty to determine the quantile value function it becomes possible to much more reliably distinguish between good and bad policies. Although the discrete MDP setting we considered here is rarely relevant in practice—instead of estimating MDPs and corresponding optimal policies from a number of distinct observation sets and then selecting from those policies, one would use all available observations to estimate a single MDP—for continuous state (and action) MDPs it is an important issue. Future work will deal with adapting the ideas to continuous MDPs.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[2] V. Stephan, K. Debes, H.-M. Gross, F. Wintrich, and H. Wintrich, "A new control scheme for combustion processes using reinforcement learning based on neural networks," *International Journal of Computational Intelligence and Applications*, 2001.

[3] A. Schaefer, D. Schneegass, V. Sterzing, and S. Udluft, "A neural reinforcement learning approach to gas turbine control," in *Proc. of the 20th International Joint Conference on Neural Networks*, 2007.

[4] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, 2003.

[5] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, 2005.

[6] M. Riedmiller, "Neural fitted Q-iteration – first experiences with a data efficient neural reinforcement learning method," in *Proc. of the 16th European Conference on Machine Learning*, 2005.

[7] D. Schneegass, S. Udluft, and T. Martinetz, "Neural rewards regression for near-optimal policy identification in Markovian and partial observable environments," in *Proc. of the European Symposium on Artificial Neural Networks*, 2007.

[8] A. M. Schaefer, S. Udluft, and H.-G. Zimmermann, "A recurrent control neural network for data efficient reinforcement learning," in *Proc. of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

[9] S. Kalyanakrishnan and P. Stone, "Batch reinforcement learning in a complex domain," in *Proc. of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007.

[10] D. Schneegass, S. Udluft, and T. Martinetz, "Uncertainty propagation for quality assurance in reinforcement learning," in *Proc. of the International Joint Conference on Neural Networks*, 2008.

[11] A. Hans and S. Udluft, "Efficient uncertainty propagation for reinforcement learning with limited data," in *Proc. of the International Conference on Artificial Neural Networks*, 2009.

[12] ——, "Uncertainty propagation for efficient exploration in reinforcement learning," in *Proc. of the 19th European Conference on Artificial Intelligence*, 2010.

[13] T. Gabel and M. Riedmiller, "Reducing policy degradation in neuro-dynamic programming," in *Proc. of the European Symposium on Artificial Neural Networks*, 2006.

[14] M. Migliavacca, A. Precorino, M. Pirotta, M. Restelli, and A. Bonarini, "Fitted policy search: Direct policy search using a batch reinforcement learning approach," in *Proc. of the 3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems*, 2010.

[15] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[16] G. D'Agostini, *Bayesian Reasoning in Data Analysis: A Critical Introduction*. World Scientific Publishing, 2003.

[17] V. Tresp, "The wet game of chicken," *Siemens AG, CT IC 4, Technical Report*, 1994.

[18] R. Dearden, N. Friedman, and D. Andre, "Model based Bayesian exploration," in *Proc. of the Conf. on Uncertainty in Artificial Intelligence*, 1999.