Q-learning with Experience Replay in a Dynamic Environment

Pieters, Mathijs; Wiering, Marco

Published in:
Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)

Publication date:
2016

Q-learning with Experience Replay in a Dynamic Environment

Mathijs Pieters
Institute of Artificial Intelligence
and Cognitive Engineering
University of Groningen, The Netherlands
Email: m.t.pieters@rug.nl

Marco A. Wiering (*IEEE Member*)
Institute of Artificial Intelligence
and Cognitive Engineering
University of Groningen, The Netherlands
Email: m.a.wiering@rug.nl

***Abstract*—Most research in reinforcement learning has focused on stationary environments. In this paper, we propose several adaptations of Q-learning for a dynamic environment, for both single and multiple agents. The environment consists of a grid of random rewards, where every reward is removed after a visit. We focus on experience replay, a technique that receives a lot of attention nowadays, and combine this method with Q-learning. We compare two variations of experience replay, where experiences are reused based on time or based on the obtained reward. For multi-agent reinforcement learning we compare two variations of policy representation. In the first variation the agents share a Q-function, while in the second variation both agents have a separate Q-function. Furthermore, in both variations we test the effect of reward sharing between the agents. This leads to four different multi-agent reinforcement learning algorithms, from which sharing a Q-function and sharing the rewards is the most cooperative method. The results show that in the single-agent environment both experience replay algorithms significantly outperform standard Q-learning and a greedy benchmark agent. In the multi-agent environment the highest maximum reward sum in a trial is achieved by using one Q-function and reward sharing. The highest mean reward sum is obtained with separate Q-functions and separate rewards.**

I. Introduction

Reinforcement Learning (RL) [1], [2] is an area within machine learning that is used to control an agent by letting it learn from its interaction with the environment. This environment is often described as a Markov Decision Process [3]. In the environment the agent aims to learn a policy from trial and error that maximizes the cumulative reward. This is different from supervised learning, where examples of correct input-output pairs are given to the system. RL algorithms work by learning a state-action value function that estimates the utility after performing an action in a certain state. With such a state-action value function the optimal policy for an agent consists of selecting the action with the highest value in each state. In order to learn the state-action value function the agent has to explore the environment. The exploration should however not be at the expense of the exploitation of its learned value function. This trade-off is known as the exploration-exploitation dilemma [4]–[6].

Reinforcement learning algorithms are most often applied in stationary environments, although this is not a necessity. Learning in dynamic environments is difficult, because the results of state-action pairs are not consistent over time, which makes predicting future rewards more complex. For non-stationary multi-armed bandit problems, ϵ-greedy action selection, softmax action selection, pursuit methods, and evolutionary algorithms have been compared [7]. Another environment that has been used for studying machine learning in a dynamic environment is robotic navigation. Hierarchical RL, a variation of RL where the problem is divided into different domains, obtained good results for this task [8], [9]. Instead of implicit detection of changes in a dynamic environment, explicit change detection can also be effective. The predictive multi-agent reinforcement learning (P-MARL) algorithm consists of three components, the prediction model, change detection model, and multi-agent system. P-MARL achieved good results in a dynamic multi-agent energy regulation task [10]. A model-based approach for learning in a dynamic environment is that of Instantiated Information, where information about changes in the environment is used to determine a new policy [11], [12].

Contributions In this paper we focus on using reinforcement learning algorithms for solving sequential decision making problems in a non-stationary environment. First, we created dynamic grid problems of several sizes where rewards are only received the first time a location is visited. With this environment, we explore the use of standard Q-learning [13] in a dynamic environment and show that it is not effective. Therefore, we developed a new Q-learning algorithm using a smart exploration policy and combined this method with experience replay (ER) [14]. Experience replay is a promising technique for learning an accurate value function from less experiences. In ER the experiences are stored for later use, in contrast to standard Q-learning. This has the major benefit that important experiences can be presented repeatedly to the underlying learning function. Subsequently, we explore two variations of experience saving, one based on time, and one based on the obtained single-step reward. Furthermore, we also study different multi-agent reinforcement learning methods for handling the dynamic environment. First of all, we compare the results of two agents that share a Q-function (policy sharing) [15] against the results of two agents that have a separate Q-function. We also examine the effect of reward sharing between agents in which the single-step rewards of the agents are averaged and given to both of them. We performed

experiments using two dynamic grid problems of different sizes and compared the proposed single and multi-agent algorithms to standard Q-learning and a greedy benchmark agent. The results show that the proposed methods significantly outperform the standard Q-learning algorithm and the greedy benchmark agent. Furthermore, the most cooperative multi-agent RL system that uses a shared Q-function and shared rewards obtains the highest reward sums in a trial.

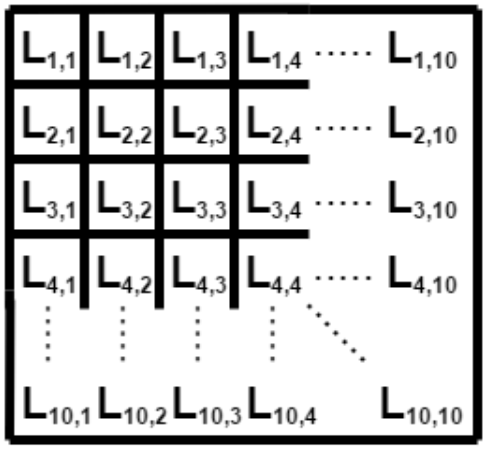

Fig. 1. Depiction of the dynamic 10×10 grid environment

A. The Dynamic Grid Problem

In order to construct a dynamic environment, we propose the use of a grid in which the agent has to maximize its cumulative reward intake. The grid is a two-dimensional environment with a closed border. The grid has a width and height of B (e.g. 10 or 20) discrete locations, illustrated in Figure 1. At each iteration the agent can move in four directions, namely North, East, South, and West. Actions that lead to a collision with a wall cannot be selected from the set of actions.

At each location L there is a randomly initiated reward $r \in [0, 1]$. After each movement the agent receives the reward of the corresponding location. However, after the agent received the reward, the reward of that location is set to zero for the rest of the trial. This makes the environment non-stationary with respect to the reward function.

Since there is no randomness involved in the result of an action in a certain state, the environment is deterministic. We use the environment for both single and multi-agent RL. Below we describe the adaption of the environment for both cases.

1) Single-Agent: We first start with a single agent, where the agent starts in the upper left location ($L_{1,1}$) of the environment. A complete trial consists of 50 (200) movements of the agent in the 10×10 (20×20) environment. It is thus not possible to visit each location in one trial. What makes this problem difficult for RL is that the optimal policy consists of both finding high rewards and avoiding low rewards, while preventing a return to a previous location.

2) Multi-Agent: We will use the same grid environment to study multi-agent RL. One agent starts at the upper left location ($L_{1,1}$), the other at the bottom right location ($L_{10,10}$ or $L_{20,20}$). In order to make the results comparable with single-agent RL, a complete trial now consists of 25 and 100 movements for each of the two agents in the 10×10 and 20×20 grid environments, respectively. Note that the reward of a location is set to zero for both agents if either one of the two agents visits that location.

Outline In Section II we will describe the used reinforcement learning algorithms. In Section III we introduce the adaptation of the RL algorithms for dynamic environments. Section IV presents the experimental results for the grid problem combined with a discussion. Finally, in Section V we summarize the main findings and propose future work.

II. Reinforcement Learning

This section presents the reinforcement learning framework that we will use in the subsequent sections. We begin with describing Markov Decision Processes. Then Q-learning is described, followed by Experience Replay.

A. Markov Decision Processes

In the field of reinforcement learning a Markov Decision Process (MDP) is generally used to describe the environment. The mathematical framework of an MDP can be described with the following terms:

- S is a set of states, where $s_t \in S$ is the state of the environment at time t.
- A is a set of actions, where $a_t \in A$ is the action executed by the agent at time t. n indicates the number of possible actions from the current location. In the environment this number is two for corners, three for edges, and four in every other position.
- $P_a(s, s')$ is the transition function, indicating the probability that after performing action a in state s, the agent arrives at state s'. In the dynamic grid problem, we will make use of a deterministic transition function.
- $R_a(s, s')$ is the reward function, indicating the reward the agent receives after performing action a in state s, leading to s', r_t indicates the reward received at time t.
- $\gamma \in [0, 1]$ is the discount factor, indicating the priority of immediate rewards compared to later rewards.

With the MDP defined, we can now specify the policy $\pi(s)$ that maps states to actions. The definition of $\pi(s)$ depends on the definition of $V(s)$, which contains the average discounted sum of rewards after following policy π from state s:

$$\pi(s) = \arg\max_a \left\{ \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V^\pi(s')) \right\}$$
$$V^\pi(s) = \sum_{s'} P_{\pi(s)}(s, s')(R_{\pi(s)}(s, s') + \gamma V^\pi(s')) \quad (1)$$

For an infinite horizon the expected cumulative reward when following policy π is defined as:

$$E(\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) | a_t = \pi(s_t)) \quad (2)$$

The optimal policy $\pi^*(s)$ maximizes equation 2. The function $V^\pi(s)$ denotes the expected cumulative reward to be received, when the agent follows policy $\pi(s)$ and is currently in state s:

$$V^\pi(s) = E(\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}) | s_0 = s, a_t = \pi(s_t), \pi) \quad (3)$$

We can use equation 1 to define the Q-function, which stores for a state-action pair the expected discounted future reward sum. For a policy π, and an action a in state s, we have:

$$Q^\pi(s, a) = \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V^\pi(s')) \quad (4)$$

The optimal Q-function Q^* can be recursively defined as a set of non-linear equations depending on the reward and transition function [16]. An agent maximizes its utility by selecting at each state the action that has the highest Q-value:

$$Q^*(s, a) = \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma \max_{a'} Q^*(s', a')) \quad (5)$$

The transition function P and the reward function R are fixed in a stationary MDP, in contrast with a non-stationary MDP which we will discuss in Section III.

B. Q-learning

Q-learning [13] is an online RL algorithm, it only uses its last experience to update its policy. The algorithm starts with an arbitrary Q-function Q_0. After each experience (s_t, a_t, r_t, s_{t+1}) at time $t = 1, 2, 3...$, the Q-function is updated as follows:

$$\begin{aligned} Q_{t+1}(s_t,a_t) = Q_t(s_t, a_t) + \\ \alpha[r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)] \end{aligned} \quad (6)$$

where $\alpha \in [0, 1]$ is the learning rate, and γ is the discount factor. The part between square brackets is referred to as the updated value and is the difference between the current estimate of the optimal Q-value $Q_t(s_t, a_t)$ for a state-action pair (s_t, a_t), and the new estimate $r_t + \gamma \max_a Q_t(s_{t+1}, a)$. The Q-function approximates the optimal state-action value function Q^*, independent of the policy that is followed [1]. It is noteworthy that the updated Q-function Q_t only depends on the previous function Q_{t-1} combined with the experience (s_t, a_t, r_t, s_{t+1}). This makes the algorithm both computationally and memory efficient. This is different from other techniques such as Experience Replay [17], where experiences are stored for later use. In order to find an optimal policy Q^*, all state-action pairs (s_t, a_t) should be visited infinitely often [18]. With the Q-function alone this condition can not be satisfied, an exploration method is required. One such an exploration method is the ϵ-greedy policy, which executes either a greedy or random action. With $\epsilon \in [0, 1]$ the action is defined as follows:

$$a_t = \begin{cases} \arg\max_a Q(s_t, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \quad (7)$$

Algorithm 1 Experience Replay

Input: RL algorithm L, exploration probability ϵ, discount factor γ, learning rate α, number of experiences to replay N, number of replays K, number of iterations before learning Z

$Q \leftarrow Q_0$ // Initialize Q-function
$M \leftarrow \emptyset$ // Set of experiences to replay
while not reached stopping criterion **do**
  $c = 0$ // iteration counter
  for $t = 1, 2, .., T$ **do**
    $a_t = \begin{cases} \arg\max_a Q(s_t, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
    perform action a_t
    observe new state s_{t+1} and reward r_t
    $E \leftarrow \{s_t, a_t, r_t, s_{t+1}\}$
    $M \leftarrow update(M, E, N)$
  end for
  $c = c + 1$
  if $c = Z$ **then**
    $Q \leftarrow L(Q, M, K, \gamma, \alpha)$
    $c = 0$ // restart counting
  end if
end while

C. Experience Replay

The Q-learning algorithm only uses the last experience combined with the current Q-function in order to update its function, previous experiences are disregarded. However, it can be advantageous to store the experiences. A rising technique that exploits this strategy is Experience Replay [17], where experiences are stored so that they can be used multiple times to update the policy. Google's DeepMind has recently obtained good results across many Atari games using experience replay by adapting the replay of experiences based on their priority [19]. The Q-learning update function that is used for reinforcement learning does not need any modification, only the moment of applying the function changes. The ER algorithm is shown in Algorithm 1, we will now describe the essential steps. After each action a_t, the experience (s_t, a_t, r_t, s_{t+1}) is stored in the database M. After some trials Z the experiences are used to update the Q-function. We can vary in the number of experiences to replay N and the number of times we learn from those experiences (the number of replays) K, with which we can trade off the speed of learning for computational complexity. In most cases we do not want to store every experience in the database, we thus have to define some pruning method. A general solution is to use only the most recent experiences, so that old experiences are disregarded (Algorithm 2). We also consider another approach, in which we store only the

$N-$best experiences in which the obtained single-step reward was highest, independent of time (Algorithm 3).

Algorithm 2 Update based on time

Input: Database M, experience E, number of experiences N

```
if size(M) < N then
    M ← M ∪ E
else
    remove least recent element from M
    M ← M ∪ E
end if
```

Output: M

Algorithm 3 Update based on reward

Input: Database M, experience E, number of experiences N

```
if size(M) < N then
    M ← M ∪ E
else
    E' ← arg min_E reward(E) for all E ∈ M
    if reward(E') < reward(E) then
        M ← M − E'
        M ← M ∪ E
    end if
end if
```

Output: M

D. Multi-agent Q-learning

The use of Q-learning is not restricted to single-agent environments. In this paper we narrow our focus on multi-agent systems to environments with two agents. We will discuss two variations of Q-learning, adjusted for a multi-agent environment.

1) Shared Q-function: The most self-evident approach is to share one single Q-function between two agents. The agents one by one apply the Q-function update (formula 6) and select ϵ-greedy actions based on the same function. When two agents are at time t in the same state s, both agents will follow exactly the same trajectory as long as both only make greedy actions with respect to the shared Q-function. Due to exploration steps, their paths will finally diverge again, however.

2) Separate Q-function: In the second approach the two agents have separate Q-functions. This can be regarded as two agents independently learning a Q-function, although within the same environment.

Note that for using the shared Q-function the agents need some kind of communication. We can incorporate another kind of communication by means of reward sharing. By knowing the rewards the other agent obtains, the agents have information how well it goes as a group and may also avoid aiming for the same locations. The shared reward received by the agents is then defined as the mean of the separate rewards of the agents after both performed an action. In case two agents visit the same location, both will receive less reward. This is in contrast to agents that do not share their reward, where only the second visiting agent receives no reward. The Q-function for reward sharing is portrayed in formula 8, where $\bar{r}$ means the average reward of the agents:

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha[\bar{r}_t + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)] \quad (8)$$

The four proposed multi-agent reinforcement learning (MARL) algorithms differ whether they use: 1) A shared Q-function or a separate Q-function, and 2) A shared reward function or separate reward functions. The MARL algorithm that shares both the Q-function and the reward function is most cooperative. It relies on maximizing not only the reward of a single agent, but that of both agents, and does this with a single Q-function that allows the combination of experiences of both agents. The algorithm that uses separate reward functions and separate Q-functions is the most competitive MARL algorithm. In this algorithm, both agents only want to optimize their own cumulative reward and do not share anything with the other agent.

III. DQLER: DYNAMIC Q-LEARNING WITH EXPERIENCE REPLAY

Until now we discussed RL algorithms that have been proven to be successful in stationary MDPs. In a non-stationary MDP the transition function P and/or the reward function R are susceptible to change. Non-stationary MDPs can be regarded as a generalization of stationary MDPs, where data are no longer assumed to be stationary [20]. With a stationary reward function we have $R_a^t(s,s') = R_a^{t+\Delta t}(s,s')$, where Δt is some arbitrary offset in time. With a non-stationary reward function, this does not necessarily hold. The transition function is stationary if $P_a^t(s,s') = P_a^{t+\Delta t}(s,s')$. If this is false for any Δt, the transition function is non-stationary. In the dynamic grid problem we deal with a non-stationary (dynamic) reward function, the transition function is in fact stationary. The reward function is non-stationary, because visiting a location s' twice results in distinctive rewards, the reward namely decreases to zero after the first visit. Note also that the Markov assumption does not hold for a dynamic environment. Furthermore, there are exponentially many action sequences in the considered dynamic grid problem, and dynamic programming techniques [16] cannot be simply used because the Markov assumption does not hold.

A. DQLER

Since the standard approach of using Q-learning is unsuccessful, we adopt Q-learning for a dynamic environment. The main obstacle with Q-learning in the proposed dynamic environment is that the reward function $R_a(s,s')$ changes over time. Since this information is not incorporated in the Q-function without visiting the same state multiple times, the agents base their actions on a sub-optimal Q-function. The

tendency we observed is that with standard Q-learning the agents often tend to go back to locations already visited, resulting in a reward of zero.

DQLER is based on Q-learning and Experience Replay with a smart exploration strategy adapted for dynamic environments. To prevent the agents from visiting the same location recurrently, we save the locations visited by the agents. We call the set with all visited locations V, at the start of a trial V is empty. After each action the new location is added to V. An action a in state s is part of A_v if:

$$P_{a \in A}(s, s') = 1 \text{ for some } s' \in V \tag{9}$$

where A is the set of all possible actions. Note that the DQLER algorithm a-priori knows the transition function, but for the deterministic grid problem this information can be easily obtained. We can now define A_n as the relative complement of A and A_v, thus $A_n = A \setminus A_v$.

Instead of using the ϵ-greedy action selection, we propose another solution. As described before, A_n are the actions that lead to a location that is not yet visited. The set of actions that lead to another location that is already visited is A_v. Because the grid problem is deterministic, it is computationally cheap to keep track of this information. We then define the exploration method of the proposed algorithm as an adjusted ϵ-greedy action selection with $\epsilon \in [0, 1]$ as follows:

$$a_t = \begin{cases} \left.\begin{array}{l} \arg\max_a Q(s_t, a \in A_n) \text{ if } A_n \neq \emptyset \\ \arg\max_a Q(s_t, a \in A_v) \text{ if } A_n = \emptyset \end{array}\right\} & \text{w.p. } 1 - \epsilon \\ \text{random action} & \text{w.p. } \epsilon \end{cases} \tag{10}$$

Now we substitute formula 10 for formula 7 in Algorithm 1 to create the DQLER algorithm. For multiple agents, the list of visited states is kept separately for both agents, so one agent does not know which states the other agent has already visited. Note that essentially the algorithm circumvents visiting some state multiple times, which makes the environment much more predictable. The proposed algorithm does not need a long history of previous state-action pairs to keep track of, and is therefore an elegant solution for the considered problem.

IV. Experiments and Results

In this section we will explain the set-up of the experiments, followed by the results. The goal for the single agent is to maximize the sum of rewards $R = \sum_{t=1}^{T} r_t$, where T is the time horizon. In the multi-agent environment the goal of the agents is to maximize their combined sum of rewards $R = \sum_{t=1}^{T} r_t^1 + \sum_{t=1}^{T} r_t^2$, where r_t^1 and r_t^2 are the rewards of the two different agents at time step t.

A. Greedy Benchmark

For both the single and multi-agent environment we want to compare the results of the proposed algorithms with some benchmark. Since such a benchmark does not exist for the dynamic grid world, we test the proposed algorithms against a greedy agent. The greedy agent a-priori knows the reward and transition functions and selects actions defined by:

$$a_t = \begin{cases} \arg\max_a R_{a \in A_n}(s_t, s'_t) & \text{if } A_n \neq \emptyset \\ \text{random action} & \text{if } A_n = \emptyset \end{cases} \tag{11}$$

where A_n is again the set of actions that leads to a state that is not yet visited and s'_t is known by the greedy agent because it has access to the transition function. The benchmark value for the single-agent environment is $R = \sum_{t=1}^{T} r_t$. In the multi-agent environment we compare the proposed algorithms against multiple greedy agents, which start at the same locations as the learning agents.

B. Experimental Set-up

In the experiment we will make a distinction between a small and large grid, which have a width and height of 10×10 and 20×20, respectively.

1) Small Grid environment: For the single agent experiment one trial exists of 50 actions, for the two agents one trial exists of 25 actions for each agent. One simulation consists of 4000 trials, after each trial the grid environment is set back to its original initialisation of the reward function.

2) Large Grid environment: One trial in the large grid of the single agent experiment consists of 200 actions, for two agents it consists of 100 actions for both agents. In this way the ratio between the number of steps and number of locations is the same in both the small and large grid environments. For the greedy benchmark we adjust the number of steps to match that of the learning agents. One simulation consists again of 4000 trials.

We test each set-up for 100 simulations. At the beginning of each different simulation, each reward for a location in the environment is set to a random value between zero and one. After a coarse search through parameter space we chose to use the following parameters for all Q-learning and ER methods: a learning rate α of 1, a discount factor γ of 0.99, and an exploration probability ϵ of 0.1. The agents also use exploration in the testing phases. Because of the recursive nature of the Q-function, the algorithm implicitly expects initial conditions. As an initial value we used 0.5, which is the average reward in the grid. Furthermore, for ER we set the number of experiences to save and replay N to 50 or 200 (for the large environment). The number of replays K is set to 10, and the number of iterations before learning Z to 1 so that the Q-function is updated after every trial. After some preliminary experiments we found that these combinations of values led to the best outcomes.

We will start off comparing the two update strategies (Algorithm 2, and Algorithm 3) for Experience Replay in a single agent environment. In the second experiment we have a multi-agent environment in which we compare DQLER using a shared Q-function against DQLER using a separate Q-function for both agents. Moreover, for both strategies in the second experiment we once use reward sharing, and once we

do not. In the multi-agent experiment we use the ER update function based on time (Algorithm 2).

In every simulation we measure the maximum and mean reward sum per trial. The maximum is defined as the highest sum of rewards received during a trial in a simulation. The mean reward sum is the average of all reward sums received in all trials. Note that for the greedy agent(s) the mean reward sum is irrelevant since there is no learning involved. The figures show the maximum sum of rewards obtained during some trial until a specific number of iterations, which is therefore constantly increasing.

C. Small Grid Environment

Results with a single agent Figure 2 plots the highest obtained reward sum until some number of iterations for the two DQLER update functions, combined with the highest reward sum of the original Q-learning algorithm and the greedy benchmark. In Table I the mean and max reward sums of the first experiment are depicted, combined with their standard errors.

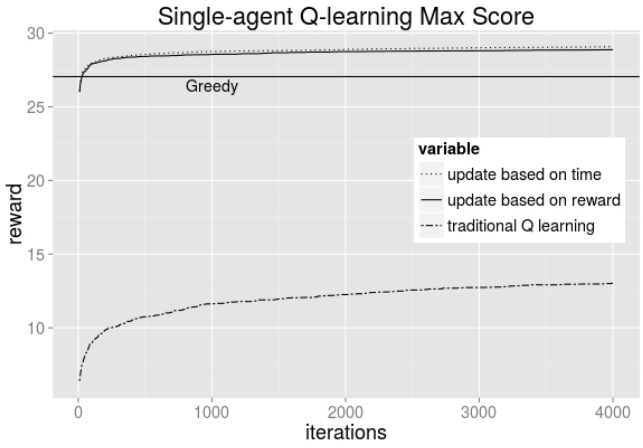

Fig. 2. The figure shows the maximum reward sum in a single trial for DQLER with updates based on time and on reward, for the small single-agent environment. It also shows the results using the greedy benchmark and standard Q-learning. The results are averaged over 100 simulations.

TABLE I
SUM OF REWARDS DURING A TRIAL OF THE SINGLE AGENT IN THE SMALL GRID ENVIRONMENT. THE MAX REWARD SUM IS THE AVERAGE OF THE MAXIMUM SCORE OF A TRIAL FOR THE 100 SIMULATIONS. THE MEAN REWARD SUM IS THE AVERAGE OF THE REWARD SUMS OF EVERY TRIAL FOR THE 100 SIMULATIONS. FOR BOTH MEASURES THE STANDARD ERROR IS INDICATED.

Algorithm	**max reward**	**mean reward**
Standard Q-learning	13.0 ± 0.14	3.9 ± 0.01
DQLER Update based on reward	28.9 ± 0.18	23.0 ± 0.20
DQLER Update based on time	$\mathbf{29.3 \pm 0.15}$	$\mathbf{23.3 \pm 0.18}$
Greedy benchmark	27.0 ± 0.45	$-$

In Figure 2 we see that all three single-agent algorithms start with a steep learning curve. Both DQLER algorithms outperform standard Q-learning from the start. Standard Q-learning never comes close to the greedy benchmark, however both DQLER algorithms outperform the benchmark early on. Table I shows that the ER update based on time achieves a slightly higher average maximum reward sum compared to the ER update based on reward. An unpaired t-test shows however that the difference is not significant. Using an unpaired t-test we find that both DQLER algorithms score a significantly higher average maximum reward sum ($P < 0.001$) compared to the greedy benchmark. With regard to the mean reward, both DQLER algorithms score significantly higher ($P < 10^{-6}$) compared to standard Q-learning.

Results with multiple agents In Figure 3 we show the results of two agents using a combined or separate Q-function, together with the effect of reward sharing and the greedy benchmark. Table II shows the mean and max reward sums and their standard errors for the four learning strategies in the multi-agent environment.

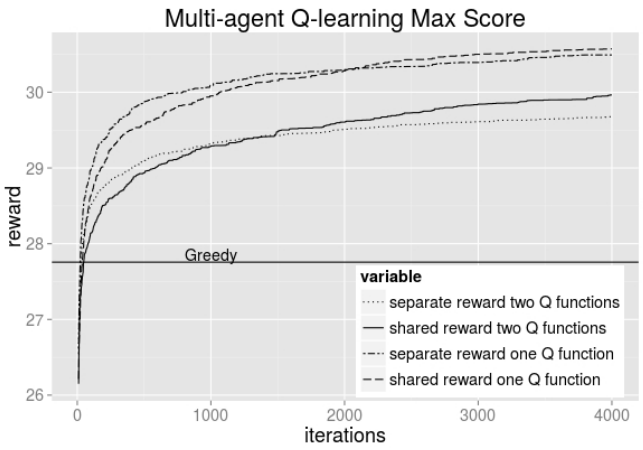

Fig. 3. The figure shows the maximal reward sum in a single trial in the small multi-agent environment for the four DQLER algorithms. It also shows the results of the greedy benchmark. The results are averaged over 100 simulations.

TABLE II
COMBINED SUM OF THE REWARDS OF THE TWO AGENTS IN THE SMALL MULTI-AGENT GRID ENVIRONMENT. THE MAX REWARD IS THE AVERAGE OF THE MAXIMUM SCORE OF A TRIAL FOR EACH OF THE 100 SIMULATIONS. THE MEAN REWARD SUM IS THE AVERAGE OF THE REWARD SUMS OF EVERY TRIAL FOR EACH OF THE 100 SIMULATIONS. FOR BOTH MEASURES THE STANDARD ERROR IS INDICATED.

Algorithm	**max reward**	**mean reward**
Separate reward two Q-functions	29.7 ± 0.18	$\mathbf{24.6 \pm 0.18}$
Shared reward two Q-functions	30.0 ± 0.17	24.2 ± 0.15
Separate reward one Q-function	30.5 ± 0.19	24.4 ± 0.19
Shared reward one Q-function	$\mathbf{30.6 \pm 0.19}$	24.1 ± 0.16
Greedy benchmark	27.8 ± 0.30	$-$

Figure 3 and Table II show that when agents share one Q-function, the maximum reward is higher compared to the use of a separate Q-function, this difference is significant ($P < 0.05$). It seems that sharing the Q-function improves the ability of the agents to determine together which locations should be visited. Furthermore, the use of reward sharing leads to slightly higher maximum rewards, this difference is however not significant. The difference in maximum reward between all four algorithms and the greedy benchmark is very significant ($P < 10^{-6}$). For the mean reward we observe that separate rewards, as well as two Q-functions, leads to higher scores. From this we can conclude that the cooperative MARL algorithm can obtain the best results during some trials, but can suffer from both agents following each other for few steps in case they reach the same state. Therefore, the maximal score is higher, but the mean score is lower for the cooperative MARL method compared to the competitive MARL method that does not share the Q-function or reward function.

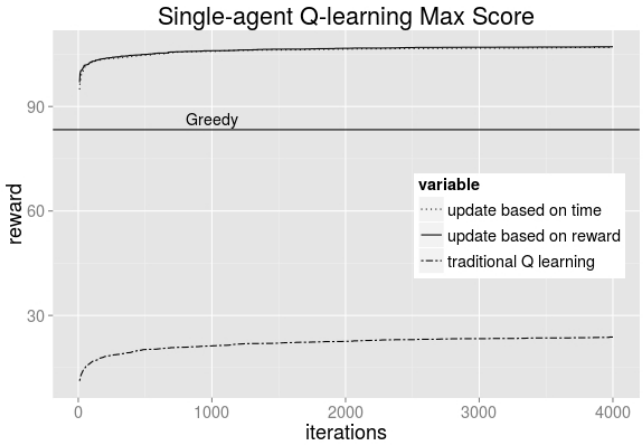

Fig. 4. The figure shows the maximal reward sum in a single trial for DQLER with updates based on time and on reward, for the large single-agent environment. It also shows the results of the greedy benchmark and standard Q-learning. The results are averaged over 100 simulations.

D. Large Grid Environment

Results with a single agent Figure 4 shows the highest reward sums for each iteration of the two DQLER algorithms for the large environment for a single agent, in combination with the highest reward sum of standard Q-learning and the greedy benchmark. Table III shows the mean and max reward sums for the single-agent environment, combined with their standard errors.

The results of the single agent in the large grid environment are comparable to the results in the small grid environment. Both single-agent DQLER algorithms outperform the greedy benchmark with respect to the maximum reward sum. An unpaired t-test shows that the difference is significant ($P < 10^{-6}$). The difference between the standard Q-learning algorithm and the DQLER algorithms has become even more substantial. In Table III we see that the DQLER algorithm based on reward significantly outperforms ($P < 0.005$) the DQLER algorithm based on time with respect to the mean reward. It is therefore profitable to replay good experiences to increase the mean sum of rewards, but for the maximum reward in a trial it is not a significant improvement.

TABLE III
SUM OF REWARDS OF THE SINGLE AGENT IN THE LARGE GRID ENVIRONMENT. THE MAX REWARD IS THE AVERAGE OF THE MAXIMUM SCORE OF A TRIAL FOR EACH OF THE 100 SIMULATIONS. THE MEAN REWARD SUM IS THE AVERAGE OF THE REWARD SUMS OF EVERY TRIAL FOR EACH OF THE 100 SIMULATIONS. FOR BOTH MEASURES THE STANDARD ERROR IS INDICATED.

RL algorithm	max reward	mean reward
Standard Q-learning	23.8 ± 0.19	7.4 ± 0.20
DQLER Update based on reward	$\mathbf{107.2 \pm 0.36}$	$\mathbf{61.3 \pm 0.95}$
DQLER Update based on time	106.9 ± 0.34	57.2 ± 1.00
Greedy benchmark	83.3 ± 2.77	$-$

Results with multiple agents Figure 5 shows the results of the two agents using a combined or separate Q-function, together with the effect of reward sharing and the greedy benchmark. Table IV shows the mean and max reward sums for the four learning strategies in the multi-agent environment, combined with their standard errors.

TABLE IV
COMBINED SUM OF THE REWARDS OF THE TWO AGENTS IN THE LARGE MULTI-AGENT GRID ENVIRONMENT. THE MAX REWARD IS THE AVERAGE OF THE MAXIMUM SCORE OF A TRIAL FOR THE 100 SIMULATIONS. THE MEAN REWARD SUM IS THE AVERAGE OF THE REWARD SUMS OF EVERY TRIAL FOR THE 100 SIMULATIONS. FOR BOTH MEASURES THE STANDARD ERROR IS INDICATED.

RL algorithm	max reward	mean reward
Separate reward two Q-functions	107.4 ± 0.35	$\mathbf{81.3 \pm 0.60}$
Shared reward two Q-functions	107.7 ± 0.34	77.1 ± 0.47
Separate reward one Q-function	108.5 ± 0.35	76.8 ± 0.72
Shared reward one Q-function	$\mathbf{109.0 \pm 0.39}$	74.7 ± 0.46
Greedy benchmark	102.6 ± 1.41	$-$

In the multi-agent environment the differences between the proposed algorithms and the greedy benchmark are very significant ($P < 10^{-6}$). Again reward sharing leads to higher maximum reward sums, although the differences are not significant. The highest maximum reward sum is achieved again by the cooperative MARL algorithm that uses both reward sharing and Q-function sharing. Using this approach leads to significant ($P < 0.05$) better results compared to both non-reward sharing with two Q-functions and reward sharing using two Q-functions. As can be seen in Table IV two agents that have separate rewards and separate Q-functions outperform every other examined multi-agent RL algorithm regarding the mean reward. This difference is highly significant ($P < 0.0001$). Moreover, the differences in mean reward between separate and shared rewards is also significant

($P < 0.05$). Thus again we can conclude that the most cooperative MARL method can obtain the highest maximal score during a simulation, but the most competitive MARL algorithm obtains the highest average scores.

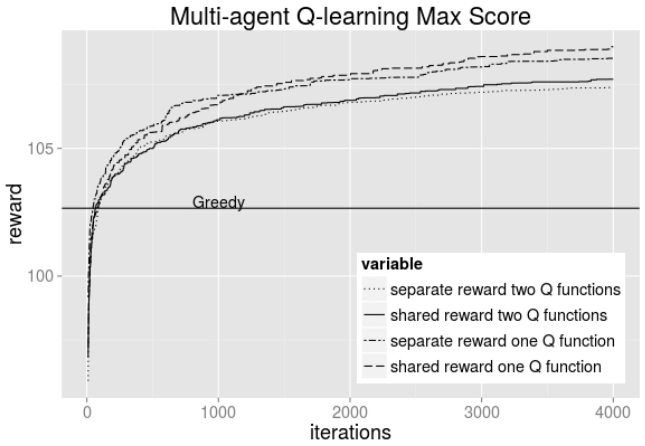

Fig. 5. The figure shows the maximal reward sum obtained in a single trial in the large multi-agent environment for the four DQLER algorithms. It also shows the results of the greedy benchmark. The results are averaged over 100 simulations.

V. Conclusion and Future Work

In this paper we described several reinforcement learning algorithms for solving a dynamic sequential decision making problem. The proposed algorithms are adjusted variants of Q-learning with Experience Replay that use a smart exploration policy. In the single-agent environment we compared two versions of Experience Replay updates, one based on time and one based on reward. Both approaches significantly outperform standard Q-learning, as well as the greedy benchmark agent. In the multi-agent environment we compared the differences in results between two agents that share one Q-function and two agents that have separate Q-functions. Furthermore, we also compared the effect of sharing rewards among the agents. This led to four different algorithms, from which the algorithm sharing the Q-function and the reward function is most cooperative. The results have shown that the cooperative agents achieve higher maximum, and lower mean scores. The lower mean score can mean that the agents follow each other with higher probability in case one Q-function is used. The higher maximum reward sum means that the agents are able to better determine together which locations to visit.

For future work, we want to compare the proposed algorithms to approaches such as ant colony systems [21] and branch and bound techniques that have been shown effective for other planning problems. We also would like to study multi-agent RL with more than two agents, for which the use of one Q-function for all agents could be less effective. Reward sharing can be extended by incorporation of an importance measure of rewards of others. Furthermore, we want to experiment with more complex reward functions, for example reward functions that vary over time. We also want to examine other experience saving techniques for experience replay, possibly using visit counters of state-action pairs. Finally, we want to focus on the application of the proposed algorithms in other non-stationary, stochastic environments, such as forest-fire control [22].

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] M. A. Wiering and M. Van Otterlo, *Reinforcement Learning: State of the Art*. Springer, 2012.

[3] R. Bellman, "A Markovian Decision Process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[4] S. B. Thrun, "Efficient exploration in reinforcement learning," Pittsburgh, PA, USA, Tech. Rep., 1992.

[5] M. A. Wiering and J. H. Schmidhuber., "Efficient model-based exploration," in *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 1998, pp. 223–228.

[6] C. Gentile, S. Li, and G. Zappella, "Online clustering of bandits," *CoRR*, vol. abs/1401.8257, 2014.

[7] D. E. Koulouriotis and A. Xanthopoulos, "Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems," *j-APPL-MATH-COMP*, vol. 196, no. 2, pp. 913–922, mar 2008.

[8] G. G. Yen and T. W. Hickey., "Reinforcement learning algorithms for robotic navigation in dynamic environments," *ISA Transactions*, vol. 43, pp. 217–230, 2004.

[9] A. Buitrago-Martínez, R. F. D. L. Rosa, and F. Lozano-Martínez, "Hierarchical reinforcement learning approach for motion planning in mobile robotics," in *Proceedings of the 2013 IEEE Latin American Robotics Symposium*, ser. LARS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 83–88.

[10] A. Marinescu, I. Dusparic, A. Taylor, V. Cahill, and S. Clarke, "Decentralised multi-agent reinforcement learning for dynamic and uncertain environments," *CoRR*, vol. abs/1409.4561, 2014.

[11] M. A. Wiering, "Reinforcement learning in dynamic environments using instantiated information," in *Proceedings of the Eighth International Conference on Machine Learning*, 2001, pp. 585–592.

[12] M. A. Wiering, "Model-based reinforcement learning in dynamic environments," Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2002-029, 2002.

[13] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 272–292, 1992.

[14] S. Kalyanakrishnan and P. Stone, "Batch reinforcement learning in a complex domain," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '07. New York, NY, USA: ACM, 2007, pp. 94:1–94:8.

[15] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993, pp. 330–337.

[16] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[17] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992.

[18] T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," *Neural Computation*, vol. 6, no. 6, pp. 1185–1201, Nov. 1994.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2015. [Online]. Available: http://arxiv.org/abs/1511.05952

[20] A. Garcia and R. L. Smith., "Solving nonstationary infinite horizon dynamic optimization problems," *Journal of Mathematical Analysis and Applications*, vol. 244, pp. 304,317, 2000.

[21] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *Trans. Sys. Man Cyber. Part B*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[22] M. A. Wiering and M. Dorigo, "Learning to control forest fires," in *Proceedings of the 12th international Symposium on Computer Science for Environmental Protection;, volume 18 of Umweltinformatik Aktuell*. Verlag, 1998, pp. 378–388.