The QV Family Compared to Other Reinforcement Learning Algorithms

Marco A. Wiering and Hado van Hasselt

Abstract— **This paper describes several new online model-free reinforcement learning (RL) algorithms. We designed three new reinforcement algorithms, namely: QV2, QVMAX, and QVMAX2, that are all based on the QV-learning algorithm, but in contrary to QV-learning, QVMAX and QVMAX2 are off-policy RL algorithms and QV2 is a new on-policy RL algorithm. We experimentally compare these algorithms to a large number of different RL algorithms, namely: Q-learning, Sarsa, R-learning, Actor-Critic, QV-learning, and ACLA. We show experiments on five maze problems of varying complexity. Furthermore, we show experimental results on the cart pole balancing problem. The results show that for different problems, there can be large performance differences between the different algorithms, and that there is not a single RL algorithm that always performs best, although on average QV-learning scores highest.**

I. Introduction

Reinforcement learning (RL) algorithms [15], [5] are very suitable for learning to control an agent by letting it interact with an environment. There are a number of different online model-free value-function-based reinforcement learning algorithms that use the discounted future reward criterion. Q-learning [18], Sarsa [10], [14], and Actor-Critic methods [15] are well known, and there are also two more recent algorithms: QV-learning [21], [22] and ACLA [22]. We also include R-learning [11], [6] in our comparison study, which is based on the average reward optimization criterion. Furthermore, a number of policy search and policy gradient algorithms have been proposed [16], [1], and there exist model-based [7] and batch reinforcement learning algorithms [9], but we leave these algorithms out of our current study.

In this paper we describe several new model-free, online RL algorithms, which are inspired by QV-learning. The new algorithms are similar to the existing algorithm, but their learning equations are somewhat different. In total we describe three new RL algorithms: QV2, QVMAX, and QVMAX2. In contrary to QV-learning, QVMAX and QVMAX2 are off-policy RL algorithms and QV2 is a new on-policy RL algorithm. We compare these new QV algorithms to the above mentioned algorithms on a number of different maze tasks and the cart pole balancing problem. The aim of this paper is to research whether there are large performance differences between the algorithms when they are applied for solving different control problems, and whether there is an algorithm that on average performs (much) better than the others. Furthermore, we want to get a better understanding of the difference in learning behavior between on-policy and off-policy algorithms.

Outline. Section II describes previously described online reinforcement learning algorithms. Section III describes the new reinforcement learning algorithms. Then, Section IV describes the results of a number of experiments on problems of varying complexities with tabular and neural network representations. Section V discusses the results and concludes this paper.

Marco Wiering is with the Department of Artificial Intelligence of the University of Groningen (email: mwiering@ai.rug.nl), and Hado van Hasselt is with the Intelligent Systems Group of Utrecht University (email: hado@cs.uu.nl)

II. Reinforcement Learning

Reinforcement learning algorithms are able to let an agent learn from the experiences generated by its interaction with an environment. We assume an underlying Markov decision process (MDP) which is not known to the agent. A finite MDP is defined as; (1) The state-space $S = \{s^1, s^2, \ldots, s^n\}$, and $s_t \in S$ denotes the state of the system at time t; (2) A set of actions available to the agent in each state $A(s)$, where $a_t \in A(s_t)$ denotes the action executed at time t; (3) A transition function $T(s, a, s')$ mapping state-action pairs s, a to a probability distribution over successor states s'; (4) A reward function $R(s, a, s')$ which denotes the average reward obtained when the agent makes a transition from state s to state s' using action a, where r_t denotes the (possibly stochastic) reward obtained at time t.

In optimal control or reinforcement learning (RL), we are interested in computing or learning an optimal policy for mapping states to actions. An optimal policy can be defined as the policy that receives the highest possible cumulative discounted rewards in its future from all states. In order to learn an optimal policy, value-function-based RL [15] estimates value-functions using past experiences of the agent. $Q^\pi(s, a)$ is defined as the expected cumulative discounted future reward if the agent is in state s, executes action a, and follows policy π afterwards:

$$Q^\pi(s,a) = E(\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi)$$

where $0 \leq \gamma < 1$ is the discount factor that values later rewards less compared to immediate rewards. Another possible performance measure is the average reward intake [11], [6]:

$$Q^\pi(s,a) = \lim_{n \to \infty} \frac{1}{n} E(\sum_{i=0}^{n-1} r_i | s_0 = s, a_0 = a, \pi)$$

If the optimal Q-function Q^* is known, the agent can select optimal actions by selecting the action with the largest value in a state: $\pi^*(s) = \arg\max_a Q^*(s, a)$.

In the experiments we compare the new algorithms to six online model-free RL algorithms that are described next.

Q-learning. A famous algorithm for learning a Q-function is Q-learning [18], [19]. Q-learning makes an update after an experience (s_t, a_t, r_t, s_{t+1}) as follows:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

where $0 \leq \alpha \leq 1$ is the learning rate. Q-learning is an off-policy reinforcement learning algorithm [15], which means that the agent learns about the optimal value-function while following another behavioral policy that includes exploration steps. This has the advantage that it does not matter how much exploration is used, as long as the agent visits all state-action pairs an infinite number of times, tabular Q-learning (with appropriate learning rate adaptation) will converge to the optimal Q-function [19], [4], [17]. A disadvantage of Q-learning is that it can diverge when combined with function approximators [2], [3]. Another possible disadvantage is that off-policy algorithms do not modify the behavior of the agent to better deal with the exploration/exploitation dilemma [12].

Sarsa. Instead of Q-learning, we can also use the on-policy algorithm Sarsa [10], [14] for learning Q-values. Sarsa makes the following update after an experience $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Tabular Sarsa converges in the limit to the optimal policy under proper learning rate annealing if the exploration policy is GLIE (greedy in the limit with infinite exploration), which means that the agent should always explore, but stop exploring after an infinite number of steps [12].

R-learning. R-learning [11], [6] uses another optimization criterion than the other RL algorithms described here: instead of optimizing the discounted cumulative future reward it aims to optimize the average future reward. In [6] it was compared to Q-learning and the results showed that Q-learning reached better performance levels on a box-pushing task than R-learning. We want to study in this paper whether these results transfer to other control problems. The learning equations for R-learning are:

$$R(s_t, a_t) := R(s_t, a_t) + \alpha(r_t + \max_a R(s_{t+1}, a) - \rho - R(s_t, a_t))$$

where the ρ variable represents the average reward intake of the policy, regardless of the state the agent occupies. It is updated with the following learning equation:

$$\rho := \rho + \beta(r_t + \max_a R(s_{t+1}, a) - \max_a R(s_t, a) - \rho)$$

We only update ρ when a greedy action a is performed.

Actor-Critic. The Actor-Critic (AC) method is an on-policy algorithm like Sarsa. In contrast to Q-learning and Sarsa, AC methods keep track of two functions; a Critic that evaluates states and an Actor that maps states to a preference value for each action [15]. After an experience (s_t, a_t, r_t, s_{t+1}) AC makes a temporal difference (TD) update [13] to the Critic's value-function V:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma V(s_{t+1}) - V(s_t)) \qquad (1)$$

where β is the learning rate. AC updates the Actor's values $P(s_t, a_t)$ as follows:

$$P(s_t, a_t) := P(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

where α is the learning rate for the Actor. The P-values should be seen as preference values and not as exact Q-values.

QV-learning. QV-learning [22] works by keeping track of both the Q- and V-functions. In QV-learning the state value-function V is learned through TD-methods. This is similar to Actor-Critic methods. The new idea is that the Q-values simply learn from the V-values using the one-step Q-learning algorithm. In contrast to AC these learned values can be seen as actual Q-values and not as preference values. The updates after an experience (s_t, a_t, r_t, s_{t+1}) of QV-learning are the use of Equation 1 and:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

Note that the V-value used in this second update rule is learned by QV-learning and not defined in terms of Q-values. There is a strong resemblance with the Actor-Critic method; the difference is the second learning rule where $V(s_t)$ is replaced by $Q(s_t, a_t)$ in QV-learning.

ACLA. The Actor Critic Learning Automaton (ACLA) [22] learns a state value-function in the same way as AC and QV-learning, but ACLA uses a learning automaton-like update rule [8] for changing the policy mapping states to probabilities (or preferences) for actions. The updates after an experience (s_t, a_t, r_t, s_{t+1}) of ACLA are the use of Equation 1, and an update rule that examines whether the last performed action was good (in which case the state-value was increased) or not. We do this with the following update rule:

$$\begin{array}{lrl}
\text{If } \delta_t \geq 0 & & \Delta P(s_t, a_t) = \alpha(1 - P(s_t, a_t)) \quad \text{and} \\
& \forall a \neq a_t & \Delta P(s_t, a) = \alpha(0 - P(s_t, a)) \\
\text{Else} & & \Delta P(s_t, a_t) = \alpha(0 - P(s_t, a_t)) \quad \text{and} \\
& \forall a \neq a_t & \Delta P(s_t, a) = \alpha(\frac{P(s_t, a)}{\sum_{b \neq a_t} P(s_t, b)} - P(s_t, a))
\end{array}$$

where $\delta_t = \gamma V(s_{t+1}) + r_t - V(s_t)$, and $\Delta P(s, a)$ is added to $P(s, a)$. ACLA uses some additional rules to ensure the targets are always between 0 and 1, independently of the initialization (e.g. of neural network weights). This is done by using 1 if the target is larger than 1, and 0 if the target is smaller than 0. If the denominator is less than or equal to 0, all targets in the last part of the update rule get the value $\frac{1}{|A|-1}$ where $|A|$ is the number of actions. ACLA was shown to outperform Q-learning and Sarsa on a number of problems when ϵ-greedy exploration was used [22].

Comparison between the algorithms. It is known that better convergence guarantees exist for on-policy methods when combined with function approximators [15], since it has been shown that off-policy methods such as Q-learning can diverge in this case [2], [3]. Therefore theoretically there are advantages for using one of the on-policy algorithms. A possible advantage of QV-learning, ACLA, and

AC compared to Q-learning and Sarsa, is that they learn a separate state-value function. This state-value function does not depend on the action and therefore is trained using more experiences than a state-action value function that is only updated if a specific action is selected. When the state-value function is trained faster, this may also cause faster bootstrapping of the Q-values or preference values. A disadvantage of QV-learning, ACLA, and AC and also R-learning is that they need an additional learning parameter that has to be tuned.

III. The Novel QV-family Members

In this section we will present three novel reinforcement learning algorithms, which are inspired by QV-learning.

QV2. QV2 works by keeping track of both the Q- and V-functions. In QV2 the state value-function V is learned with the following equation:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

The Q-function is then learned in the same way as in QV-learning:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

There is a strong resemblance with QV-learning; the difference is the first learning rule where $V(s_t)$ is replaced by $Q(s_t, a_t)$ in QV2.

QVMAX. QVMAX also works by keeping track of both the Q- and V-functions. In QVMAX the state value-function V is learned with the following equation:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma \max_a Q(s_{t+1}, a) - V(s_t))$$

The Q-function is then learned in the same way as in QV-learning:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

There is a strong resemblance with QV-learning; the difference is the first learning rule where $V(s_{t+1})$ is replaced by $\max_a Q(s_{t+1}, a)$ in QVMAX. Note that in contrast to QV-learning, QVMAX is an off-policy algorithm.

QVMAX2. QVMAX2 also works by keeping track of both the Q- and V-functions. In QVMAX2 the state value-function V is learned with the following equation:

$$V(s_t) := V(s_t) + \beta(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

The Q-function is then learned in the same way as in QV-learning:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

There is a strong resemblance with QVMAX; the difference is the first learning rule where $V(s_t)$ is replaced by $Q(s_t, a_t)$ in QVMAX2. QVMAX2 is also an off-policy RL algorithm.

Discussion. QVMAX and QVMAX2 use the Max-operator to get a new, off-policy RL algorithm. It is also possible to replace the Max-operator with a summation sign that weighs the next actions with their probability as defined by the policy. This would amount to two possible new QV-family members.

IV. Experiments

Robot navigation is one important application for reinforcement learning algorithms. However, in contrast to most research in RL that only focuses on static maze tasks, robots often have to deal with a partially observable and dynamic environment. A maze containing obstacles is still a very reasonable model for robot navigation problems in 2D environments. However, we argue that the maze tasks should be of a dynamic and/or partially observable nature to be a challenge. The 6 algorithms: Q-learning, Sarsa, R-learning, AC, QV-learning, and ACLA, are compared to the three novel RL algorithms, QV2, QVMAX, and QVMAX2. We performed experiments with five different maze tasks (one simple and four more complex problems) and the cart pole balancing problem to compare all presented reinforcement learning algorithms. In the first experiment, the RL algorithms are combined with tabular representations and are compared on a small maze task. In all other experiments neural networks are used as function approximators. In the second experiment a partially observable maze is used. In the third experiment a dynamic maze is used where the obstacles are not placed at fixed positions. In the fourth experiment a dynamic maze is used where the goal is not placed at a fixed position. In the fifth experiment a generalized maze [20] task is used where the goal and the obstacles are not placed at fixed positions. In the last experiment the cart pole balancing problem is used.

Experimental set-up for maze experiments. The goal of the maze experiments is to arrive at the goal state as soon as possible under the influence of stochastic (noisy) actions. The reward for arriving at the goal is 100. If the agent bumps against a wall or border of the environment it stays still and receives a reward of -2. For other steps the agent receives a reward of -0.1. A trial is finished after the agent hits the goal or 1000 actions have been performed. The random replacement (noise) in action execution is 20%, this means that a selected action is uniform randomly replaced with one of the four actions with 20% probability. This reward function and noise is used in all maze experiments of this paper.

In the experiments, all parameters were optimized for all RL algorithms, where they were evaluated using the average reward intake and the final performance is optimized. We used the average reward intake just like in the comparison study between Q-learning and R-learning [6] to make a comparison with R-learning (that does not use a discount factor) possible. This also means that for the algorithms that use a discount factor, the discount factor can be treated as a free parameter that can be optimized. Although in general it can cause problems to learn to optimize the discounted reward intake while evaluating with the average reward intake, for the studied problems the dominating objective is to move closer to the goal each step, which is optimized using both criteria for our current experiments. Since we evaluate all methods using the average reward criterion, the different discount factors do not influence the comparison.

We used Boltzmann exploration in the following experiments unless stated differently. Boltzmann exploration performed much better than the use of ϵ-greedy exploration in almost all experiments.

A. Small Maze Experiment

We first performed experiments with Sutton's Dyna maze shown in Figure 1. This simple maze consists of 9×6 states and there are 4 actions; north, east, south, and west. The goal is to arrive at the goal state G as soon as possible starting from the starting state S under the influence of stochastic (noisy) actions. We kept the maze small, since we want to compare the results with the experiments on the more complex maze tasks, which would otherwise cost too much computational time.

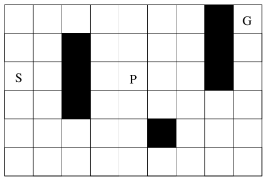

Fig. 1. Sutton's Dyna maze. The starting position is indicated by S and the goal position is indicated by G. In the partially observable maze of the second experiment the goal position is P and the starting position is arbitrary.

We used a tabular representation and first performed simulations to find the best learning rates, discount factors, and greediness (inverse of the temperature) used in Boltzmann exploration. The best learning parameters are shown in Table I. We found that initializing the table entries optimistically (to 100) helped most algorithms except for R-learning and ACLA. For ACLA and R-learning we initialized the table entries to small random values (between -0.1 and 0.1), but we found that an optimistic high initialization value for ρ (to 10) improved the performance of R-learning a lot.

TABLE I
TABULAR LEARNING RATES α/β, DISCOUNT FACTOR AND GREEDINESS (INVERSE OF THE TEMPERATURE FOR BOLTZMANN EXPLORATION) FOR THE ALGORITHMS.

Method	α	β	γ	G
Q	0.2	–	0.9	1
Sarsa	0.2	–	0.9	1
R-learning	0.15	0.0007	–	2
AC	0.1	0.2	0.95	1
QV	0.2	0.2	0.9	1
ACLA+	0.005	0.1	0.99	9
QV2	0.2	0.2	0.9	1
QVMAX	0.2	0.2	0.9	1
QVMAX2	0.2	0.2	0.9	1

In Table II we show average results and standard deviations of 500 simulations of the final reward intake during the last 2500 learning-steps and the total summed reward (adding all 20 average reward intakes after each 2,500 steps) during the entire trial lasting 50,000 learning-steps. This latter evaluation measure shows the overall performance and the learning rate with which good solutions are obtained.

TABLE II
THE FOUR COLUMNS SHOW FINAL AND CUMULATIVE RESULTS FOR THE TABULAR REPRESENTATION AND THE RANKS OF THE DIFFERENT ALGORITHMS (SIGNIFICANCE OF T-TEST $p = 0.05$). RESULTS ARE AVERAGES OF 500 SIMULATIONS.

Method	Final	Rank	Cumulative	Rank
Q	5.22 ± 0.15	7-9	81.8 ± 0.6	9
Sarsa	5.29 ± 0.13	6	84.0 ± 0.5	8
R-learning	5.37 ± 0.12	1	97.1 ± 1.5	1
AC	5.22 ± 0.15	7-9	87.6 ± 2.3	7
QV	5.34 ± 0.16	2-5	94.6 ± 2.6	2-3
ACLA+	5.20 ± 0.23	7-9	90.2 ± 5.2	6
QV2	5.34 ± 0.16	2-5	94.7 ± 2.7	2-3
QVMAX	5.33 ± 0.11	2-5	90.7 ± 0.6	4-5
QVMAX2	5.33 ± 0.15	2-5	90.7 ± 0.7	4-5

The ranks are computed using the student t-test with $p = 0.05$. Note that since 500 simulations were performed, small differences may still turn out to be significant.

The results show that R-learning outperforms all other algorithms. This is not a result of using the average reward intake, and to show this we also computed the final discounted cumulative reward from the starting state with $\gamma = 0.9$ for R-learning, Sarsa, Q-learning, and QV-learning. R-learning had the highest discounted cumulative reward intake of 16.0 $\pm$ 0.7 followed QV-learning (15.8 $\pm$ 1.0), then Sarsa (15.6 $\pm$ 0.8) and the worst performance was by Q-learning (15.2 $\pm$ 0.8). This also shows that both performance measures rank the algorithms in the same way. Furthermore, we note that the QV family performs very well, and that (maybe) surprisingly Q-learning performs not very well.

B. Partially Observable Maze

In this experiment we use Markov localization and neural networks to solve a partially observable Markov decision process in the case where the model of the environment is known. We use Markov localization to track the belief state (or probability distribution over the states) of the agent given an action and observation after each time-step. This belief state is then the input for a neural network. We used 20 sigmoidal hidden neurons in our experiments, and the maze shown in Figure 1 with the goal indicated by P and each state can be a starting state. The initial belief state is a uniform distribution where only states that are not obstacles get assigned a non-zero belief. After each action a_t the belief state $b_t(s)$ is updated with the observation o_{t+1}:

$$b_{t+1}(s) = \eta P(o_{t+1}|s) \sum_{s'} T(s', a_t, s) b_t(s')$$

where η is some normalization factor. The observations are whether there is a wall to the north, east, south, and west. Thus, there are 16 possible observations. We use 20% noise in the action execution and 10% noise for observing each independent wall (or empty cell) at the sides. That means that an observation is correct with probability 0.9^4 = 66%. Note that we use a model of the environment to be able to compute the belief state, and the model is based on the uncertainties in the transition and observation functions.

We performed experiments consisting of 100,000 learning steps with Boltzmann exploration. For evaluation after each 5,000 steps we measured the average reward intake during that period. Table III shows the best learning parameters. The neural network weights were initialized to random values between -0.1 and 0.1. For R-learning we again initialized ρ optimistically. This weight initialization is used in all further maze experiments. Only for R-learning we needed to use ϵ-greedy exploration, since this worked significantly better than the use of Boltzmann exploration. The other algorithms performed better with Boltzmann exploration.

TABLE III
LEARNING PARAMETERS FOR THE PARTIALLY OBSERVABLE MAZE.

Method	α	β	γ	G
Q	0.02	–	0.95	1
Sarsa	0.02	–	0.95	1
R-learning	0.007	0.002	–	ϵ = 0.05
AC	0.02	0.03	0.95	1
QV	0.02	0.01	0.9	1
ACLA+	0.035	0.005	0.99	10
QV2	0.02	0.01	0.9	1
QVMAX	0.02	0.01	0.9	1
QVMAX2	0.02	0.01	0.9	1

TABLE IV
FINAL RESULTS (AVERAGE REWARD FOR LAST 5,000 STEPS) AND CUMULATIVE RESULTS FOR A NEURAL NETWORK REPRESENTATION ON THE PARTIALLY OBSERVABLE MAZE. RESULTS ARE AVERAGES OF 100 SIMULATIONS.

Method	Final	Rank	Cumulative	Rank
Q	9.65 ± 0.33	1-4	154.1 ± 7.3	2
Sarsa	9.41 ± 1.26	1-8	137.6 ± 21.1	5-9
R-learning	9.22 ± 0.27	7-8	149.5 ± 10.2	3
AC	9.33 ± 0.31	6-7	159.4 ± 3.5	1
QV	9.59 ± 0.31	1-6	135.3 ± 12.3	6-9
ACLA+	8.44 ± 0.27	9	135.1 ± 3.7	6-9
QV2	9.55 ± 0.30	2-6	133.0 ± 13.5	6-9
QVMAX	9.59 ± 0.28	1-6	141.4 ± 8.0	4-6
QVMAX2	9.56 ± 0.29	2-6	142.4 ± 8.3	4-5

Table IV shows that Q-learning achieves the best overall performance, while ACLA is falling behind all other algorithms. The QV-algorithms all perform similarly, although the off-policy QVMAX and QVMAX2 algorithms learn faster than QV-learning and QV2. The learning curves of Q-learning, QV-learning and R-learning are shown in Figure 2. The learning curves of the other algorithms are similar.

C. Solving a Maze with Dynamic Obstacles

We also compared the algorithms on a dynamic maze, where in each trial there are several obstacles at random locations (see Fig. 3).

In order to deal with this task the agent uses a neural network that receives as inputs whether a particular state-cell contains an obstacle (1) or not (0). The neural network uses $2 \times 54 = 108$ inputs including the position of the agent and 60 sigmoidal hidden units. At the start of each new trial there are between 4 and 8 obstacles generated at random positions and it is made sure that a path to the goal exists from the

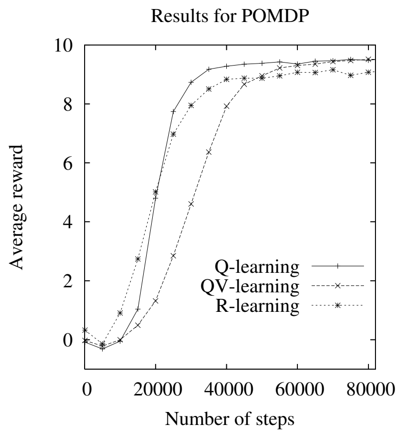

Fig. 2. The learning curves for Q-learning, QV-learning and R-learning on the POMDP.

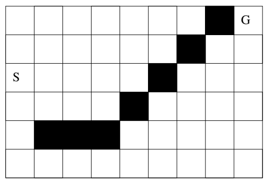

Fig. 3. The 9×6 maze with dynamic obstacles used in the third experiment. The starting position is denoted by S and the goal position is indicated by G. The obstacles indicated in black are dynamically generated at the start of each new trial.

fixed starting location S. Since there are many instances of this maze, the neural network has to learn the knowledge of a path planner. A simulation lasts for 3,000,000 learning steps and we measure performance after each 150,000 steps. The learning parameters are shown in Table V. Again R-learning performed much better with ϵ-greedy exploration than with Boltzmann exploration.

TABLE V
LEARNING PARAMETERS FOR THE MAZE WITH DYNAMIC OBSTACLES.

Method	α	β	γ	G
Q	0.01	–	0.95	1
Sarsa	0.01	–	0.95	1
R-learning	0.0005	0.0003	–	$\epsilon = 0.05$
AC	0.015	0.003	0.95	1
QV	0.01	0.01	0.9	0.4
ACLA+	0.06	0.002	0.98	6
QV2	0.01	0.01	0.9	0.4
QVMAX	0.01	0.01	0.9	0.4
QVMAX2	0.01	0.01	0.9	0.4

Table VI shows the final and total performance of the different algorithms. It shows that Q-learning performs best, followed by Sarsa. R-learning did not perform very well, despite extensive parameter tuning to get its optimal performance. This may be explained by the dynamic nature of this problem where average reward intake fluctuates quite a lot.

D. Solving a Maze with Dynamic Goal Positions

In this fourth maze experiment, we use the same small maze as before (see Figure 1) where the starting position is indicated by S, but now the goal is placed at a different

TABLE VI

Final results (average reward for last 150,000 steps) and cumulative results for a neural network representation on the maze with dynamic obstacles. Results are averages of 50 simulations.

Method	Final	Rank	Cumulative	Rank
Q	6.79 ± 0.21	1	116.2 ± 2.7	1
Sarsa	6.66 ± 0.37	2	112.6 ± 5.9	2
R-learning	4.59 ± 0.39	9	45.7 ± 5.5	9
AC	5.98 ± 0.31	7	97.7 ± 9.4	7
QV	6.27 ± 0.20	3-6	108.3 ± 2.7	3-5
ACLA+	5.39 ± 0.15	8	89.2 ± 1.3	8
QV2	6.31 ± 0.20	3-6	108.9 ± 2.4	3-4
QVMAX	6.25 ± 0.19	3-6	107.2 ± 2.0	6
QVMAX2	6.25 ± 0.15	3-6	108.0 ± 1.7	4-5

location in each trial. To deal with this, we use a neural network function approximator that receives the position of the goal as input. Therefore there are 54 $\times$ 2 inputs, that indicate the position of the agent and the position of the goal. A simulation lasts for 3,000,000 learning steps and we measure performance after each 150,000 steps. We used feedforward neural networks with 20 sigmoidal hidden units. The learning parameters are shown in Table VII.

TABLE VII

Learning parameters for the maze with dynamic goal positions.

Method	α	β	γ	G
Q	0.005	–	0.95	0.5
Sarsa	0.008	–	0.95	0.6
R-learning	0.003	0.0015	–	$\epsilon = 0.07$
AC	0.006	0.008	0.95	0.6
QV	0.012	0.004	0.95	0.6
ACLA+	0.06	0.006	0.98	10
QV2	0.012	0.004	0.95	0.6
QVMAX	0.012	0.004	0.95	0.6
QVMAX2	0.012	0.004	0.95	0.6

Table VIII shows the final and total performance of the different algorithms. It shows that QV2 reaches the best final performance whereas AC learns fastest. R-learning again performs worst for this problem followed by Q-learning, QVMAX and ACLA.

TABLE VIII

Final results (average reward for last 150,000 steps) and cumulative results for a neural network representation on the maze with dynamic goal positions. Results are averages of 50 simulations.

Method	Final	Rank	Cumulative	Rank
Q	10.05 ± 0.37	6-8	152.7 ± 8.3	7
Sarsa	10.69 ± 0.47	1-4	176.9 ± 8.7	2
R-learning	4.79 ± 3.50	9	16.5 ± 12.4	9
AC	10.65 ± 0.11	2-4	180.5 ± 3.3	1
QV	10.66 ± 1.16	1-6	169.4 ± 20.2	3-4
ACLA+	10.11 ± 1.80	4-8	121.5 ± 25.6	8
QV2	10.83 ± 0.35	1-3	171.7 ± 10.9	3-4
QVMAX	10.07 ± 1.39	5-8	160.0 ± 23.1	5-6
QVMAX2	10.33 ± 0.85	4-7	161.8 ± 20.2	4-6

E. Solving the Generalized Maze

In this last maze experiment, we use the same small maze as before, but now the goal and walls are placed at a different location in each trial. This is what Werbos and Pang call the "Generalized Maze" [20] experiment. To deal with this, a neural network function approximator receives the position of the agent, goal and the dynamic walls as input. Therefore there are 54 $\times$ 3 inputs. A simulation lasts for 15,000,000 learning steps and we measure performance after each 750,000 steps. The feedforward neural networks have 100 sigmoidal hidden units. The learning parameters are shown in Table IX. Surprisingly R-learning performed a bit better for this problem with Boltzmann exploration than with ϵ-greedy exploration, although its performance was anyway not very good.

TABLE IX

Learning parameters for the generalized maze.

Method	α	β	γ	G
Q	0.003	–	0.95	0.3
Sarsa	0.003	–	0.92	0.3
R-learning	0.0001	0.00005	–	1.0
AC	0.014	0.0015	0.95	0.5
QV	0.002	0.001	0.95	0.2
ACLA+	0.1	0.001	0.98	5
QV2	0.002	0.001	0.95	0.2
QVMAX	0.002	0.001	0.95	0.2
QVMAX2	0.002	0.001	0.95	0.2

Table X shows the final and total performance of the different algorithms. Here Q-learning obtains the best final results and AC comes as second best. Although again R-learning performed worst, it is surprising that Sarsa also obtained much worse results than the other algorithms. Off-policy discounted methods such as QVMAX and QVMAX2 perform better than on-policy variants such as QV and QV2.

TABLE X

Final results (average reward for last 750,000 steps) and cumulative results for a neural network representation on the generalized maze. Results are averages of 50 simulations.

Method	Final	Rank	Cumulative	Rank
Q	6.92 ± 0.16	1	96.6 ± 1.1	1
Sarsa	1.06 ± 0.20	8	13.3 ± 0.9	8
R-learning	0.79 ± 0.04	9	7.1 ± 0.1	9
AC	5.84 ± 0.15	2	89.0 ± 1.0	2
QV	5.17 ± 0.16	5-6	76.6 ± 1.1	5-6
ACLA+	4.81 ± 0.12	7	56.7 ± 1.5	7
QV2	5.19 ± 0.13	5-6	76.9 ± 1.1	5-6
QVMAX	5.63 ± 0.14	3	80.6 ± 0.9	3-4
QVMAX2	5.55 ± 0.16	4	80.6 ± 1.0	3-4

The learning curves for Q-learning, QV-learning, and R-learning are shown in Figure 4.

F. Cart Pole

For the last experiment we investigated performance on a well known different type of task: the Cart Pole. In this task the agents should learn to balance a pole on a cart by

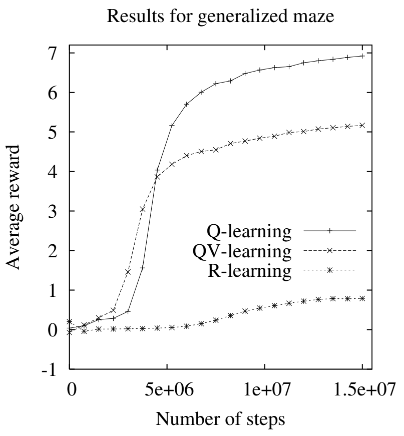

Fig. 4. The learning curves of Q-learning, QV-learning, and R-learning on the generalized maze.

pushing the cart left or right. The action space consists of all integer amounts of force between -10 N and 10 N, where negative force corresponds to pushing left and positive force to pushing right. Whenever the pole had an angle of more than 12 degrees with the vertical line or whenever the cart was more than 2.4 m from the center of the track the agent received a reward of -1 and the trial was finished. On all other time steps, the agent received a positive reward of 1. Each trial starts with the cart at the center of the track and the pole at a random angle between -3 and 3 degrees. One step lasts 0.02 simulated seconds and a whole trial lasts 4000 simulated seconds. The state space consists of the position and velocity of the cart and the angle and angular velocity of the pole. All value functions were approximated by neural networks with 15 sigmoidal hidden units.

Table XII shows the result of the different algorithms on this task and Table XI shows the parameters that were used to obtain these results. All these results were obtained during and after training on separate test runs that were run every 40 seconds without updating the algorithms and without exploration.

TABLE XI
LEARNING PARAMETERS FOR THE CART POLE TASK.

Method	α	β	γ	G
Q	0.008	–	0.95	1
Sarsa	0.004	–	0.95	2
R-learning	0.008	0.001	–	10
AC	0.01	0.002	0.95	2
QV	0.01	0.004	0.95	1
ACLA+	0.08	0.004	0.95	10
QV2	0.002	0.0001	0.95	0.5
QVMAX	0.01	0.01	0.95	1
QVMAX2	0.01	0.0004	0.95	0.2

As can be seen in Table XII, in this task QV obtains a solution that balances the pole for at least 20 seconds the fastest, on average after 171 seconds. Note that in fact the solution will probably be found earlier, but we only tested for successful runs every 40 seconds.

It is interesting to note that even though most algorithms find a perfect policy for this problem on some point during training, none reached perfect final results on every trial. This is due to a problem related to overfitting: because the algorithms continue to learn after reaching a good policy, they will mostly experience states that are close to optimal and update the neural networks that were used as function approximators with this knowledge. However, because of the distributed nature of these networks, this will cause knowledge about states that are experienced less to fade. As can be seen in Table XIII ACLA, Actor Critic and Sarsa reach the best final average results, with near perfect scores of 0.998, 0.996 and 0.994, respectively. The fast learning of QV can also be seen by its relatively high cumulative results, although ACLA performs better using this performance metric.

TABLE XII
AVERAGE NUMBER OF SECONDS UNTIL THE FIRST TEST RUN WHERE THE POLE WAS BALANCED FOR 20 S AND THE PERCENTAGE OF SIMULATIONS THAT INCLUDED SUCH A SUCCESSFUL TEST RUN ARE GIVEN. IF NO SUCCESSFUL RUN WAS OBSERVED, THE MAXIMUM VALUE OF 4000 S WAS USED TO CALCULATE HET AVERAGE. TEST RUNS WERE CONDUCTED AFTER EVERY 40 S OF TRAINING. IN TOTAL 50 SIMULATIONS WERE CONDUCTED.

Method	**Episodes**	Rank	% success
Q	485 ± 674	3-5	100
Sarsa	811 ± 937	5-6	96
R-learning	3285 ±1355	9	34
AC	1311 ±1297	7	90
QV	171 ± 136	1	100
ACLA+	395 ± 284	3-4	100
QV2	2436 ±1179	8	66
QVMAX	287 ± 183	2	100
QVMAX2	646 ± 635	4-6	100

TABLE XIII
FINAL AND CUMULATIVE RESULTS FOR THE CART POLE. RESULTS ARE AVERAGES OF 50 SIMULATIONS.

Method	Final	Rank	Cumulative	Rank
Q	0.987 ± 0.006	4-7	98.1 ± 0.4	6-7
Sarsa	0.994 ± 0.006	2-3	98.4 ± 0.5	3-5
R-learning	0.967 ± 0.023	8	96.6 ± 1.0	8
AC	0.996 ± 0.006	2-3	98.8 ± 1.6	2-5
QV	0.988 ± 0.008	4-7	98.7 ± 0.3	2-3
ACLA+	0.998 ± 0.002	1	99.6 ± 0.1	1
QV2	0.844 ± 0.265	9	83.6 ± 12.3	9
QVMAX	0.986 ± 0.006	4-7	98.4 ± 0.3	3-5
QVMAX2	0.987 ± 0.006	4-7	98.2 ± 0.3	6-7

V. DISCUSSION

The RL algorithms were compared on six different control problems. We are now interested in comparing the overall final performances of the different algorithms, for which we optimized the learning parameters. Since we used two evaluation measures in the last cart pole experiment, we combine all seven rankings in Table XIV and Table XV.

The overall scores in Table XV show that QV-learning on overall performs best, but it is closely followed by QVMAX, Q-learning, Sarsa, and QVMAX2. R-learning performs worst on average, despite its great learning performance on the

TABLE XIV

THE RANKS OF THE DIFFERENT METHODS WHEN WE LOOK AT FINAL PERFORMANCE OF THE LEARNED CONTROLLERS FOR THE FIRST 4 EXPERIMENTS.

Exp. nr.	1 (Tab.)	2 (NN)	3 (NN)	4 (NN)
Q	7-9	1-4	1	6-8
Sarsa	6	1-8	2	1-4
R-learning	1	7-8	9	9
AC	7-9	6-7	7	2-4
QV	2-5	1-6	3-6	1-6
ACLA	7-9	9	8	4-8
QV2	2-5	2-6	3-6	1-3
QVMAX	2-5	1-6	3-6	5-8
QVMAX2	2-5	2-6	3-6	4-7

TABLE XV

THE RANKS OF THE DIFFERENT METHODS WHEN WE LOOK AT FINAL PERFORMANCE OF THE LEARNED CONTROLLERS FOR THE LAST TWO EXPERIMENTS AND THE OVERALL SCORES.

Exp. nr.	5 (NN)	6a (NN)	6b (NN)	Total
Q	1	3-5	4-7	29
Sarsa	8	5-6	2-3	31
R-learning	9	9	8	52.5
AC	2	7	2-3	36
QV	5-6	1	4-7	27
ACLA	7	3-4	1	42.5
QV2	5-6	8	9	36.5
QVMAX	3	2	4-7	28.5
QVMAX2	4	4-6	4-7	32

simple maze problem. ACLA also does not perform very well on average.

The variants of QV-learning seem to perform a bit worse than QV-learning itself, although QVMAX can outperform QV-learning for problems where off-policy algorithms outperform on-policy algorithms, see for example the fifth experiment. The fourth experiments shows another perspective: here Sarsa outperforms Q-learning and QV and QV2 perform also better than their off-policy counterparts.

The results also indicate that there are for particular control problems large performance differences between the different algorithms. The most striking example is the generalized maze experiment in which Sarsa does not perform well, whereas Q-learning achieves excellent results. It seems that one way out of the dilemma of choosing a single RL algorithm, is to use a combination of all algorithms in an ensemble to get the best out of them [23].

In future work we want to compare all algorithms on the large partially observable generalized maze problem, which will be a very difficult task for RL algorithms. If some of them are able to obtain good results on this problem, it would be interesting to add game-like features such as gold, dragons, exits, and multiple levels. Finally, we want to make the transition to hierarchical extensions of the algorithms and compare their batch versions.

REFERENCES

[1] J. Baxter and P.L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[2] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 369–376. MIT Press, Cambridge MA, 1995.

[3] G.J. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Carnegie Mellon University, 1995.

[4] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[6] S. Mahadevan. To discount or not to discount in reinforcement learning: A case study comparing R learning and Q learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 164–172. Morgan Kaufmann, 1994.

[7] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[8] K. S. Narendra and M. A. L. Thathatchar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:323–334, 1974.

[9] M. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML'05)*, pages 317–328, 2005.

[10] G.A. Rummery and M. Niranjan. On-line Q-learning using connectionist sytems. Technical Report CUED/F-INFENG-TR 166, Cambridge University, UK, 1994.

[11] A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 298–305. Morgan Kaufmann, Amherst, MA, 1993.

[12] S.P. Singh, T. Jaakkola, M.L. Littman, and C. Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[13] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[14] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1045. MIT Press, Cambridge MA, 1996.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT press, Cambridge MA, A Bradford Book, 1998.

[16] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[17] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.

[18] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[19] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[20] P.J. Werbos and X. Pang. Generalized maze navigation: Srn critics solve what feedforward nets cannot. *IEEE Transactions on Systems, Man, and Cybernetics*, 3:1764–1769, 1996.

[21] M.A. Wiering. QV(lambda)-learning: A new on-policy reinforcement learning algorithm. In D. Leone, editor, *Proceedings of the 7th European Workshop on Reinforcement Learning*, pages 29–30, 2005.

[22] M.A. Wiering and H. van Hasselt. Two novel on-policy reinforcement learning algorithms based on TD(λ)-methods. In *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 280–287, 2007.

[23] M.A. Wiering and H. van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions, SMC Part B, special issue on Adaptive Dynamic Programming and Reinforcement Learning in Feedback Control*, 2008.