Ensemble Algorithms in Reinforcement Learning

Marco A. Wiering and Hado van Hasselt

***Abstract*—This paper describes several ensemble methods that combine multiple different reinforcement learning (RL) algorithms in a single agent. The aim is to enhance learning speed and final performance by combining the chosen actions or action probabilities of different RL algorithms. We designed and implemented four different ensemble methods combining the following five different RL algorithms: Q-learning, Sarsa, actor–critic (AC), QV-learning, and AC learning automaton. The intuitively designed ensemble methods, namely, majority voting (MV), rank voting, Boltzmann multiplication (BM), and Boltzmann addition, combine the policies derived from the value functions of the different RL algorithms, in contrast to previous work where ensemble methods have been used in RL for representing and learning a single value function. We show experiments on five maze problems of varying complexity; the first problem is simple, but the other four maze tasks are of a dynamic or partially observable nature. The results indicate that the BM and MV ensembles significantly outperform the single RL algorithms.**

***Index Terms*—Dynamic mazes, ensemble algorithms, partially observable environments, reinforcement learning (RL).**

I. Introduction

REINFORCEMENT learning (RL) algorithms [1], [2] 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 RL algorithms that use the discounted future reward criterion. Q-learning [3], Sarsa [4], [5], and AC methods [1] are well known, and there are also two more recent algorithms: QV-learning [6] and AC learning automaton (ACLA) [6]. Furthermore, a number of policy search and policy gradient algorithms have been proposed [7], [8], and there exist model-based [9] and batch RL algorithms [10].

In this paper, we describe several ensemble methods that combine multiple RL algorithms in a single agent. The aim is to enhance learning speed and final performance by combining the chosen actions or action probabilities of different algorithms. In supervised learning, ensemble methods such as bagging [11], boosting [12], and mixtures of experts [13] have been used a lot. Such ensembles are used for learning and combining multiple classifiers by using, for example, a (weighted) majority voting (MV) scheme. In RL, ensemble methods have been used for representing and learning the value function [14]–[17]. In contrast to this previous research, here, we introduce ensembles that combine different RL algorithms in a single agent. The system learns multiple value functions, and the ensembles combine the policies derived from the value functions in a final policy for the agent. We designed the following ensemble methods for combining RL algorithms: 1) The MV method combines the best action of each algorithm and bases its final decision on the number of times an action is preferred by each algorithm; 2) the rank voting (RV) method lets each algorithm rank the different actions and combines these rankings to select a final action; 3) the Boltzmann multiplication (BM) method is based on using Boltzmann exploration for each algorithm and multiplies the Boltzmann probabilities of each action computed by each algorithm; and 4) the Boltzmann addition (BA) method is similar to the BM method but adds the Boltzmann probabilities of actions.

Outline: Section II describes a number of online RL algorithms that will be used in the experiments. Section III describes different ensemble methods for combining multiple RL algorithms. Then, Section IV describes the results of a number of experiments on maze problems of varying complexities with tabular and neural-network representations. Section V discusses the results and concludes this paper.

II. Reinforcement Learning

RL 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 does not have to be known by the agent. A finite MDP is defined as follows: 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'; and 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 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 [1] estimates value functions by 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 afterward

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

Manuscript received August 31, 2007; revised January 26, 2008. This paper was recommended by Guest Editor F. L. Lewis.

M. A. Wiering is with the Department of Artificial Intelligence, University of Groningen, 9400 AK Groningen, The Netherlands (e-mail: mwiering@ai.rug.nl).

H. van Hasselt is with the Intelligent Systems Group, Department of Information and Computing Sciences, Utrecht University, 3508 TB Utrecht, The Netherlands (e-mail: hado@cs.uu.nl).

Digital Object Identifier 10.1109/TSMCB.2008.920231

where $0 \leq \gamma \leq 1$ is the discount factor that values later rewards less compared with immediate rewards. Another possible objective is to maximize the average reward intake. 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 will use five different online model-free RL algorithms that optimize the discounted cumulative future reward intake of an agent while it is interacting with an (unknown) environment. Q-learning [3], [18] and Sarsa [4], [5] will not be described here because they are very well-known methods.

AC: The 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 [1]. After an experience (s_t, a_t, r_t, s_{t+1}), AC makes a temporal difference (TD) update to the critic's value function V

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

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

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

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 [6] works by keeping track of both the Q- and V-functions. In QV-learning, the state-value function V is learned with TD methods [19]. This is similar to AC 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 (1) and

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

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 AC method; the only difference is the second learning rule where $V(s_t)$ is replaced by $Q(s_t, a_t)$ in QV-learning.

ACLA: ACLA [6] learns a state-value function in the same way as AC and QV-learning, but ACLA uses a learning automatonlike update rule [20] 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 (1), and now, we use 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:

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

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 that the targets are always between 0 and 1, independent of the initialization (e.g., of neural-network weights). This is done by using 1 if the target is larger than 1 and by using 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 $1/(|A| - 1)$, 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 [6].

Comparison Between the Algorithms: It is known that better convergence guarantees exist for on-policy methods when combined with function approximators [1] because it has been shown that Q-learning can diverge in this case [21], [22]. 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 is therefore trained by 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 is that they need an additional learning parameter that has to be tuned.

III. Ensemble Algorithms in RL

Ensemble methods have been shown to be effective in combining single classifiers in a system, leading to a higher accuracy than obtainable with a single classifier. Bagging [11] is a simple method that trains multiple classifiers using a different partitioning of the training set and combines them by majority voting. If the errors of the single classifiers are not strongly correlated, this can significantly improve the classification accuracy. In RL, ensemble methods have been used for combining function approximators to store the value function [14]–[17], and this can be an efficient way for improving an RL algorithm. In contrast to previous research, we combine different RL algorithms that learn separate value functions and policies. Because the value functions of, for example, AC that learns preference values and of Q-learning that learns state-action values are of a different nature, it is impossible to combine their value functions directly. Therefore, in our ensemble approaches, we combine the different policies derived from the value functions learned by the RL algorithms. We

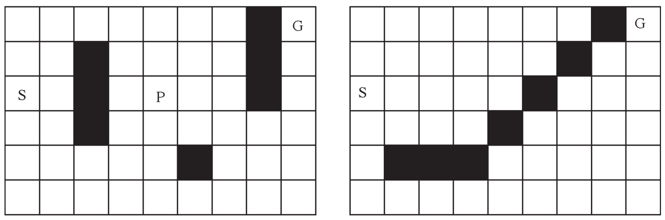

Fig. 1. (a) Sutton's Dyna maze. The starting position is indicated by S, and the goal position is indicated by G. In the partially observable maze of the second experiment, the goal position is P, and the starting position is arbitrary. (b) The 9×6 maze with dynamic obstacles is used in the third experiment. 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.

designed four different ensemble methods which are related to ensembles used in supervised learning, although a big difference is that we have to take into account that RL agents need exploration.

Next, we present the different ensemble methods that we use for combining the RL algorithms described before. The best action according to algorithm j at time t will be denoted by a_t^j. The action selection policy of this algorithm is π_t^j. We also enumerate the set of possible actions for each state for ease of use: $A(s_t) = \{a[1], \ldots, a[m]\}$. The first two ensemble methods, MV and RV, use a Boltzmann distribution over the preference values p_t's of the ensemble for each action, which ensures exploration. The resulting ensemble policy in those cases is

$$\pi_t(s_t, a[i]) = \frac{e^{p_t(s_t,a[i])/\tau}}{\sum_k e^{p_t(s_t,a[k])/\tau}}$$

where τ is a temperature parameter, and p_t is defined for the different cases later on. The other two ensemble methods, BM and BA, work already with probabilities generated by the Boltzmann distribution over actions according to independent RL algorithms and do not use another Boltzmann distribution. Instead, they use

$$\pi_t(s_t, a[i]) = \frac{p_t(s_t, a[i])^{\frac{1}{\tau}}}{\sum_k p_t(s_t, a[k])^{\frac{1}{\tau}}}.$$

After calculating the action probabilities, the ensemble selects an action, and all algorithms learn from this selected action. Note that this is the only sensible thing to do because the effects of not-executed actions are unknown.

MV: The preference values calculated by the MV ensemble using n different RL algorithms are

$$p_t(s_t, a[i]) = \sum_{j=1}^{n} I\left(a[i], a_t^j\right)$$

where $I(x, y)$ is the indicator function that outputs 1 when $x = y$ and 0 otherwise. The most probable action is simply the action that is most often the best action according to the algorithms. This method resembles a bagging ensemble method for combining classifiers with majority voting, with the big difference that, because of exploration, we do not always select the action which is preferred by most algorithms.

RV: Let $r_t^j(a[1]), \ldots, r_t^j(a[m])$ denote the weights according to the ranks of the action selection probabilities, such that if $\pi_t^j(a[i]) \geq \pi_t^j(a[k])$, then $r_t^j(a[i]) \geq r_t^j(a[k])$. For example, the most probable action could be weighted m times, the second most probable $m-1$ times, and so on. This is the weighting that we used in our experiments. The preference values of the ensemble are

$$p_t(s_t, a[i]) = \sum_j r_t^j(a[i]).$$

BM: Another possibility is multiplying all the action selection probabilities for each action based on the policies of the algorithms. The preference values of the ensemble are

$$p_t(s_t, a[i]) = \prod_j \pi_t^j(s_t, a[i]).$$

A potential problem with this method is that one algorithm can set the preference values of any number of actions to zero when it has a zero probability of choosing those actions. Because all our algorithms use Boltzmann exploration, this was not an issue in our experiments.

BA: As a last method, we can also sum the action selection probabilities of the different algorithms. Essentially, this is a variant of RV, using $r_t^j = \pi_t^j$. The preference values of the ensemble are

$$p_t(s_t, a[i]) = \sum_j \pi_t^j(s_t, a[i]).$$

IV. Experiments

We performed experiments with five different maze tasks (one simple and four more complex problems) to compare the different ensemble methods to the individual algorithms. In the first experiment, the RL algorithms are combined with tabular representations and are compared on a small-maze task. In the second experiment, a partially observable maze is used, and neural networks are used as function approximators. In the third experiment, a dynamic maze is used, where the obstacles are not placed at fixed positions, and neural-network function approximators are used. In the fourth experiment, a dynamic maze is used, where the goal is not placed at a fixed position, and neural networks are used as function approximators. In the fifth and final experiment, a generalized maze [23] task is used, where the goal and the obstacles are not placed at fixed positions, and, again, neural networks are used as function approximators.

A. Small-Maze Experiment

The different ensemble methods, namely, MV, RV, BA, and BM, are compared to the five individual algorithms, namely, Q-learning, Sarsa, AC, QV-learning, and ACLA. We performed experiments with Sutton's Dyna maze shown in Fig. 1(a). This simple maze consists of 9×6 states, and there are four actions, i.e., 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.

TABLE I
TABULAR LEARNING RATES α/β, DISCOUNT FACTORS, AND GREEDINESS (INVERSE OF THE TEMPERATURE FOR BOLTZMANN EXPLORATION) FOR THE ALGORITHMS. THE LAST 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	α	β	γ	G	Final	Rank	Cumulative	Rank
Q	0.2	–	0.9	1	5.14 $\pm$ 0.49	8	84.9 $\pm$ 11.5	9
Sarsa	0.2	–	0.9	1	5.26 $\pm$ 0.31	3-5	90.3 $\pm$ 8.3	5-8
AC	0.1	0.2	0.95	1	5.21 $\pm$ 0.17	6-7	91.1 $\pm$ 3.3	5-7
QV	0.2	0.2	0.9	1	5.25 $\pm$ 0.25	4-5	91.4 $\pm$ 7.8	4-7
ACLA+	0.005	0.1	0.99	9	5.20 $\pm$ 0.23	6-7	90.2 $\pm$ 5.2	7-8
Majority voting	–	–	–	1.6	5.33 $\pm$ 0.12	1-2	96.7 $\pm$ 2.2	1
Rank voting	–	–	–	0.6	5.01 $\pm$ 0.36	9	92.2 $\pm$ 7.9	4-5
Boltzmann mult	–	–	–	0.2	5.34 $\pm$ 0.15	1-2	95.3 $\pm$ 3.9	2
Boltzmann add	–	–	–	1	5.28 $\pm$ 0.12	3-4	93.5 $\pm$ 1.9	3

We kept the maze small because we want to compare the results with the experiments on the more complex maze tasks, which would otherwise cost too much computational time.

Experimental Setup: The reward for arriving at the goal is 100. When 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%. These reward function and noise are used in all experiments of this paper.

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. All parameters were optimized for the single RL algorithms, where they were evaluated using the average reward intake, and the final performance was 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 each step closer to the goal, which is optimized using both criteria if the discount factor is large enough. We also optimize the discount factors because we found that they had a large influence on the results. If we would have always used the same discount factor for all algorithms, the results of some algorithms would have been much worse, and therefore, it would be impossible to select a fair discount factor. Because we evaluate all methods using the average reward criterion, the different discount factors do not influence the comparison between algorithms. The ensemble methods used the parameters of the individually optimized algorithms, so that only the ensemble's temperature had to be set.

In Table I, 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 per 2500 steps) during the entire trial lasting for 50 000 learning steps. This latter evaluation measure shows the overall performance and the learning rate with which good solutions are obtained. The ranks are computed using the Student t-test with $p = 0.05$. Note that because 500 simulations were performed, small differences may still turn out to be significant. The results show that the MV and BM ensembles outperform all other methods. To show that the chosen discount factors matter, we also experimented with Sarsa with a discount factor of 0.95 and with ACLA using a discount factor of 0.9. Using the best found other learning parameters, Sarsa's performances were 4.89 $\pm$ 1.14 and 84.7 $\pm$ 20.0 for the final and total learning performances, respectively. Using the best other learning parameters, ACLA's performances were 4.86 $\pm$ 0.86 and 82.7 $\pm$ 18.6 for the final and total learning performances, respectively. This clearly shows that care should be taken to optimize the discount factor if one wishes to compare different RL algorithms. All algorithms converge to a stable performance within 15 000 learning steps, but the best ensembles reach better performance levels and initially learn faster.

B. Partially Observable Maze

In this experiment, we use Markov localization and neural networks to solve a partially observable MDP 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 the neural network. We used 20 sigmoidal hidden neurons in our experiments. The maze is shown in Fig. 1(a), with the goal being 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 nonzero 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 a neural network representation and the Boltzmann exploration rule. For evaluation per 5000 steps, we measured the average reward intake during that period. Table II shows that the BM ensemble method learns fastest in this problem. We also experimented with a BM ensemble consisting of five

TABLE II
FINAL RESULTS (AVERAGE REWARD FOR LAST 5000 STEPS) AND CUMULATIVE RESULTS FOR A NEURAL-NETWORK REPRESENTATION ON THE PARTIALLY OBSERVABLE MAZE. RESULTS ARE AVERAGES OF 100 SIMULATIONS

Method	α	β	γ	G	Final	Rank	Cumulative	Rank
Q	0.02	–	0.95	1	9.65 ± 0.33	1-4	154.1 ± 7.3	4
Sarsa	0.02	–	0.95	1	9.41 ± 1.26	1-8	137.6 ± 21.1	5-9
AC	0.02	0.03	0.95	1	9.33 ± 0.31	4-7	159.4 ± 3.5	3
QV	0.02	0.01	0.9	1	9.59 ± 0.31	1-4	135.3 ± 12.3	6-9
ACLA+	0.035	0.005	0.99	10	8.44 ± 0.27	9	135.1 ± 3.7	6-9
Majority voting	–	–	–	1.4	9.37 ± 0.31	4-7	139.8 ± 8.2	5-6
Rank voting	–	–	–	0.8	9.30 ± 0.28	4-7	133.1 ± 13.3	6-9
Boltzmann mult	–	–	–	0.2	9.56 ± 0.32	1-4	174.6 ± 2.9	1
Boltzmann add	–	–	–	1	9.11 ± 0.31	7-8	162.2 ± 2.6	2

TABLE III
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	α	β	γ	G	Final	Rank	Cumulative	Rank
Q	0.01	–	0.95	1	6.79 ± 0.21	3	116.2 ± 2.7	3
Sarsa	0.01	–	0.95	1	6.66 ± 0.37	4-5	112.6 ± 5.9	4
AC	0.015	0.003	0.95	1	5.98 ± 0.31	8	97.7 ± 9.4	8
QV	0.01	0.01	0.9	0.4	6.27 ± 0.20	6	108.3 ± 2.7	5-7
ACLA+	0.06	0.002	0.98	6	5.39 ± 0.15	9	89.2 ± 1.3	9
Majority voting	–	–	–	2.6	6.93 ± 0.14	2	122.1 ± 1.4	2
Rank voting	–	–	–	0.8	6.59 ± 0.21	4-5	108.0 ± 2.6	5-7
Boltzmann mult	–	–	–	0.2	7.04 ± 0.13	1	125.0 ± 1.6	1
Boltzmann add	–	–	–	1	6.08 ± 0.12	7	107.7 ± 1.3	5-7

differently initialized Q-learning algorithms. The performances of this ensemble were 9.61 ± 0.31 and 153.1 ± 8.0 for the final and total learning performances, respectively. This shows that combining different RL algorithms speeds up learning performance compared to an ensemble of the best single RL algorithm. All algorithms converge to a stable performance within 60 000 learning steps, but the best ensembles reach a good performance much earlier.

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. 1(b)]. 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 four and eight obstacles generated at random positions, and it is made sure that a path to the goal exists from the fixed starting location S. Because 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 per 150 000 steps.

Table III shows the final and total performances of the different algorithms. The BM ensemble outperforms the other algorithms on this problem: It reaches the best final performance and also has the best overall learning performance. We also experimented with a BM ensemble consisting of five differently initialized Q-learning algorithms. The performances of this ensemble were 6.87 ± 0.22 and 117.0 ± 2.7 for the final and total learning performances, respectively. This shows again that combining different RL algorithms in an ensemble performs better than an ensemble consisting of the best single RL algorithm. The best ensembles reach a better performance at the end than the single RL algorithms.

D. *Solving a Maze With Dynamic Goal Positions*

In this fourth maze experiment, we use the same small maze as before [see Fig. 1(a)], where the starting position is indicated by S, but now, the goal is placed at a different 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 54×2 inputs, which indicate the positions of the agent and goal. A simulation lasts for 3 000 000 learning steps, and we measure performance per 150 000 steps. We used feedforward neural networks with 20 sigmoidal hidden units.

Table IV shows the final and total performances of the different algorithms. The MV ensemble outperforms the other algorithms on this problem: It reaches the best final performance and also has the best overall learning performance. We also experimented with an MV ensemble consisting of five differently initialized Sarsa algorithms. The performances of this ensemble were 11.02 ± 0.22 and 176.7 ± 4.9 for the final and total learning performances, respectively. This shows again that a combination of different RL algorithms in an ensemble learns faster than an ensemble consisting of the best single RL algorithm, although an ensemble of the same RL algorithm can also increase the final performance of that algorithm. The best ensembles reach a better performance at the end than the single RL algorithms and initially have a faster learning speed.

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 [23] call the "generalized maze" experiment. To deal with this, a neural-network function approximator receives the position of the agent, goal, and dynamic walls as input. Therefore, there are 54×3 inputs. A simulation lasts for 15 000 000 learning

TABLE IV
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	α	β	γ	G	Final	Rank	Cumulative	Rank
Q	0.005	–	0.95	0.5	10.05 ± 0.37	7-9	152.7 ± 8.3	7
Sarsa	0.008	–	0.95	0.6	10.69 ± 0.47	2-7	176.9 ± 8.7	4
AC	0.006	0.008	0.95	0.6	10.65 ± 0.11	3-7	180.5 ± 3.3	3
QV	0.012	0.004	0.95	0.6	10.66 ± 1.16	2-7	169.4 ± 20.2	5-6
ACLA+	0.06	0.006	0.98	10	10.11 ± 1.80	3-9	121.5 ± 25.6	8
Majority voting	–	–	–	2.4	11.06 ± 0.06	1	188.6 ± 2.1	1
Rank voting	–	–	–	1.2	10.58 ± 2.08	2-7	82.4 ± 30.8	9
Boltzmann mult	–	–	–	0.2	10.74 ± 0.06	2-5	187.8 ± 1.9	2
Boltzmann add	–	–	–	1	10.12 ± 0.09	7-9	170.7 ± 2.5	5-6

TABLE V
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 FOR THE SINGLE ALGORITHMS AND 10 SIMULATIONS FOR THE ENSEMBLES

Method	α	β	γ	G	Final	Rank	Cumulative	Rank
Q	0.003	–	0.95	0.3	6.92 ± 0.16	1	96.6 ± 1.1	3
Sarsa	0.003	–	0.92	0.3	1.06 ± 0.20	9	13.3 ± 0.9	9
AC	0.014	0.0015	0.95	0.5	5.84 ± 0.15	5	89.0 ± 1.0	4
QV	0.002	0.001	0.95	0.2	5.17 ± 0.16	7	76.6 ± 1.1	6
ACLA+	0.1	0.001	0.98	5	4.81 ± 0.12	8	56.7 ± 1.5	7
Majority voting	–	–	–	2.4	6.68 ± 0.23	2-3	102.6 ± 0.8	1
Rank voting	–	–	–	1.0	6.02 ± 0.16	4	48.6 ± 2.2	8
Boltzmann mult	–	–	–	0.2	6.54 ± 0.13	2-3	100.8 ± 1.0	2
Boltzmann add	–	–	–	1	5.65 ± 0.14	6	86.0 ± 0.8	5

steps, and we measure performance per 750 000 steps. The feedforward neural networks have 100 sigmoidal hidden units.

Table V shows the final and total performances of the different algorithms. Here, Q-learning obtains the best final results, but the MV ensemble has the best overall learning performance. It is surprising that Sarsa obtains much worse results than the other algorithms, even though we optimized all its learning parameters. We also experimented with an MV ensemble consisting of five Q-learning algorithms. The performances of this ensemble were 7.20 ± 0.16 and 103.4 ± 0.9 for the final and total learning performances, respectively; thus, this ensemble reaches the best final performance, and its learning speed is almost significantly better than the MV ensemble using different RL algorithms. This is the only experiment where an ensemble consisting of the same best single RL algorithm leads to a significantly better final performance compared with the best ensemble consisting of different single RL algorithms. These results can be explained by the fact that, in this problem, Q-learning performs much better than the other algorithms. At the end of the learning trial, Q-learning outperforms the best ensembles, although the ensembles initially have a faster learning speed.

V. DISCUSSION

From the results, it is clear that the BM and MV ensembles significantly outperform the other methods in terms of final performance. The BM ensemble significantly outperforms the other methods in terms of total learning performance, and the MV method comes as second best. The RV and BA ensembles do not outperform single RL algorithms.

The results showed that the best ensemble always learns fastest, but one may have noticed that the ensembles cost more computational time. Although this is true, many real-world scenarios, such as robotics, require the least number of experiences because the actions taken in real time require much more time than the actual calculation done by the learning algorithms. Furthermore, in all experiments, the best ensemble has a better or equal final performance compared to the best single RL algorithm. Even in the generalized maze, the ensemble consisting of five Q-learning algorithms outperforms the single Q-learning algorithm. Experiments also showed that an ensemble with different RL algorithms often outperforms an ensemble consisting of the best RL algorithm, even though some RL algorithms perform considerably worse. This is due to the fact that ensembles improve independent algorithms most if the algorithms' predictions are less correlated. We think that good ensemble algorithms outperform single RL algorithms because the ensemble can make a better tradeoff between exploration and exploitation by determining action choices based on the uncertainties of all algorithms. If the algorithms want to choose the same action, it is more likely that this action will be exploited than when algorithms disagree.

In future work, we want to focus on batch [10] and model-based RL algorithms [9], which can be very useful in reducing the number of experiences. We are also currently studying methods for learning to weigh each independent RL algorithm, which could increase the performance of the ensembles even further. Finally, we want to compare all algorithms on the partially observable generalized maze problem.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996.
[3] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, U.K., 1989.
[4] G. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge Univ., Cambridge, U.K., Tech. Rep. CUED/F-INFENG-TR 166, 1994.
[5] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems*, vol. 8, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA: MIT Press, 1996, pp. 1038–1045.

[6] M. Wiering and H. van Hasselt, "Two novel on-policy reinforcement learning algorithms based on TD(λ)-methods," in *Proc. IEEE Int. Symp. Adaptive Dyn. Program. Reinforcement Learn.*, 2007, pp. 280–287.
[7] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, vol. 12. Cambridge, MA: MIT Press, 2000, pp. 1057–1063.
[8] J. Baxter and P. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artif. Intell. Res.*, vol. 15, pp. 319–350, 2001.
[9] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Mach. Learn.*, vol. 13, no. 1, pp. 103–130, Oct. 1993.
[10] M. Riedmiller, "Neural fitted Q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. 16th ECML*, 2005, pp. 317–328.
[11] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.
[12] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 148–156.
[13] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, 1991.
[14] S. P. Singh, "The efficient learning of multiple task sequences," in *Advances in Neural Information Processing Systems*, vol. 4, J. Moody, S. Hanson, and R. Lippman, Eds. San Mateo, CA: Morgan Kaufmann, 1992, pp. 251–258.
[15] C. Tham, "Reinforcement learning of multiple tasks using a hierarchical CMAC architecture," *Robot. Auton. Syst.*, vol. 15, no. 4, pp. 247–274, 1995.
[16] R. Sun and T. Peterson, "Multi-agent reinforcement learning: Weighting and partitioning," *Neural Netw.*, vol. 12, no. 4/5, pp. 727–753, Jun. 1999.
[17] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, Dec. 2005.
[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 279–292, 1992.
[19] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988.
[20] K. S. Narendra and M. A. L. Thathatchar, "Learning automata—A survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-4, no. 4, pp. 323–334, Jul. 1974.
[21] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems*, vol. 7, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 369–376.
[22] G. Gordon, "Stable function approximation in dynamic programming," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-95-103, 1995.
[23] P. Werbos and X. Pang, "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1996, vol. 3, pp. 1764–1769.

Marco A. Wiering received the Ph.D. degree in reinforcement learning (RL) in 1999 from the University of Amsterdam, Amsterdam, The Netherlands.

From January 2000 to September 2007, he was an Assistant Professor with Utrecht University, Utrecht, The Netherlands. He is currently working toward a tenure track in the Department of Artificial Intelligence, University of Groningen, Groningen, The Netherlands, in the field of cognitive robotics. His research interests include machine learning, particularly RL, robotics, and machine vision.

Hado van Hasselt received the M.S. degree in cognitive artificial intelligence from the Utrecht University, Utrecht, The Netherlands, in 2006, where he is currently working toward the Ph.D. degree in the Intelligent Systems Group, Department of Information and Computing Sciences.

His research interests include machine learning, reinforcement learning (RL), and handwritten-text recognition.