Convergence and Divergence in Standard and Averaging Reinforcement Learning

Marco A. Wiering

Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University
marco@cs.uu.nl **

Abstract. Although tabular reinforcement learning (RL) methods have been proved to converge to an optimal policy, the combination of particular conventional reinforcement learning techniques with function approximators can lead to divergence. In this paper we show why off-policy RL methods combined with linear function approximators can lead to divergence. Furthermore, we analyze two different types of updates; standard and averaging RL updates. Although averaging RL will not diverge, we show that they can converge to wrong value functions. In our experiments we compare standard to averaging value iteration (VI) with CMACs and the results show that for small values of the discount factor averaging VI works better, whereas for large values of the discount factor standard VI performs better, although it does not always converge.

1 Introduction

Reinforcement learning algorithms [13, 7] are very suitable for learning to control an agent by letting it interact with an environment. In case of tabular representations of the state-action space, convergence with probability 1 to an optimal policy has been proved for different reinforcement learning (RL) algorithms [18, 6, 16, 11]. However, in case of very large or continuous spaces, a tabular representation of the value function is not suitable. Therefore, instead of tabular representations, function approximators (FA) for representing the value function are necessary. In particular cases the combination of function approximators (e.g. neural networks) with reinforcement learning has been very successful [15]. However, in some cases divergence of particular combinations of RL algorithms with function approximators has been demonstrated [4, 5, 2].

Although some light has been shed on this problem of divergence, the cause of this divergence has not yet been demonstrated in an illustrative way. In this paper we use a very illustrative example to show why off-policy RL methods (e.g., on-line value iteration and Q-learning [17]) combined with function approximators may lead to divergence if parameters are shared between different

** In Proceedings of the 15th European Conference on Machine Learning (ECML'04), edited by J-F Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, Springer-Verlag Berlin Heidelberg, pages: 477-488 states and sufficient high exploration is used. The reason of this is that the agent can stay in the same state for a while due to exploration, while continuously updating the value function on the largest action value that also increases due to the update.

There are in principle two types of RL methods when combined with function approximators; (1) Standard RL updating, and (2) Averaging RL updating. In standard RL, parameters of the function approximator are updated using the difference between the reward plus the next state's value and the Q-value of the current state-action pair that is computed using all parameters. In averaging RL, parameters are updated using the difference between the reward plus the next state's value and the parameter value. Although averaging RL has stronger convergence properties [5, 9] than standard RL, we show that averaging RL and standard RL are very much related, but that co-operative parameter learning only is the case for standard RL and that averaging RL can converge to non-optimal value functions.

In our experiments, we compare averaging to standard dynamic programming (DP) [1], and show that the obtained value function can be much worse for using averaging DP compared to standard DP for very large values of the discount factor, whereas the obtained value function and policy is better for averaging DP for smaller values of the discount factor. This leaves us with the question which RL method would be most appropriate; one which always converges, but to a possibly bad value function approximation if the discount factor is large, or one that does not have to converge, but that sometimes converges to a better approximation of the optimal value function.

Outline of this paper. In Section 2 we will describe reinforcement learning algorithms and in Section 3 we present the combination of standard and averaging reinforcement learning with linear function approximators. In Section 4 we will show our example for which standard off-policy RL algorithms with linear function approximators will diverge. In Section 5 we will compare averaging dynamic programming to standard dynamic programming with CMACs on some maze problems. Finally, Section 6 concludes this paper.

2 Reinforcement Learning

Reinforcement learning algorithms are able to let an agent learn from its experiences generated by its interaction with an environment. We assume an underlying Markov decision process (MDP) which does not have to be known to the agent. A finite MDP is defined as; (1) The state-space $S = \{s^1, s^2, \dots, s^n\}$, where $s_t \in S$ 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 by the agent at time t; (3) A transition function $P(s, a, s')$ mapping state-action pairs s, a to a probability distribution over successor states s'; (4) A reward function

[1] As DP method we use value iteration which can be seen as an off-policy DP algorithm in contrast to policy iteration which is on-policy. When we mention DP in this paper we restrict our conclusions to value iteration.

$R(s, a, s')$ 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; (5) A discount factor $0 \leq \gamma \leq 1$ which discounts later rewards compared to immediate rewards.

2.1 Value Functions and Dynamic Programming

In optimal control or reinforcement learning, we are interested in computing or learning the optimal policy for mapping states to actions. We denote the optimal deterministic policy as $\pi^*(s) \rightarrow a^*|s$. It is well known that for each MDP, one or more optimal deterministic policies exist. The optimal policy is defined as the policy which receives the highest possible cumulative discounted rewards in its future from all states. In order to learn the optimal policy, value-function based reinforcement learning [13] uses value functions to summarize the results of experiences generated by the agent in the past. We denote the value of a state $V^\pi(s)$ as the expected cumulative discounted future reward when the agent starts in state s and follows a particular policy π:

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

The optimal policy is the one which has the largest state-value in all states: $\pi^* = \arg\max_\pi V^\pi$.

In most cases reinforcement learning algorithms used for learning to control an agent also make use of a Q-function for evaluating state-action pairs. Here $Q^\pi(s, a)$ is defined as the expected cumulative discounted future reward if the agent is in state s, executes action a, and follows policy π afterwards:

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

It is easy to see that if the optimal Q-function, Q^* is known, that the agent can select optimal actions by selecting the action with the largest value in a state: $\pi^*(s) = \arg\max_a Q^*(s, a)$. and furthermore the optimal value of a state should correspond to the highest action value in that state according to the optimal Q-function: $V^*(s) = \max_a Q^*(s, a)$.

It is also well-known that there exists a recursive equation known as the Bellman optimality equation [3] which relates a state-action value of the optimal value function to other optimal state-values which can be reached from that state using a single local transition:

$$Q^*(s, a) = \sum_{s'} P(s, a, s')(R(s, a, s') + \gamma V^*(s'))$$

The Bellman equation has led to very efficient dynamic programming (DP) techniques for solving known MDPs [3, 13]. One of the most used DP algorithms is value iteration which uses the Bellman equation as an update:

$$Q^{k+1}(s,a) := \sum_{s'} P(s,a,s')(R(s,a,s') + \gamma V^k(s'))$$

Where $V^k(s) = \max_a Q^k(s,a)$. In each step the Q-function looks ahead one step, using this recursive update rule. It can be easily shown that $\lim_{k\to\infty} Q^k = Q^*$, when starting from an arbitrary Q^0 containing only finite values.

2.2 Reinforcement Learning Algorithms

Although dynamic programming algorithms can be efficiently used for computing optimal solutions for particular MDPs they have some problems for more practical applicability; (1) The MDP should be known a-priori; (2) For large state-spaces the computational time would become very large; (3) They cannot be directly used in continuous state-action spaces.

Reinforcement learning algorithms can cope with these problems; first of all the MDP does not need to be known a-priori, all that is required is that the agent is allowed to interact with an environment which can be modelled as an MDP. Secondly, for large or continuous state-spaces, a RL algorithm can be combined with a function approximator for learning the value function. When combined with a function approximator, the agent does not have to compute state-action values for all states, but can generalize from experiences and concentrate itself on parts of the state-space where the best policies lead into.

One particular algorithm for learning a Q-function is Q-learning [17, 18]. 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 $0 < \alpha \leq 1$ is the learning rate. Q-learning is an off-policy reinforcement learning algorithm [13], which means that the agent learns about the optimal value function while following another behavioral policy which usually includes exploration steps. This has as 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 [18, 6, 16]. On the other hand, Q-learning does not learn about its behavioral policy, so if the behavioral policy always receives low cumulative discounted rewards, the agent does not try to improve it.

Instead of Q-learning, the on-policy algorithm SARSA(0) for learning Q-values has been proposed in [10, 12]. SARSA(0) 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(0) converges to the optimal policy under some conditions on the learning rate after an infinite number of steps if the exploration policy is GLIE (greedy in the limit of infinite exploration), which means that the agent should always explore, but stop exploring after an infinite number of steps [11].

3 Reinforcement Learning with Function Approximation

To learn value functions for problems with many state variables, there is the curse of dimensionality; the number of states increases exponentially with the number of state variables, so that a tabular representation would quickly become infeasible in terms of storage space and computational time. Also for a problem having continuous states, a tabular representation requires a good discretization which has to be done a-priori using knowledge of the system, and a fine-grained discretization will also quickly lead to a large number of states. Therefore, instead of using tabular representations it is more appropriate to use function approximators to deal with large or continuous state spaces.

In this paper we will concentrate ourselves on linear function approximators where the state-vector s_t which is received by the agent at time t is mapped upon a feature-vector $\boldsymbol{\phi}(s_t)$. Then we use a parametrized function approximator where the set of parameters is a vector $\boldsymbol{\theta}$. The value of a state-action pair is:

$$Q(s,a) = \sum_i \theta_{i,a}\phi_i(s)$$

Well-known function approximators (FA) of this type are CMACs, locally weighted learning, radial basis networks, and linear neural networks, but not multi-layer perceptrons since these change the mapping from input-space to feature-space.

Using reinforcement learning techniques, we can learn the parameter vector $\boldsymbol{\theta}$ which makes up the value function. There are two different types of RL updates; (1) Standard RL updating; (2) Averaging RL updating.

3.1 Standard RL with Linear Function Approximators

In case we use standard RL updating, we get the following algorithms:
Standard Q-learning. For Q-learning with linear function approximators, we use the following update after an experience (s_t, a_t, r_t, s_{t+1}) for all i:

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

Q-learning using this update-rule may diverge to infinite values as we show in the following section, therefore no general convergence proofs can be obtained using standard Q-learning with linear function approximators.
Standard SARSA(0). For SARSA(0) with linear function approximators, we use the following update after an experience $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ for all i:

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

SARSA(0) is an on-policy RL method, and Gordon (2002) proved that using this update for a fixed policy the parameters will converge, and for a changing policy the parameters will stay fixed in a bounded region. Furthermore Perkins and Precup (2002) proved that the parameters using SARSA(0) will converge if the policy improvement operator produces ϵ-soft policies and is Lipschitz continuous in the action values with a constant that is not too large.

3.2 Averaging RL with Linear Function Approximators

In averaging RL updates do not use the difference between two successor state-values, but between the successor state-value and the parameter's value itself. As Reynolds (2002) has shown, averaging RL will not diverge if the feature-vector for each state is normalized, but the value functions will remain in some region. Reynolds did not prove convergence with averaging RL, but Szepesvari and Smart (2004) have proved convergence of averaging Q-learning with interpolative function approximators, although the proof requires that the sampling distribution is fixed and follows the stationary distribution. In the following we assume normalized feature vectors, so we will use $\phi'_i(s)$ as the normalized feature-vector computed as: $\phi'_i(s) = \frac{\phi_i(s)}{\sum_j \phi_j(s)}$, where we assume all $\phi_i(s)$ are positive. Furthermore we define $Q(s, a)$ in terms of $\phi'(s)$.
Q-learning. The averaging Q-learning update looks as follows for all i:

$$\theta_{i,a_t} := \theta_{i,a_t} + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - \theta_{i,a_t})\phi'_i(s_t)$$

The θ parameters learn towards the desired value and are not cooperating in the learning updates. The use of this averaging Q-learning rule will not lead to divergence [9]. Furthermore, under a fixed stationary distribution with interpolative function approximators, averaging Q-learning has been proved to converge [14]. The requirement that the function approximator is interpolative is quite strict, however, since locally weighted learning and CMACs do not fall in this class of function approximators.
SARSA(0). For SARSA(0), an averaging RL update looks as follows for all i:

$$\theta_{i,a_t} := \theta_{i,a_t} + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - \theta_{i,a_t})\phi'_i(s_t)$$

So, what can we expect when we use averaging RL instead of standard RL? Their convergence properties seem stronger as averaging RL does at least not lead to divergence for all different RL algorithms, but averaging RL can learn wrong value functions, as we will show in the next subsection.

3.3 Sample-based Dynamic Programming

In case we have a model of the environment, we can also use sample-based Dynamic Programming as was proposed by Boyan and Moore (1995) and Gordon (1995). Sample-based value iteration goes as follows. First we select a subset of states $S_0 \subset S$. Often S_0 is chosen arbitrarily, but in general S_0 should be large enough and be spread over the state space. Then we are going to use the states $s \in S_0$ for updating the value function using a model of the environment. Again this can be done by averaging methods or by standard methods.
Averaging Q-value iteration with linear FA. Averaging sample-based DP uses the following update, where $T(.)$ is the backup operator:

$$T(\theta_{i,a}) = \frac{1}{\sum_{s \in S_0} \phi'_i(s)} \sum_{s \in S_0} \phi'_i(s) \sum_{s'} P(s, a, s')(R(s, a, s') + \gamma \max_b \sum_j \phi'_j(s')\theta_{j,b})$$

This DP-update rule is obtained from the averaging Q-learning rule with linear function approximators (where we set the learning rate α to 1). Remember that we had: $\theta_{i,a_t} := \theta_{i,a_t} + (r_t + \gamma \max_a Q(s_{t+1}, a) - \theta_{i,a_t})\phi'_i(s_t)$. We rewrite this as:

$$\theta_{i,a_t} := \theta_{i,a_t} + (r_t + \gamma \max_a Q(s_{t+1}, a))\phi'_i(s_t) - \theta_{i,a_t}\phi'_i(s_t)$$
$$\theta_{i,a_t} := \frac{(r_t + \gamma \max_a Q(s_{t+1}, a))\phi'_i(s_t)}{\phi'_i(s_t)}$$

The Q-value iteration backup operator is obtained by averaging over all states in the sweep. Why is this averaging DP? Well, because we take the weighted average of all targets irrespective of other parameter values for the current state, so parameter estimation is done in a non co-operative way. Note that for a tabular representation, this would simply be conventional Q-value iteration. We note that averaging RL has a problem. Consider two examples $0.5 \rightarrow 1$ (which means that one feature value is 0.5 and the discounted cumulative reward is 1) and $1 \rightarrow 2$. Thus, the best value of the parameter would be 2, but averaging Q-value iteration will compute the value $\frac{5}{3}$. Note that this would also be the fixed point if averaging Q-learning is used: (1 - 5/3)*0.5 + (2 - 5/3)*1 = 0. So although it can be shown that averaging DP converges [5], averaging DP or RL does not have to converge to the best possible value function.

Standard Q-value iteration with linear FA. For standard value-iteration with linear function approximators, the resulting algorithm will look a bit non-conventional. First of all we introduce the value of a state if a particular parameter i is not used as:

$$Q_{-i}(s, a) = \sum_{j \neq i} \theta_{j,a}\phi'_j(s)$$

Now we have $Q(s, a) = Q_{-i}(s, a) + \theta_{i,a}\phi_i(s)$. Furthermore $Q_{-i}(s, a) = 0$ in case of tabular representations with only feature i active. The standard Q-value iteration algorithm that is co-operative, but may diverge, is:

$$T(\theta_{i,a}) = \frac{1}{\sum_{s \in S_0} \phi'_i(s)^2} \sum_{s \in S_0} \phi'_i(s) \sum_{s'} P(s, a, s')(R(s, a, s') + \gamma \max_b \sum_j \phi'_j(s')\theta_{j,b} - Q_{-i}(s, a))$$

This rule is obtained by closely examining the Standard Q-learning rule with linear function approximators (with $\alpha = 1$). Remember that we had:

$$\theta_{i,a} := \theta_{i,a} + (r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a))\phi'_i(s_t)$$

We can rewrite this as:

$$\theta_{i,a} := \theta_{i,a} + (r_t + \gamma \max_b Q(s_{t+1}, b) - Q_{-i}(s_t, a))\phi'_i(s_t) - \theta_{i,a}\phi'_i(s_t)^2$$
$$\theta_{i,a} := \frac{(r_t + \gamma \max_b Q(s_{t+1}, b) - Q_{-i}(s_t, a))\phi'_i(s_t)}{\phi'_i(s_t)^2}$$

The standard Q-value iteration is the result from using the backup operator where all sample states are used together with a model. If we examine the same examples: $0.5 \rightarrow 1$ and $1 \rightarrow 2$, we can see that standard DP would compute the value 2, which is correct.

In the case of a tabular representation with only one state active, the resulting algorithm is again the same algorithm as conventional Q-value iteration. The problem of this standard value-iteration algorithm with function approximators, which is also similar (although it looks quite different) to the one used by Boyan and Moore (1995), is that it may diverge, just as Q-learning may diverge.

4 Divergence of Q-learning with Function Approximators

Off-policy RL methods such as online value iteration or standard Q-learning can diverge for a large number of function approximators such as linear neural networks, locally weighted learning, and radial basis networks. We show an example which shows why divergence to infinite parameter values can be obtained by using online value iteration together with exploration. The online value iteration algorithm uses a model of the environment, but only updates on states visited by the agent. This divergence to infinite values will also happen for Q-learning by taking action values into account, but divergence will not happen for on-policy methods such as SARSA or TD-learning with standard updating, neither for averaging RL methods.

Our example is very simple; suppose there is one state s with value 0.5 or 1. The agent can select actions a_t from $\{0.1, 0.2, 0.3, \ldots, 0.9, 1.0\}$. An absorbing state has been reached if $s = 1$, else the next state s_{t+1} is influenced by the action a_t by setting $s_{t+1} = 0.5$ if $a_t < 1$ and $s_{t+1} = 1$ if $a_t = 1$. The reward on all transitions is 0, and we use a discount factor γ. The initial state $s_0 = 0.5$. We use one feature $\phi(s)$ which has the same value as the state s. It is clear that the optimal parameter $\theta^* = 0$. Suppose that we initialize θ to a positive value. Now the learning algorithm computes the following update if $s = 0.5$:

$$\theta = \theta + \alpha(\gamma\theta - 0.5\theta)0.5$$

And the following update if $s = 1$: $\theta = \theta + \alpha(0 - \theta)$

If the agent often selects random actions, in many cases the agent will be in state $s = 0.5$. Suppose it stays on average h times in state $s = 0.5$ before making a step to $s = 1$. Then it will make the following average update:

$$\theta = \theta + \alpha((\gamma\theta - 0.5\theta)0.5h - \theta)$$

This will lead to increasing values of θ if: $h > \frac{1}{0.5\gamma - 0.25}$. Thus, for large enough γ and exploration, the parameter will always increase.

If the next state and the current state share parameters, it is easy to see that if the estimated value of the current state grows, that the estimated value of the best next state grows at the same time. This may lead to divergence. If instead we use a Monte-Carlo return, and do not use bootstrapping at all, we will not have the problem. Also if we use the averaging RL update divergence is prevented. Finally, if we use on-policy algorithms such as SARSA or TD-learning divergence will not happen, because an increasing update is only made if the agent actually makes a transition to the higher valued state.

5 Experiments with Standard and Averaging DP

We executed a number of experiments to compare standard and averaging DP with CMACs [1] as a linear function approximator. For this, we always normalize the activity of all tile-cells which are active given a state so that the combined activity adds up to 1.0. We use Q-value iteration (as described in Subsection 3.3) using all states of the environment as sampling states.

The environment is a maze of size 51×51 with one goal state. Each state can be a starting state except for the goal state. There are deterministic single-step actions North, East, South, and West. The reward for every action is -1. The process stops if the agent is in the goal-location. The tilings we use divide the maze in horizontal slices and vertical slices. So there is one tile with 51 cells which denotes the X-position, and one tile with 51 cells which denotes the Y-position.

We executed 3 experiments; 1 with an empty maze, where we examine the obtained value functions of the different DP methods using CMACs. Furthermore, we examine convergence of the algorithms if we initialize the parameters to a range of larger randomly generated numbers. We also run experiments on a maze containing walls, which makes it impossible for the representation to come up with an optimal policy. Therefore we will examine the number of steps which results after the DP algorithm terminates (or after a large number of iterations in case of standard DP which does not always terminate).

5.1 Experiments on an Empty Maze

Although it is easy to come up with an optimal policy for an empty maze, it is interesting to study the resulting value function for standard and averaging RL with CMACs compared to the optimal value function. It should be clear that our representation does not always allow to represent the optimal value function for all different values of the discount factor. Therefore, we examined the cumulative absolute difference between the optimal value function and the obtained value functions of standard DP with CMACs and averaging DP with CMACs for a large range of values of the discount parameter $\gamma \in \{0, 0.05, \ldots, 0.95, 1.0\}$. The experimental results are shown in Figure 1(A), where we initialized all value-function parameters to random values between 0 and 100. Figure 1(A) shows: (1) For $\gamma = 0$, both algorithms compute the optimal value function. (2) For $\gamma = 1$, standard DP is able to compute the perfect value function, but averaging RL makes a large error. (3) For γ values in between 0.05 and 0.9 the value function obtained by averaging RL is slightly better than the one obtained with standard DP. Both algorithms always converged to a unique value function given the discount factor.

In Figure 1(B) the convergence on the empty maze with discount factor $\gamma = 1$ is shown to the unique final value function (which does not need to have 0-error compared to the optimal value function) of both algorithms. We can see that although standard DP converges faster, it sometimes increases the error, whereas averaging DP never increases the error. Although it is not shown in the figures, we note that for $\gamma = 0.8$ averaging DP converges faster. The fact that standard

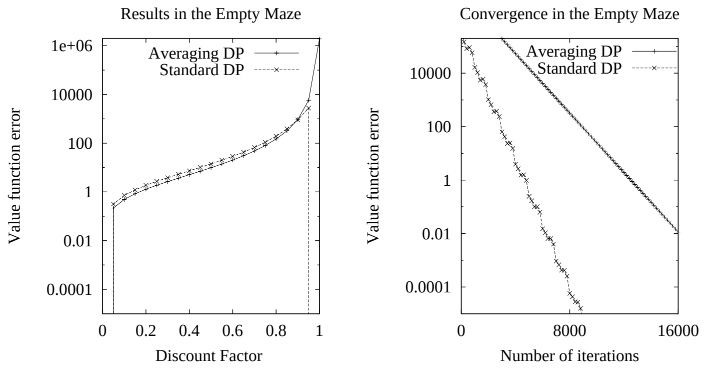

Fig. 1. *(A) The error (cumulative absolute difference between V-values computed over all states) of the value functions computed by standard and averaging DP. (B) The convergence of both algorithms when $\gamma = 1.0$. Note that standard DP sometimes increases the error, although it is hard to see.*

DP computes the optimal value function for the empty maze with $\gamma = 1$ is remarkable. It does this by computing 0, -2, -4, etc. for one tiling with distances to the goal of 0, 1, 2, etc. In this way the value of state (2,1) will be the average of -4 and -2 which is -3, which is optimal for $\gamma = 1$. For many other values of γ, standard DP cannot compute an optimal value function, and actually performs slightly worse than averaging DP. We also performed experiments with different initializations of the parameters of the value function where $\gamma = 1$. As expected averaging DP always converged to the same value function. For standard DP, initializations of 0 to 100 and 0 to 1000 led to convergence, but if we initialized the parameters to values between 0 and 10000, the parameters always diverged.

5.2 Experiments on a Maze Containing a Wall

The next experiment uses the maze shown in Figure 2(A). It is impossible for the value function obtained with CMACs and our chosen features to compute an optimal policy given all states as starting state. Instead we will examine how well the policy obtained by both algorithms after it converged (or a sufficient number of sweeps in case of standard DP) performs. The results are shown in Figure 2(B). It depicts the total number of steps the greedy policy needs to find the goal where not finding the goal is counted as 1000 steps (so that the maximum number would be $(51 \times 51 - 1) \times 1000$). We can see that averaging RL performs much better over a wide range of values for γ. Only when γ is 1.0, standard DP works better and in this case also obtains the best results. We can see that low values of γ do not harm averaging RL, but standard DP works better for $\gamma = 1$, since otherwise the subtraction of the other parameters in the standard DP update rule plays an important role. It should be said that averaging RL always converges, whereas standard DP with $\gamma = 1$ was constantly changing its

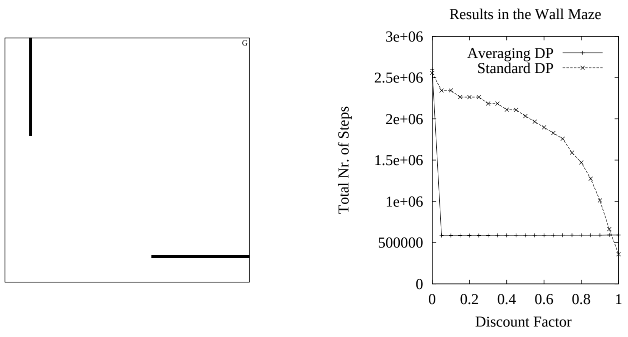

Fig. 2. *(A) The second maze containing two walls: one wall is between $x = 5$ and $x = 6$ where $y > 30$ and the other wall is between $y = 5$ and $y = 6$ where $x > 30$. (B) The number of steps in this maze with the policy computed by standard or averaging DP.*

policy, and did not converge, although the parameters did not diverge to infinite values (we used an initialization of parameter values between 0 and 100).

6 Conclusion

In this paper we have described two types of reinforcement learning algorithms when linear function approximators are used; standard RL is used in most applications of RL with function approximators [13], and averaging RL is another way which has received little attention. Since particular standard RL algorithms with function approximators may diverge, it is worth taking a look at averaging RL methods which have been proved not to diverge. We explained the reason why standard off-policy RL algorithms may diverge when combined with linear function approximators — if an agent stays in the same state due to exploration, it may increase parameters which are shared by the state and the best possible next state. This divergence will not happen for standard on-policy RL algorithms or averaging RL. Averaging sample-based dynamic programming techniques converge, although the resulting value function may not be the best possible value function, whereas standard sample-based (off-policy) dynamic programming does not have to converge. To compare standard RL to averaging RL, we executed some experiments on maze problems with Q-value iteration and CMACs as a linear function approximator. The results indicated a number of things. First averaging DP always converges (as expected) and standard DP also often converges, but can diverge if we set initial parameters to very large values. Furthermore, the resulting value function and policy of averaging DP may be better if the discount factor is not very large, whereas standard DP works best for very large values of the discount factor. In future work, we want to compare averaging and standard RL on a number of different problems. We are especially interested in comparing convergence speed of on-policy RL and averaging RL methods, since these methods seem most promising. Furthermore, we want to analyze the convergence of averaging RL methods with changing policies.

References

1. J. S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61, 1975.
2. L. Baird. Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 30–37. Morgan Kaufmann Publishers, San Francisco, CA, 1995.
3. R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.
4. 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.
5. G.J. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-95-103, Carnegie Mellon University, 1995.
6. T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.
7. L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
8. T.J. Perkins and D. Precup. A convergent form of approximate policy iteration. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13.* MIT Press, 2002.
9. S.I. Reynolds. The stability of general discounted reinforcement learning with linear function approximation. In *Proceedings of the UK Workshop on Computational Intelligence (UKCI-02)*, pages 139–146, 2002.
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. 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.
12. 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.
13. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT press, Cambridge MA, A Bradford Book, 1998.
14. C. Szepesvari and W.D. Smart. Convergent value function approximation methods. 2004. Accepted in the International Conference om Machine Learning (ICML'04).
15. G.J. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38:58–68, 1995.
16. J. N Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16:185–202, 1994.
17. C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England, 1989.
18. C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.