As appears in the Proceedings of the *Twelth National Conference on Artificial Intelligence*, pp. 202-207, 1994.

Reinforcement Learning Algorithms for Average-Payoff Markovian Decision Processes

Satinder P. Singh
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139
singh@psyche.mit.edu

Abstract

Reinforcement learning (RL) has become a central paradigm for solving learning-control problems in robotics and artificial intelligence. RL researchers have focussed almost exclusively on problems where the controller has to maximize the *discounted* sum of payoffs. However, as emphasized by Schwartz (1993), in many problems, e.g., those for which the optimal behavior is a limit cycle, it is more natural and computationally advantageous to formulate tasks so that the controller's objective is to maximize the average payoff received per time step. In this paper I derive *new average-payoff* RL algorithms as stochastic approximation methods for solving the system of equations associated with the *policy evaluation* and *optimal control* questions in average-payoff RL tasks. These algorithms are analogous to the popular TD and Q-learning algorithms already developed for the discounted-payoff case. One of the algorithms derived here is a significant variation of Schwartz's R-learning algorithm. Preliminary empirical results are presented to validate these new algorithms.

Introduction

Reinforcement learning has become a central paradigm for solving problems involving agents controlling external environments by executing actions. Previous work on reinforcement learning (e.g., Barto, Bradtke, & Singh to appear) (RL) has focused almost exclusively on developing algorithms for maximizing the discounted sum of payoffs received by the agent. Discounting future payoffs makes perfect sense in some applications, e.g., those dealing with economics, where the distant future is indeed less important than the near future, which in turn is less important than the immediate present. As recently noted by Schwartz (1993), in many other applications, however, all time periods are equally important, e.g., foraging, queuing theory problems, and problems where the optimal trajectory is a limit cycle. A natural measure of performance in such *undiscounted* applications is the *average payoff per time step* received by the agent (e.g., Bertsekas 1987). For problems where either the discounted-payoff or the average-payoff formulations can be used, often there are strong computational reasons to prefer the average-payoff formulation (see Schwartz 1993, for a recent discussion).

Recently, Jaakkola, Jordan, & Singh (to appear) have developed a fairly complete mathematical understanding of discounted-payoff RL algorithms as stochastic approximation methods for solving the system of Bellman (1957) equations associated with discounted-payoff Markovian decision processes (MDPs) (also see Tsitsiklis 1993). In this paper, I develop average-payoff RL algorithms by deriving stochastic approximation methods for solving the analogous Bellman equations for MDPs in which the measure to be optimized is the average payoff per time step. These algorithms for the average-payoff case are analogous to the popular temporal differences (Barto, Sutton, & Anderson 1983; Sutton 1988) (TD) and Q-learning (Watkins 1989; Watkins & Dayan 1992) algorithms for the discounted-payoff case. One of the four algorithms derived here using the formal stochastic approximation perspective is a significant variation of the R-learning algorithm developed recently by Schwartz (1993), who initiated the interest in average-payoff RL, but whose derivation was more heuristic. I also present preliminary empirical results on a test set of artificial MDPs.

Average-Payoff Reinforcement Learning

A large variety of sequential embedded-agent tasks of interest to AI researchers can be formulated as MDPs which are a class of discrete-time optimal control tasks. At each time step the agent senses the state of the environment, executes an action, and receives a payoff in return. The action executed by the agent along with some unmodeled disturbances, or noise, stochastically determine the state of the environment at the next time step. The actions the agent executes constitute its control policy. The task for the learning agent is to determine a control policy that maximizes some predefined cumulative measure of the payoffs received by the agent over a given time horizon.

Notation: Let S be the set of states, let $P_{xy}(a)$ denote the probability of transition from state x to state y

on executing action a, and let $R(x,a)$ be the payoff received on executing action a in state x. Further, let $A(x)$ be the set of actions available in state x, and let x_t, a_t, and R_t represent the state, the action taken, and the payoff at time step t. A stationary closed-loop control policy $\pi : S \rightarrow A$ assigns an action to each state. For MDPs there always exists an optimal stationary deterministic policy and therefore one only needs to consider such policies.

In discounted-payoff MDPs the *return* for, or *value* of a fixed policy π when the starting state of the environment is x is as follows: $V^\pi(x) = E^\pi\{\sum_{t=0}^{\infty} \gamma^t R_t | x_0 = x\}$, where $0 \leq \gamma < 1$ is a discount factor, and E^π is the expectation symbol under the assumption that policy π is executed forever. In average-payoff MDPs the average payoff per time step for a fixed policy π when the starting state of the environment is x is as follows: $\rho^\pi(x) = \lim_{N \rightarrow \infty} E^\pi\{\frac{\sum_{t=0}^{N} R_t}{N}\}$. Bertsekas (1987) shows that $\rho^\pi(x) = \rho^\pi(y)$ for all $x, y \in S$ under the assumption that the Markov chain for policy π is ergodic.

Assumption: For the rest of this paper, I am going to assume, just as in the classical average-payoff (AP) dynamic programming (DP) literature, that the MDP is ergodic for all stationary policies. Under that assumption the average payoff is always independent of the start state. The average payoff for policy π will be denoted ρ^π, and the optimal average payoff will be denoted ρ^*.

The quantity $E^\pi\{\sum_{t=0}^{\infty}(R_t - \rho^\pi)|x_0 = x\}$ is called the *relative* value of state x and is denoted $V^\pi(x)$ because it plays a role analogous to the role the value function plays in discounted-payoff MDPs. It is called a relative value because

$$V^\pi(x) - V^\pi(y) = E^\pi\{\sum_{t=0}^{\infty} R_t | x_0 = x\} - E^\pi\{\sum_{t=0}^{\infty} R_t | x_0 = y\},$$

may be seen as the long-term difference in the total payoff (not average payoff per time step) due to starting at state x rather than state y.

Reinforcement Learning as stochastic approximation

RL algorithms are iterative, asynchronous. stochastic approximation algorithms that use the state transition and payoff that occur at each time step to update the estimated relative value function and the estimated average-payoff. Both RL and asynchronous (on-line) DP take the following general form:

$$V_{t+1}(x_t) = (1 - \alpha_t(x_t))V_t(x_t) + \alpha_t(x_t)(B(V_t)(x_t) - \rho_t) \quad (1)$$

where t is the time index, $\alpha_t(x_t)$ is a learning rate constant, and ρ_t is the estimated average payoff, and V_t is the estimated relative value function. The only difference between Equation 1 and the corresponding equation for the discounted-payoff case is that the average payoff, ρ_t, is subtracted out on the RHS to form the new estimate. In DP the operator $B(V)(x)$ is deterministic and involves computing the expected relative value of all one-step neighbors of state x. I will obtain RL algorithms by replacing the deterministic backup operator B in classical DP algorithms (see Equation 1) by a random operator $\mathcal{B}$ that merely samples a single next state. This is necessary in RL algorithms because the real-environment only makes a stochastic transition to a single next state, and RL algorithms do not assume an environment model that can be used to lookup all the other possible next states. The relationship to stochastic approximation is in the following fact: $E\{\mathcal{B}\} = B$ (see Singh 1993 for an explanation).

Policy Evaluation

Policy evaluation involves determining the average payoff and the relative values for a fixed policy π. Strictly speaking, policy evaluation is a prediction problem and not a RL problem. However, because many RL architectures are based on policy evaluation (e.g., Barto, Sutton, & Anderson 1983), I will first develop average-case policy evaluation algorithms. Using the Markov assumption it can be shown that ρ^π and V^π are solutions to the following systems of linear equations:

Policy evaluation equations for the average-payoff case (e.g., Bertsekas 1987)

$$\rho + V(x) = R(x, \pi(x)) + \sum_{y \in S} P_{xy}(\pi(x))V(y), \quad (2)$$

where to get a unique solution, we set $V(r) = 0$, for some arbitrarily chosen reference state $r \in S$. This is needed because there are $|S| + 1$ unknowns and only $|S|$ equations. Note, that from Equation 2, $\forall x \in S$,

$$\rho^\pi = R(x, \pi(x)) + \sum_{y \in S} P_{xy}(\pi(x))V^\pi(y) - V^\pi(x) \quad (3)$$

Define a deterministic operator: $B_\pi(V)(x) = R(x, \pi(x)) + \sum_{y \in S} P_{xy}(\pi(x))V(y)$.

Asynchronous version of Classical DP (AP) algorithm:

$$V_{t+1}(x_t) = B_\pi(V_t)(x_t) - \rho_t$$
$$\rho_{t+1} = B_\pi(V_t)(x_t) - V_t(x_t)$$

where $\forall y \neq x_t$, $V_{t+1}(y) = V_t(y)$.

Reinforcement Learning Algorithms for (AP) Policy Evaluation:

Define the random operator $\mathcal{B}_\pi(V_t)(x_t) = R(x_t, \pi(x_t)) + V_t(x_{t+1})$, where the next state, x_{t+1}, is chosen from the probability distribution $P_{x_t x_{t+1}}(\pi(x_t))$. Note that $E\{\mathcal{B}_\pi(V)\} = B_\pi(V)$.

Algorithm 1:

$$\begin{aligned} V_{t+1}(x_t) &= (1-\alpha_t(x_t))V_t(x_t) \\ &\quad +\alpha_t(x_t)(\mathcal{B}_\pi(V_t)(x_t)-\rho_t) \\ &= (1-\alpha_t(x_t))V_t(x_t) \\ &\quad +\alpha_t(x_t)(R(x_t,\pi(x_t))+V_t(x_{t+1})-\rho_t) \end{aligned}$$

where $\rho_0 = 0$, and

$$\rho_{t+1} = (1-\beta_t)\rho_t + \beta_t[\mathcal{B}_\pi(V_t)(x_t) - V_t(x_t)].$$

Algorithm 2:

$$\begin{aligned} V_{t+1}(x_t) &= (1-\alpha_t(x_t))V_t(x_t) \\ &\quad +\alpha_t(x_t)(\mathcal{B}_\pi(V_t)(x_t)-\rho_t) \\ &= (1-\alpha_t(x_t))V_t(x_t) \\ &\quad +\alpha_t(x_t)(R(x_t,\pi(x_t))+V_t(x_{t+1})-\rho_t) \\ \rho_{t+1} &= \frac{(t*\rho_t)+R(x_t,\pi(x_t))}{t+1} \end{aligned}$$

where for both Algorithms 1 and 2; $\forall y \neq x_t$, $V_{t+1}(y) = V_t(y)$, and $\forall t, V_t(r) = 0$. Note that the difference between Algorithms 1 and 2 is in the estimation of the average payoff: Algorithm 1 estimates it using Equation 3 while Algorithm 2 estimates it as the sample average of the payoffs. Algorithm 1 corresponds closely to Sutton's TD(0) algorithm for policy evaluation in discounted-payoff MDPs.

Optimal Control

In this section I present algorithms to find optimal policies, π^*, for average-payoff MDPs. As in the discounted-payoff case, we have to use the Q-notation of Watkins (1989) to develop RL algorithms for the average-payoff case optimal control question. Again, as in classical DP, we will assume that the average payoff is independent of the starting state-action pair for all stationary policies. The average payoff for the optimal policy is denoted ρ^* and the relative Q-values are denoted Q^*, and they are solutions to the following system of nonlinear equations:

Bellman equations in the Q-notation:

$$\rho^* + Q(x,a) = R(x,a) + \sum_{y\in S} P_{xy}(a)\left[\max_{a'\in A(y)} Q(y,a')\right],$$

where the optimal action in state x can be derived as follows:

$$\pi^*(x) = \text{argmax}_{a\in A(x)} Q^*(x,a).$$

In Q-notation: $\forall i \in S$, $a \in A$,

$$\begin{aligned} \rho^* &= R(i,a) + \sum_{j\in S} P_{ij}(a)\left[\max_{a'\in A(j)} Q^*(j,a')\right] \\ &\quad -Q^*(i,a) \qquad (4) \end{aligned}$$

Define the deterministic operator $B(Q)(x_t,a_t) = R(x_t,a_t) + \sum_{y\in S} P_{x_t y}(a_t)\max_{a'\in A(y)} Q(y,a')$.

Classical Asynchronous Dynamic Programming Algorithm (AP):

$$\begin{aligned} Q_{t+1}(x_t,a_t) &= B(Q_t)(x_t,a_t) - \rho_t \\ \rho_{t+1} &= B(Q_t)(x_t,a_t) - Q_t(x_t,a_t), \qquad (5) \end{aligned}$$

where $\rho_0 = 0.0$. The above equation is an asynchronous version of the synchronous algorithm developed in Bertsekas (1987). Jalali and Ferguson (1990) have developed asynchronous DP algorithms that estimate the transition probabilities on-line, but are otherwise similar to the algorithm presented in the above equation.

Reinforcement Learning for (AP) Optimal Control:

Define the random operator $\mathcal{B}(Q)(x_t,a_t) = R(x_t,a_t) + \max_{a'\in A(x_{t+1})} Q(x_{t+1},a')$, where the next state, x_{t+1}, is chosen with probability $P_{x_t x_{t+1}}(a_t)$. Note that $E\{\mathcal{B}(Q)\} = B(Q)$.

Algorithm 3. (A significant variation to Schwartz's R-learning)

$$\begin{aligned} Q_{t+1}(x_t,a_t) &= (1-\alpha_t(x_t,a_t))Q_t(x_t,a_t) \\ &\quad +\alpha_t(x_t,a_t)(\mathcal{B}(Q_t)(x_t,a_t)-\rho_t) \\ &= (1-\alpha_t(x_t,a_t))Q_t(x_t,a_t) \\ &\quad +\alpha_t(x_t,a_t)[R(x_t,a_t) \\ &\quad + \max_{a'\in A(x_{t+1})} Q_t(x_{t+1},a') - \rho_t] \\ \rho_{t+1} &= (1-\beta_t)\rho_t + \beta_t(\mathcal{B}(Q_t)(x_t,a_t) \\ &\quad -Q_t(x_t,a_t)) \qquad (6) \end{aligned}$$

The difference between Algorithm 3 and R-learning is that in R-learning the estimated average payoff is updated only when the greedy action is executed, while in Algorithm 3 the average payoff is updated with every action. This suggests that Algorithm 3 could be more efficient than R-learning since R-learning seems to waste information whenever a non-greedy action is taken, which is quite often, especially in the beginning when the agent is exploring heavily. Updating ρ with every action makes sense because the optimal average payoff, ρ^*, satisfies Equation 4 for every state-action pair, and not just for the optimal action in each state. This change from R-learning is a direct result of the systematic derivation of RL from classical DP undertaken in this paper.

A further difference resulting from the approach taken here is that, just as in classical DP (e.g., Bertsekas 1987), I am proposing that the value of an arbitrarily chosen reference state-action pair be grounded to a constant value of zero — Schwartz's R-learning does not do that. A possible disadvantage of not grounding one state-action pair's Q-value to zero is that the relative Q-values could become very large.

Algorithm 4.

$$\begin{aligned} Q_{t+1}(x_t,a_t) &= (1-\alpha(x_t,a_t))Q_t(x_t,a_t) \\ &\quad +\alpha(x_t,a_t)(\mathcal{B}(Q_t)(x_t,a_t)-\rho_t) \end{aligned}$$

$$= (1 - \alpha(x_t, a_t))Q_t(x_t, a_t) + \alpha(x_t, a_t)[R(x_t, a_t) + \max_{a' \in A(x_{t+1})} Q_t(x_{t+1}, a') - \rho_t].$$

Let t_g be the number of times the *greedy* action has been chosen in t time steps.
If $(t+1)_g - t_g \neq 0$

$$\rho_{t+1} = \frac{(\rho_t * t_g) + R(x_t, a_t)}{(t+1)_g}$$

else, $\rho_{t+1} = \rho_t$. Note that the only difference between Algorithms 3 and 4 is in the way the average payoff is estimated; Algorithm 3 estimates it using Equation 4 while Algorithm 4 estimates it as the sample average of the payoffs received for greedy actions. As in Q-learning it is required that the Q-value of every state-action pair is updated infinitely often.

Preliminary Empirical Results

We tested Algorithms 1 through 4 on MDPs with randomly constructed transition matrices and payoff matrix. Figures 1 and 2 show the learning curves for a 20 state and 5 action problem, and Figures 3 and 4 show the learning curves for a 100 state and 10 action problem. The x-axis of all the graphs shows the number of states visited, while the y-axis shows the total error in the relative value function relative to the correct value function (V^π in the case of policy evaluation, and Q^* for optimal control). Each graph is obtained by averaging the results of 10 different runs with different random number seeds. The simulation results presented here are preliminary and are just intended to show that on the particular problems tried by the author all the four algorithms learned good approximations to the desired relative (Q-) value functions. See Figure captions for further details about the simulations.

Conclusion

The main contribution of this work is in the use of the stochastic approximation framework to develop *new* average-payoff RL algorithms to solve the policy evaluation and the optimal control questions for Markovian decision tasks. This is of substantial interest because for many embedded-agent problems, especially those in which the optimal behavior is a limit cycle, formulating them as average-payoff MDPs has many practical advantages over formulating them as discounted-payoff MDPs. Further, this paper also relates the important work begun by Schwartz on average-payoff RL to what is already known in the discounted-payoff literature by deriving R-learning and other new algorithms in the same manner as TD and Q-learning would be derived today; and it also better relates R-learning to what is already known in the classical control literature about average-payoff DP.

It is also hoped that explicit derivation as stochastic approximation methods will allow convergence results for these algorithms, just as for TD and Q-learning.[1]

Acknowledgements

I thank Anton Schwartz and the anonymous reviewers for extensive and helpful comments. This project was supported by grant ECS-9214866 from the National Science Foundation to Prof. A. G. Barto, and by a grant from Siemens Corporation to Prof. M. I. Jordan.

References

Barto, A.G.; Sutton, R.S.; and Anderson, C.W. 1983. Neuronlike elements that can solve difficult learning control problems. *IEEE SMC* 13:835–846.

Barto, A.G.; Bradtke, S.J.; and Singh, S.P. to appear. Learning to act using real-time dynamic programming. *Artificial Intelligence.*

Bellman, R.E. 1957. *Dynamic Programming.* Princeton University Press, Princeton, NJ.

Bertsekas, D.P. 1982. Distributed dynamic programming. *IEEE Transactions on Automatic Control* 27:610–616.

Bertsekas, D.P. 1987. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ.

Jaakkola, T.; Jordan, M.I.; and Singh, S.P. to appear. Stochastic convergence of iterative DP algorithms. *Neural Computation.*

Jalali, A. and Ferguson, M. 1990. Adaptive control of markov chains with local updates. *Systems & Control Letters* 14:209–218.

Schwartz, A. 1993. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth Machine Learning Conference.*

Singh, S. P. 1993. *Learning to Solve Markovian Decision Processes.* Ph.D. Dissertation, Department of Computer Science, University of Massachusetts. also, CMPSCI Technical Report 93-77.

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

Tsitsiklis, J. 1993. Asynchronous stochastic approximation and Q-learning. Submitted.

Watkins, C.J.C.H. and Dayan, P. 1992. Q-learning. *Machine Learning* 8(3/4):279–292.

Watkins, C.J.C.H. 1989. *Learning from Delayed Rewards.* Ph.D. Dissertation, Cambridge Univ., Cambridge, England.

[1] I have recently become aware that there is a published counterexample to the convergence of the asynchronous DP algorithm given by Equation 5 (Bertsekas 1982). However, unlike Equation 5, Algorithms 3 and 4 use a relaxation process, and that difference may be crucial in allowing a convergence proof.

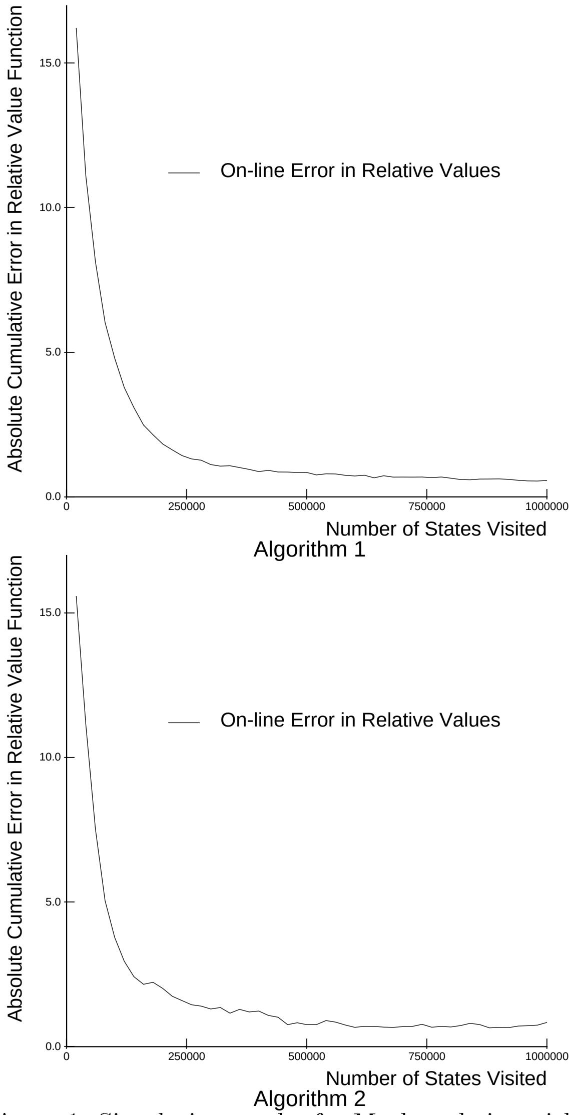

Figure 1: Simulation results for Markov chains with 20 states. The upper graph presents for Algorithm 1 the absolute error in the relative value function, summed over all the 20 states, as a function of the number of state-updates. The results presented are averages over ten Markov chains generated with different random number seeds. The transition probabilities and payoff function were chosen randomly. For each Markov chain the start state and the initial value function were chosen randomly. The bottom graph presents results averaged over the same ten Markov chains for Algorithm 2.

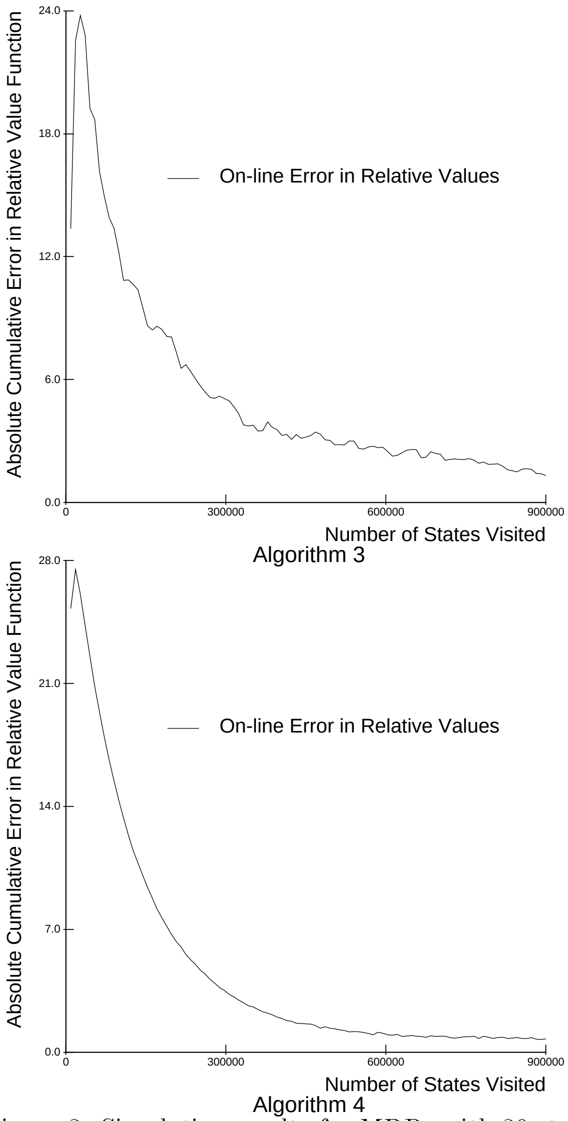

Figure 2: Simulation results for MDPs with 20 states, and 5 actions. The upper graph presents for Algorithm 3 the absolute error in the relative Q-value function, summed over all the 20 states, as a function of the number of state-updates. The results presented are averages over ten MDPs generated with different random number seeds. The transition probabilities and payoff function for the MDPs were chosen randomly. For each MDP the start state and the initial Q-value function were chosen randomly. The Boltzman distribution was used to determine the exploration strategy, i.e., $Prob(a|t) = \frac{e^{\frac{1}{T}Q_t(x_t,a)}}{\sum_{b \in A(x_t)} e^{\frac{1}{T}Q_t(x_t,b)}}$, where $Prob(a|t)$ is the probability of taking action a at time t. The temperature T was decreased slowly. The bottom graph presents results averaged over the same ten MDPs for Algorithm 4.

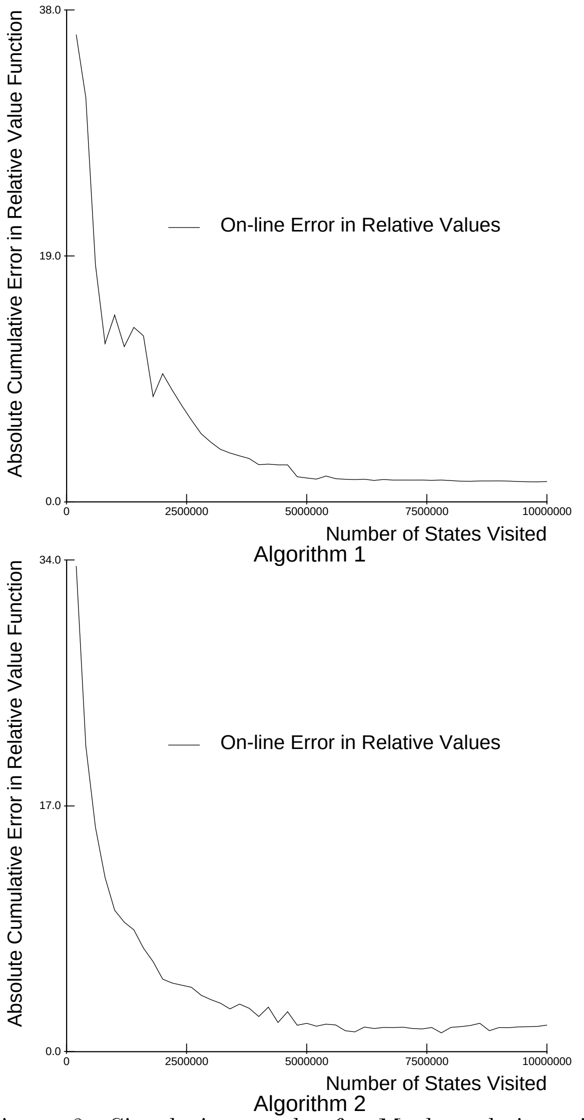

Figure 3: Simulation results for Markov chains with 100 states. The upper graph presents for Algorithm 1 the absolute error in the relative value function, summed over all the 100 states, as a function of the number of state-updates. The results presented are averages over ten Markov chains generated with different random number seeds. The transition probabilities and payoff function were chosen randomly. For each Markov chain the start state and the initial value function were chosen randomly. The bottom graph presents results averaged over the same ten Markov chains for Algorithm 2.

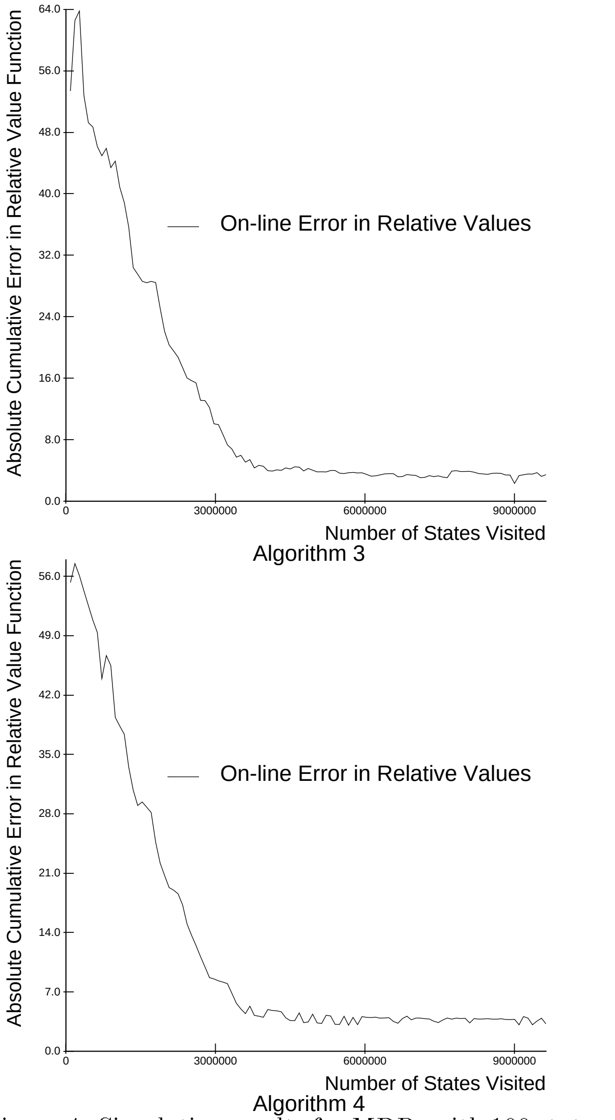

Figure 4: Simulation results for MDPs with 100 states, and 10 actions. The upper graph presents for Algorithm 3 the absolute error in the relative Q-value function, summed over all the 100 states, as a function of the number of state-updates. The results presented are averages over ten MDPs generated with different random number seeds. The transition probabilities and payoff function for the MDPs were chosen randomly. For each MDP the start state and the initial Q-value function were chosen randomly. The Boltzman distribution was used to determine the exploration strategy, i.e., $Prob(a|t) = \frac{e^{\frac{1}{T}Q_t(x_t,a)}}{\sum_{b \in A(x_t)} e^{\frac{1}{T}Q_t(x_t,b)}}$, where $Prob(a|t)$ is the probability of taking action a at time t. The temperature T was decreased slowly. The bottom graph presents results averaged over the same ten MDPs for Algorithm 4.