Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2012) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

Complex Adaptive Systems, Publication 2
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2012- Washington D.C.

Approximate Policy Iteration for Markov Control Revisited

Abhijit Gosavi*

Missouri University of Science and Technology, 219 Engineering Management Building, Rolla, MO 65409, USA

Abstract

Q-Learning is based on value iteration and remains the most popular choice for solving Markov Decision Problems (MDPs) via reinforcement learning (RL), where the goal is to bypass the transition probabilities of the MDP. Approximate policy iteration (API) is another RL technique, not as widely used as Q-Learning, based on modified policy iteration. In this paper, we present and analyze an API algorithm for discounted reward based on (i) a classical temporal differences update for policy evaluation and (ii) simulation-based mean estimation for policy improvement. Further, we analyze for convergence API algorithms based on Q-factors for (i) discounted reward and (ii) for average reward MDPs. The average reward algorithm is based on relative value iteration; we also present results from some numerical experiments with it.

Keywords: Approximate policy iteration, Q-P-Learning, average reward, relative value iteration

1. **Introduction**

Sequential decision-making problems involving stochastic discrete-event systems in which the underlying dynamic system is governed by Markov chains and the decision-maker is required to select an action (control) in a subset of states visited by the system often belong to a class of problems called Markov decision processes (MDPs). MDPs can be solved via dynamic programming (DP) methods when the state-action space is relatively small. Value iteration [1] and policy iteration [2] are two popular DP methods and have been used for many years now. More recently, Reinforcement Learning (RL) methods have emerged [3, 4, 5]. RL methods seek to use simulation (or interaction with a real system) to solve MDPs *without generating the transition probabilities* of the underlying Markov chains; determining the values of these transition probabilities can be very difficult for large-scale systems and often leads to what is called the curse of dimensionality, which plagues DP for large-scale systems. Thus, RL avoids the curse of dimensionality and has hence attracted a great deal of interest within the control community.

Q-Learning [6], based on value iteration, is one of the most popular algorithms of RL. A class of RL algorithms called Approximate Policy Iteration (API) has generated much interest recently within the RL community [7, 3]. API is rooted in the principles of policy, rather than value, iteration, but to be more precise, it is based on modified policy iteration [8]. Policy iteration has two phases: policy evaluation and policy improvement. Policy evaluation

in the classical policy iteration algorithm is performed by solving a linear equation, via linear-equation solving methods (e.g., Gaussian elimination), called the Bellman *policy* equation [2] (not to be confused with the Bellman *optimality* equation associated with value iteration). The main idea in *modified* policy iteration, and hence also in API, is to employ the policy improvement step as it is, but use value iteration for policy evaluation, instead of linear-equation solving (e.g., Gaussian elimination). In this paper, we make three contributions. Firstly, we analyze an API algorithm based on (i) the so-called TD(0) update for policy evaluation, which estimates the value function, and (ii) a simulation-based evaluation of Q-factors for policy improvement. Secondly, we analyze the convergence properties of the Q-P-Learning algorithm [5, 9] for discounted reward that bypasses the value function and estimates the Q-factors needed for policy improvement in the policy evaluation phase. Thirdly, we analyze the convergence properties of the Q-P-Learning algorithm for average reward. We also present numerical experiments with the average reward algorithm.

2. API Algorithms

API was first presented in Werbos [7], where it was discussed in the context of the value function of dynamic programming. There, it was called an "adaptive critic." These ideas have now metamorphosed into the broad umbrella of API. This is unlike Q-Learning, which estimates Q-factors (or Q-values or action values). Before presenting details of API, we present some notation that we will need throughout this paper.

- S: Set of states
- $A(i)$: Set of actions permitted in state i
- $p(i,a,j)$: transition probability of going from state i to state j under the influence of action a
- $r(i,a,j)$: immediate reward earned in transitioning from state i to state j under the influence of action a
- $h(i)$: value function of state i (generally associated with dynamic programming)
- $Q(i,a)$: Q-factor for the state-action pair (i,a)
- λ: discount factor
- $\mu(i)$: action selected in state i when policy μ is pursued

API has two main phases: policy evaluation and policy improvement. For the first algorithm we study, we use the so-called optimistic TD(0) update from [3] (pg. 229) for policy evaluation. Note that for policy evaluation, the classical version of API in [3] uses a mechanism based on either a multi-step temporal difference update or a Monte-Carlo-simulation-based update. Both of these mechanisms have a higher computational burden than TD(0). The step of policy improvement is not clearly discussed in the literature *for the scenario in which the transition probabilities are unavailable.* For instance, Bertsekas and Tsitsiklis [3] (pg. 192) discuss the notion of "simulation and averaging if necessary" to obtain Q-factors, but an explicit algorithmic scheme is not presented. Therefore, for policy improvement, we employ a simulation-based averaging scheme (based on the Robbins-Monro scheme) to obtain the Q-factors that are essential to perform policy improvement. We now present details of our API algorithm.

Algorithm 1:

Step 1: Simulate a policy μ and update its value function as follows after the transition to state j from state i:

$$h(i) \leftarrow h(i) + \alpha[r(i,\mu(i),j) + \lambda h(j) - h(i)],$$

where α is the learning rate or step size that is generally decayed to zero. In the above, usually one starts with arbitrary values for $h(i)$ for all i in S. The above step is carried out for a large number of iterations until the h-values converge.

Step 2: A fresh simulation is started in which every action is selected with the same probability in every state. Q-factors for all state-action pairs are initialized to 0 at the start. Then, when the system goes from i to j under action a, we update the Q-factor as follows:

$$Q(i,a) \leftarrow Q(i,a) + \beta\,[r(i,a,j) + \lambda h(j) - Q(i,a)],$$

where β is a learning rate like α that is gradually decayed to 0 and the function *h(.)* is fixed (already estimated in Step 2). The above step is carried out for a large number of iterations until the Q-factors converge.
Step 3: Select a new policy μ' where $\mu'(i) = argmax_{a\in A(i)}Q(i,a)$ for every *i* in *S*. If policy μ is not identical to μ', then replace μ by $\mu'$and return to the policy evaluation phase (Step 1); otherwise terminate with μ'as the optimal policy.

Optimistic versions of classical API, in which the policy evaluation phase (Step 1 above) is performed for only one state transition, are popular in the literature. Bertsekas [10] states that, in practice, optimistic API can lead to a phenomenon called *chattering* (or oscillation) in which the "improved" policy μ' is worse than μ. This can cause optimistic API to take a very long time to converge if it converges at all. As stated above, a different version of API, based on Q-factors, that avoids the value function altogether has been proposed under the name Q-P-Learning (see Gosavi [11]). This algorithm is similar to SARSA [12, 13], but differs in a crucial manner: the policy being evaluated, which is stored in the form of the P-factors in Q-P-Learning, is in SARSA an exploratory policy the exploration of which is gradually reduced. We present below a version of Q-P-Learning for discounted reward MDPs that appeared in [5, 9] without any convergence analysis.

Algorithm 2:

Step 1: Set *P(i,a)* to arbitrary values for each state-action pair *(i,a)*.
Step 2: (Policy evaluation): Choose each action with the same probability in every state. After each transition, update the Q-factors. When the system goes from *i* to *j* under action *a*, update the Q-factors as follows:

$$Q(i,a) \leftarrow Q(i,a) + \alpha\,[r(i,a,j) + \lambda\, Q\left(j, argmax_{b\in A(j)}P(j,b)\right) - Q(i,a)]$$

where α is a learning rate which is gradually decayed to 0. The above step is carried out within a simulator until the Q-factors converge.
Step 3: (Policy improvement) Set $P(i,a) = Q(i,a)$ for every state-action pair *(i,a)*. If the policy contained in the new P-factors is different than that in the old P-factors, return to Step 2; otherwise go to Step 4.
Step 4: (Termination) Compute for every *i* in *S*: $d(i) \in argmax_{a\in A(i)}P(i,a)$, and declare *d* to be the optimal policy.

Note that the above algorithm does not estimate the *h*-values, and its policy evaluation step does not contain another long simulation (remember that estimating the Q-factors or h-values requires long simulations); in other words, one iteration of the algorithm contains only one long simulation, unlike API above that requires two long simulations per iteration (one for the h-values and one for the Q-values). We will show the convergence of this algorithm subsequently.
We now discuss the average reward case in which one is interested in maximizing the average reward per time step. The steps in the associated Q-P-Learning algorithm (Gosavi, 2003, 2009), which has also appeared without any convergence analysis, are as follows.

Algorithm 3:

Step 1: Set *P(i,a)* to arbitrary values for each state-action pair *(i,a)*. Select any state-action pair to be the distinguished state-action pair (i^*, a^*).
Step 2: (Policy evaluation): Choose each action with the same probability in every state. After each transition update the Q-factors. When the system goes from *i* to *j* under action *a*, update the Q-factors as follows:

$$Q(i,a) \leftarrow Q(i,a) + \alpha\,\left[r(i,a,j) + \lambda\, Q\left(j, argmax_{b\in A(j)}P(j,b)\right) - Q(i^*,a^*) - Q(i,a)\right]$$

where α is a learning rate gradually decayed to 0. The above step is carried out within a simulator until the

Q-factors converge.
Step 3: (Policy improvement) Set $P(i,a) = Q(i,a)$ for every state-action pair *(i,a)*. If the policy contained in the new P-factors is different than that in the old P-factors, return to Step 2; otherwise go to Step 4.
Step 4: (Termination) Compute for every *i* in *S*: $d(i) \in argmax_{a \in A(i)} P(i,a)$, and declare *d* to be the optimal policy.

3. Convergence Properties

We now present the main ideas underlying the proofs of convergence of all three algorithms discussed above.
Algorithm 1: The analysis is based on the standard convergence theorem that exploits the ordinary differential equation (ODE) underlying the iterates. We first define a transformation underlying Step 1. For any *i*,

$$Th(i) = \Sigma_{j \in S} p(i,a,j)[r(i,a,j) + \lambda h(j)].$$

It is easy to show that the transformation T(.) is contractive and hence must have a unique fixed point. Further, it can be shown [14] that underlying the iterates in Step 1, there exists a continuous time process $h'(t)$. The behavior of the iterates can be studied via the behavior of the continuous-time process. Associated with $h'(t)$, one can show that there exists the following ODE:

$$\frac{dh'(t)}{dt} = T\big(h'(t)\big) - h'(t).$$

For all our algorithms, we will assume that the algorithm and its step sizes follow the asynchronous conditions specified in [15] (pg. 842) and also the condition in Equation (7.1.3) from [14].

Lemma 1. The iterates of Algorithm 1 converge with probability 1 to the unique fixed point of the transformation T(.) above.
Proof. T(.) is contractive, which implies from [14] (Theorem 7; Chap. 3) that the iterates in Step 1 will remain bounded. The result then follows directly from [14] (Theorem 2; Chap. 7). QED.

Theorem 1. Algorithm 1 converges almost surely to the optimal policy.
Proof. Using Prop. 4.1 in [5], it is easy to show that the Q-factors in Step 2 will converge with probability 1 to the Q-factors for the policy μ being evaluated. Then, from the policy improvement steps, the algorithm will mimic classical policy iteration and must converge to the optimal policy in the limit. QED.

Algorithm 2. We first define a transformation underlying Step 2.

$$T'Q(i,a) = \Sigma_{j \in S} p(i,a,j)\left[r(i,a,j) + \lambda Q\left(j, argmax_{b \in A(j)} P(j,b)\right)\right].$$

Like in the previous algorithm, it is easy to show that the above transformation is contractive, and hence it must have a unique fixed point.
Theorem 2. The policy generated by the policy improvement step (Step 3) of Algorithm 2 will converge to the optimal policy with probability 1.
Proof. We can use arguments very similar to those in Lemma 1 above to show that the iterates in Step 2 of the algorithm will converge to the fixed point of transformation T'almost surely, i.e., to the Q-factors of the policy being evaluated. Then, from the policy improvement steps, the algorithm will mimic classical policy iteration and hence must converge to the optimal policy in the limit. QED.

Algorithm 3. We now define a transformation underlying Step 2.

$$T''Q(i,a) = \Sigma_{j \in S} p(i,a,j)\left[r(i,a,j) + Q\left(j, argmax_{b \in A(j)} P(j,b)\right) - Q(\mathrm{i}^*, \mathrm{a}^*)\right].$$

It can be shown that there exists a unique solution [16] to the following equation:

$$T''Q(i,a) = Q(i,a)$$

for every state-action pair *(i,a)*. Let that solution be denoted by $Q^*(i,a)$. Further, it can be shown [14] that underlying the Q-factor iterates in Step 2, there exists a continuous time process *q(t)*. The behavior of the iterates can be studied via the behavior of the continuous-time process. Associated with *q(t)*, one can show that there exists the following ODE:

$$\frac{dq(t)}{dt} = T''\big(q(t)\big) - q(t).$$

The following result establishes an important property of the solution.

Lemma 2. Q^*, the unique solution of the equation, $T''Q(i,a) = Q(i,a)$ for all *(i,a)* pairs, is the unique globally asymptotically stable equilibrium point of the ODE above.

Proof. The proof follows from an analysis very similar to that in Theorem 3.4 in [16]. Q.E.D.

Lemma 3. The iterates in Step 2 of Algorithm 3 converge almost surely to Q^*.

Proof. From Lemma 2, we have the existence of the unique globally asymptotically stable equilibrium of the ODE. This implies boundedness and hence convergence of the iterates, almost surely, to Q^*, as argued in Lemma 1. Q.E.D.

Theorem 3. The policy generated by the policy improvement step (Step 3) of Algorithm 2 will converge to the optimal policy almost surely.

Proof. Using Lemma 3, like in classical policy iteration, one can show that the policy improvement steps will converge to the optimal policy. QED.

4. Numerical Results

We now present results from numerical experiments with Algorithm 3. Our tests are conducted on a small MDP with two states and two actions allowed in each state. We present the details for the baseline MDP (Case 1) first: r(1,1,1) = 6; r(1,1,2) = -5; r(2,1,1) = 7 ; r(2,1,2) = 12; r(1,2,1) = 10; r(1,2,2) = 17; r(2,2,1) = -14; r(2,2,2) = 13; p(1,1,1) = 0.7; p(2,1,1) = 0.4; p(1,2,1) = 0.9; p(2,2,1) = 0.2. We study three other MDPs for which the parameters will be the same as those for Case 1 with the following differences. Case 2: r(1, 1, 2) = 5 and r(2, 2, 1) = 14; Case 3: r(1, 2, 2) = 42; Case 4: r(2, 1, 2) = 25.

Results are presented in Table 1. The optimal policy is denoted by (μ*(1), μ*(2)). The table also shows the the optimal average reward, ρ*, and the Q-factors obtained at the end. We selected (i*,a*)= (1,1). Hence Q(1,1) estimates ρ*. We used a step size of α =100 / (1000+*k*), where *k* denotes the number of iterations of learning (or one-step simulation). Further, each policy was evaluated for 1000 iterations (state transition). In each of the four cases, the algorithm generated an optimal solution after M policy evaluations. However, using fewer than 1000 (approximately) iterations per policy evaluation led to the chattering reported in [10]. Note that the algorithm requires 1000M iterations of one-step simulation, which appears to be a significant computational burden for a problem with two states and two actions.

Table 1. Numerical results from Algorithm 3

Case #	μ*	ρ*	M	Q(1,1)	Q(1,2)	Q(2,1)	Q(2,2)
1	(2,1)	10.56	2	11.52	19.23	15.72	14.01
2	(2,2)	11.53	2	11.23	14.98	19.28	23.28
3	(2,1)	12.56	2	13.98	24.96	13.42	9.52
4	(2,1)	11.16	2	12.40	16.54	36.67	31.35

5. Conclusions

We presented a new API algorithm for discounted reward along with its convergence analysis and analyzed for convergence two existing API algorithms, one for average reward and the other for discounted reward, from the literature. We also presented numerical results with the average reward algorithm. In future work, we intend to test these algorithms on large-scale problems.

References

1. R. Bellman (1954). The theory of dynamic programming. *Bull. Amer. Math. Soc.* 60, 503–516.
2. R. Howard (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.
3. D.P. Bertsekas, J. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Athena Scientific, Nashua, NH.
4. R. Sutton, A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
5. A. Gosavi (2003). *Simulation-Based Optimization_ Parametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic Publishers, Boston.
6. C.J. Watkins (1989). Learning from delayed rewards. Ph.D. thesis, Kings College, Cambridge, UK.
7. P. J. Werbös (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Trans. Systems, Man, Cybernetics,* 17, 7–20.
8. J. A. E. E. van Nunen (1976). A set of successive approximation methods for discounted Markovian decision problems. *Z. Oper. Res.* 20 203–208.
9. A. Gosavi (2009). Reinforcement Learning: A Tutorial Survey, *INFORMS Journal on Computing*, 21(2), 178-192.
10. D.P. Bertsekas (2011). Approximate Policy Iteration: A Survey and Some New Methods. *Journal of Control Theory and Applications*, 9(3), 310-335.
11. A. Gosavi (2004). A reinforcement learning algorithm based on policy iteration for average reward: Empirical results with yield management and convergence analysis. *Machine Learning,* 55, 5–29.
12. G.A. Rummery, M. Niranjan (1994). On-line *Q*-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, Cambridge, UK.
13. R. Sutton (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Neural Information Processing Systems*, Vol. 8. MIT Press, Cambridge, MA, 1038–1044.
14. V.S. Borkar (2008). Stochastic Approximation: A Dynamical Systems Viewpoint, Cambridge Univ. Press.
15. V.S. Borkar (1998). Asynchronous stochastic approximation. *SIAM J. Control Optim.* 36, 840–851.
16. J. Abounadi, D. P. Bertsekas, V. S. Borkar (2001). Learning algorithms for Markov decision processes with average cost. *SIAM J. Control Optim.* 40, 681–698.