MATHEMATICS OF OPERATIONS RESEARCH
Vol. 37, No. 1, February 2012, pp. 66–94
ISSN 0364-765X (print) | ISSN 1526-5471 (online)

http://dx.doi.org/10.1287/moor.1110.0532
© 2012 INFORMS

Q-Learning and Enhanced Policy Iteration in Discounted Dynamic Programming

Dimitri P. Bertsekas, Huizhen Yu
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
{dimitrib@mit.edu, janey_yu@mit.edu}

We consider the classical finite-state discounted Markovian decision problem, and we introduce a new policy iteration-like algorithm for finding the optimal state costs or Q-factors. The main difference is in the policy evaluation phase: instead of solving a linear system of equations, our algorithm requires solving an optimal stopping problem. The solution of this problem may be inexact, with a finite number of value iterations, in the spirit of modified policy iteration. The stopping problem structure is incorporated into the standard Q-learning algorithm to obtain a new method that is intermediate between policy iteration and Q-learning/value iteration. Thanks to its special contraction properties, our method overcomes some of the traditional convergence difficulties of modified policy iteration and admits asynchronous deterministic and stochastic iterative implementations, with lower overhead and/or more reliable convergence over existing Q-learning schemes. Furthermore, for large-scale problems, where linear basis function approximations and simulation-based temporal difference implementations are used, our algorithm addresses effectively the inherent difficulties of approximate policy iteration due to inadequate exploration of the state and control spaces.

Key words: Markov decision processes; Q-learning; policy iteration; value iteration; stochastic approximation; dynamic programming; reinforcement learning
MSC2000 subject classification: Primary: 90C40, 93E20, 90C39; secondary: 68W15, 62L20
OR/MS subject classification: Primary: dynamic programming/optimal control, analysis of algorithms; secondary: Markov, finite state
History: Received October 14, 2010; revised May 9, 2011. Published online in *Articles in Advance* January 13, 2012.

1. Introduction.

We consider the discounted infinite horizon dynamic programming (DP) problem with n states, denoted $i = 1, \ldots, n$. State transitions (i, j) under control u occur at discrete times according to transition probabilities $p_{ij}(u)$ and generate a cost $\alpha^k g(i, u, j)$ at time k, where $\alpha \in (0, 1)$ is a discount factor. We consider deterministic stationary policies μ such that for each i, $\mu(i)$ is a control that belongs to a constraint set $U(i)$. We denote by $J_\mu(i)$ the total discounted expected cost of μ over an infinite number of stages starting from state i, and by $J^*(i)$ the minimal value of $J_\mu(i)$ over all μ. We denote by $Q^*(i, u)$ the optimal Q-factor corresponding to a state-control pair (i, u). It represents the optimal expected cost starting from state i, using control u at the first stage, and subsequently using an optimal policy. Optimal Q-factors and costs are related by the equation

$$J^*(i) = \min_{u \in U(i)} Q^*(i, u), \quad \forall\, i = 1, \ldots, n. \tag{1.1}$$

Similarly, we denote by $Q_\mu(i, u)$ the Q-factors of a policy μ corresponding to initial state-control pairs (i, u), (that is, the expected cost of starting with state i and control u and subsequently using policy μ); they yield the policy costs $J_\mu(i)$ via

$$J_\mu(i) = Q_\mu(i, \mu(i)), \quad \forall\, i = 1, \ldots, n. \tag{1.2}$$

We denote by J_μ, J^*, Q_μ, and Q^*, the vectors with components $J_\mu(i)$, $J^*(i)$, $i = 1, \ldots, n$, and $Q_\mu(i, u)$, $Q^*(i, u)$, $i = 1, \ldots, n$, $u \in U(i)$, respectively. This is the standard discounted Markovian decision problem (MDP) context, discussed in many sources (e.g., DP books such as those by Bertsekas [6] and Puterman [35], and approximate DP books such as those by Bertsekas and Tsitsiklis [11], Sutton and Barto [39], Cao [22], and Powell [33]).

Value iteration (VI) and policy iteration (PI) are fundamental algorithms for solving the MDP, and can be applied both in cost space to find J^* and in Q-factor space to find Q^*. Mathematically, the algorithms produce identical results, whether executed in cost space or Q-factor space, but the use of Q-factors is well suited to the "model-free" context where the transition probabilities $p_{ij}(u)$ and one-stage costs $g(i, u, j)$ may not be known explicitly, but may be computable using a simulator. In the case of Q-factors, these algorithms compute Q_μ and Q^* as the unique fixed points of the mappings F_μ and F, respectively, defined by

$$(F_\mu Q)(i, u) = \sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha Q(j, \mu(j))\big), \quad \forall\, u \in U(i), \quad i = 1, \ldots, n, \tag{1.3}$$

$$(FQ)(i, u) = \sum_{j=1}^{n} p_{ij}(u)\Big(g(i, u, j) + \alpha \min_{v \in U(j)} Q(j, v)\Big), \quad \forall\, u \in U(i), \quad i = 1, \ldots, n, \tag{1.4}$$

which are known to be sup-norm contractions. The VI algorithm is just the iteration $Q_{k+1} = FQ_k$ and converges to Q^* starting from any initial Q_0. The Q-learning algorithm of Watkins [48] is a stochastic approximation method based on this VI algorithm (see Bertsekas and Tsitsiklis [11] and Sutton and Barto [39] for descriptions and discussion, and Tsitsiklis [42] for a convergence analysis). The Q-factors Q_μ of policies μ may also be obtained with the VI algorithm $Q_{k+1} = F_\mu Q_k$ or a corresponding Q-learning/stochastic approximation method.

Let us describe the kth step of PI. It starts with the current policy μ_k and consists of two phases: the *policy evaluation* phase, which finds the fixed point Q_{μ_k} of F_{μ_k} (possibly by VI), and the *policy improvement* phase, which updates μ_k to a new policy μ_{k+1} via $\mu_{k+1}(i) = \arg\min_{u \in U(i)} Q_{\mu_k}(i, u)$ for all i. Intermediate between PI and VI is *modified PI* (also called *optimistic PI*), where policy evaluation of μ_k is performed inexactly, using a finite number m_k of VI that aim to compute Q_{μ_k}. It starts the kth step with a pair (μ_k, Q_k), it evaluates μ_k approximately, using

$$Q_{k+1} = F_{\mu_k}^{m_k} Q_k, \tag{1.5}$$

and it obtains a new policy μ_{k+1} via

$$\mu_{k+1}(i) = \arg\min_{u \in U(i)} Q_{k+1}(i, u), \qquad \forall\, i = 1, \ldots, n. \tag{1.6}$$

The method is described in detail in many sources (e.g., the textbooks by Puterman [35], Bertsekas [6]), and with judicious choice of the number m_k of VI, it is generally believed to be more efficient than either (exact) PI or VI. Its convergence for a broad class of discounted problems and a synchronous implementation has been established by Rothblum [37]. (See also the more recent work by Canbolat and Rothblum [21], which extends some of the results of Rothblum [37].) However, when modified PI is implemented asynchronously, it has some convergence difficulties and generally requires an assumption like $F_{\mu_0} Q_0 \le Q_0$ to maintain monotonic convergence, as shown through counterexamples by Williams and Baird [49]. Moreover, there are no known convergent stochastic implementations of this method that parallel the Q-learning algorithm of Watkins [48]. These difficulties are due to an inherent lack of consistency of the method: it successively applies mappings F_μ that aim at different fixed points as μ is changing irregularly, so it needs some additional convergence mechanism. This is provided by the assumption $F_{\mu_0} Q_0 \le Q_0$, which guarantees monotonic convergence ($Q_k \downarrow Q^*$), much like in standard convergence proofs of PI.

The purpose of this paper is to propose a new approach to Q-learning and modified PI. On one hand, our algorithms involve lower overhead per iteration than ordinary Q-learning, by obviating the need for minimization over all controls at every iteration (this is the generic advantage that modified PI has over VI). On the other hand, our algorithms are reliable, bypass the convergence difficulties of modified PI just mentioned, and admit asynchronous and stochastic iterative implementations with guaranteed convergence.

The novelty of our approach lies in replacing the policy evaluation phase of the classical PI method with (possibly inexact) solution of an *optimal stopping problem*. This problem is defined by a stopping cost and by a randomized policy, which are suitably adjusted at the end of each iteration. The randomized policy encodes aspects of (or as a special case, may be equal to) the "current policy" and gives our algorithm a modified PI character. To our knowledge, there have been no formulations similar to ours in the context of both exact PI and simulation-based PI, although ours is mostly motivated by the latter. Also there is very little literature on PI-like algorithms for learning Q-factors. In this connection, we note that Bhatnagar and Babu [15] have proposed Q-learning/PI-type algorithms with lookup table representation, based on two-time-scale stochastic approximation, and established their convergence for synchronous/partially asynchronous implementations. Their algorithms also have low computation overhead per iteration like our algorithms. However, viewed at the slow time scale, their algorithms are close to the standard Q-learning method and have a different basis than our algorithms.

Mathematically, our approach rests on a new idea, which is to iterate within a larger space, with a mapping that encodes the optimal stopping structure. Instead of iterating on just Q-factors Q, like the modified PI (1.5)–(1.6), we iterate on pairs of Q-factor and stopping cost vectors, Q and J. Instead of the set of mappings F_μ, we use a set of mappings that operate on pairs (J, Q), we encode the optimal stopping formulation of policy evaluation in the Q-component mappings, and we calculate new stopping cost vectors with the J-component mappings in a way that resembles the policy improvement operation. These mappings, to be introduced and analyzed in §§2–4, are sup-norm contractions that have (J^*, Q^*) as a common fixed point. As a result, our modified PI and Q-learning algorithms may change arbitrarily the policies involved, while consistently aiming at the desired fixed point (J^*, Q^*). This overcomes the convergence difficulties of the modified PI (1.5)–(1.6) in a totally asynchronous, possibly stochastic, computation environment.

In this paper, we develop our approach primarily in the context of exact DP with lookup table cost function representation. Aside from their conceptual/analytical value in small-scale problems, our methods can be applied to large-scale problems through the use of aggregation (a low-dimensional aggregate representation of a large, possibly infinite-dimensional problem; see Jaakkola et al. [28, 29], Gordon [25], Tsitsiklis and Van Roy [44], Baras and Borkar [2], Bertsekas [5, 8]). They can also be easily combined with function approximation to obtain approximate DP/compact representation methods for large-scale problems, although in that context, convergence and approximation properties of the algorithms deserve further study. Despite our emphasis on exact DP, we devote the last two sections of this paper to cost function approximation issues. We show that our approach may be combined with the simulation-based temporal difference (TD) method of Tsitsiklis and Van Roy [45], which solves optimal stopping problems with compact representation, and we also show that our approach can be used to address the difficulties due to inadequate exploration that arise in the context of PI with cost function approximation.

This paper is organized as follows. In §2, we introduce our approach to PI-like algorithms for the case of exact/lookup table representation of Q-factors. We explain the nature of our novel policy evaluation procedure that is based on solving an optimal stopping problem, and we derive some basic properties of our algorithm. In §3, we discuss the deterministic asynchronous implementations of our method and prove their convergence. These algorithms have improved convergence properties over the standard asynchronous PI algorithm for Q-factors. In §4, we develop stochastic iterative methods that are intermediate between Q-learning/value iteration and policy iteration, and we establish their convergence in a totally asynchronous computation framework. In §5, we provide some computational results and a comparison with Q-learning. In §6, we consider the possibility of approximations in our algorithms, and we derive a corresponding error bound. In §7, we discuss the exploration issue in simulation-based/model-free learning, and we also briefly describe implementations of policy evaluation with linear feature-based approximations and simulation-based optimal stopping algorithms, such as the one of Tsitsiklis and Van Roy [45]. These algorithms use calculations of low dimension (equal to the number of features) and require low overhead per iteration.

2. New policy iteration algorithms.

In this section we introduce our policy iteration-like Q-learning algorithm in synchronous exact form. We provide insights and motivations, and we discuss the qualitative behavior of the algorithm. We also derive some properties of the associated mappings, which, among others, will be important in the convergence analysis. Both the algorithm, in its prototype form, and the analysis will serve as a basis for asynchronous and stochastic iterative algorithms to be developed in the subsequent sections.

2.1. An optimal stopping formulation for policy evaluation.

The standard PI algorithm involves the mapping F_μ of Equation (1.3) for policy evaluation. The key idea of our approach is to replace F_μ with another mapping $F_{J,\nu}$, which depends on a vector J, with components denoted $J(i)$, and on a randomized policy ν that for each state i defines a probability distribution

$$\{\nu(u \mid i) \mid u \in U(i)\}$$

over the feasible controls at i. It maps Q, a vector of Q-factors, to $F_{J,\nu}Q$, the vector of Q-factors with components given by

$$(F_{J,\nu}Q)(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \sum_{v\in U(j)} \nu(v \mid j) \min\{J(j), Q(j,v)\}\right), \quad \forall\, u \in U(i), \quad i = 1,\ldots,n. \tag{2.1}$$

Comparing $F_{J,\nu}$ and the classical Q-learning mapping F of Equation (1.4) (or the mapping F_μ of Equation (1.3)), we see that they take into account the Q-factors of the next state j differently: F (or F_μ) uses the minimal Q-factor $\min_{v\in U(j)} Q(j,v)$ (the Q-factor $Q(j,\mu(j))$, respectively), while $F_{J,\nu}$ uses a randomized Q-factor (according to $\nu(v \mid j)$), but only up to the threshold $J(j)$. Note that $F_{J,\nu}$ does not require the overhead for minimization over all controls that the Q-learning mapping F does (see Equation (1.4)).

The mapping $F_{J,\nu}$ can be interpreted in terms of an optimal stopping problem defined as follows:

(a) The state space is the set of state-control pairs (i,u) of the original problem.

(b) When at state (i,u), if we decide to stop, we incur a stopping cost $J(i)$ (independent of u).

(c) When at state (i,u), if we decide not to stop, we incur a one-stage cost $\sum_{j=1}^{n} p_{ij}(u) g(i,u,j)$ and transition to state (j,v) with probability $p_{ij}(u)\nu(v \mid j)$.

The mapping $F_{J,\nu}$ is a sup-norm contraction of modulus α for all ν and J, i.e.,

$$\|F_{J,\nu}Q - F_{J,\nu}\tilde{Q}\|_\infty \le \alpha \|Q - \tilde{Q}\|_\infty, \quad \forall\, Q, \tilde{Q},$$

where $\|\cdot\|_\infty$ denotes the sup-norm ($\|Q\|_\infty = \max_{(i,u)} |Q(i,u)|$). This follows from well-known general properties of Q-learning for optimal stopping problems (see Bertsekas and Tsitsiklis [11], Tsitsiklis and Van Roy [45]). Hence, $F_{J,\nu}$ has a unique fixed point, which we denote by $Q_{J,\nu}$. We may interpret $Q_{J,\nu}(i,u)$ as a Q-factor of the optimal stopping problem corresponding to the nonstopping action, i.e., the optimal cost-to-go starting at (i,u) and conditioned on the first decision being not to stop.

There is a relation between $Q_{J,\nu}$ of the optimal stopping problem and the Q-factors of the original problem, which gives insight and motivation of the mapping $F_{J,\nu}$: if J is the cost of some policy π, which can be randomized and history dependent, then the components of $Q_{J,\nu}$ correspond to the Q-factors of a policy that switches optimally from following the policy ν to following the policy π. This relation is desirable in model-free learning and motivates the use of $F_{J,\nu}$ for policy evaluation in that setting, because by the nature of learning, one has to try out policies that are not necessarily overall better than the extensively tested ones. If J represents the present approximation of the optimal costs, then $F_{J,\nu}$ has the built-in capability to take the "improving part" of ν into account in policy evaluation while avoiding the "nonimproving part" of ν. A special, extreme case of the relation is the following important property of $F_{J,\nu}$.

Lemma 2.1. *For all ν, we have*

$$F_{J^*,\nu}Q^* = Q^*, \qquad Q_{J^*,\nu} = Q^*.$$

Proof. Since $J^*(i) = \min_{u\in U(i)} Q^*(i,u)$ (see Equation (1.1)), we have using Equations (1.4) and (2.1), $F_{J^*,\nu}Q^* = FQ^* = Q^*$ for all ν, which is the first equality. The second equality then follows from the first and the fact that $F_{J^*,\nu}$ has a unique fixed point. □

The mapping $F_{J,\nu}$ has another property, which will be essential in forming contraction mappings on the joint space of (J,Q) that underly our algorithms.

Proposition 2.1. *For all ν, J, $\tilde J$, Q, and $\tilde Q$, we have*

$$\|F_{J,\nu}Q - F_{\tilde J,\nu}\tilde Q\|_\infty \le \alpha \max\{\|J-\tilde J\|_\infty,\ \|Q-\tilde Q\|_\infty\}.$$

In particular,

$$\|F_{J,\nu}Q - Q^*\|_\infty \le \alpha \max\{\|J-J^*\|_\infty,\ \|Q-Q^*\|_\infty\}. \tag{2.2}$$

Proof. In the proof, let J^{ext} denote the vector J extended to the space of state-control pairs by

$$J^{\text{ext}}(i,u) = J(i), \quad \forall\, u \in U(i),$$

and let minimization over two vectors be interpreted componentwise, i.e., let $\min\{Q_1,Q_2\}$ denote the vector with components $\min\{Q_1(i,u), Q_2(i,u)\}$.

We write

$$F_{J,\nu}Q = \bar g + \alpha \overline{P}_\nu \min\{J^{\text{ext}}, Q\}, \tag{2.3}$$

where $\bar g$ is the vector with components

$$\sum_{j=1}^n p_{ij}(u) g(i,u,j), \quad \forall\, u\in U(i),\ \ i = 1,\ldots,n,$$

and $\overline{P}_\nu$ is the transition probability matrix with probabilities of transition $(i,u)\to(j,v)$ equal to

$$p_{ij}(u)\nu(v\mid j), \quad \forall\, i\in U(i),\ \ v\in U(j),\ \ i,j=1,\ldots,n.$$

From Equations (2.3), we obtain

$$\|F_{J,\nu}Q - F_{\hat J,\nu}\tilde Q\|_\infty \le \alpha \|\min\{J^{\text{ext}},Q\} - \min\{\tilde J^{\text{ext}},\tilde Q\}\|_\infty.$$

We also have[1]

$$\|\min\{J^{\text{ext}},Q\} - \min\{\tilde J^{\text{ext}},\tilde Q\}\|_\infty \le \max\{\|J-\tilde J\|_\infty,\ \|Q-\tilde Q\|_\infty\}.$$

The preceding two relations imply the first inequality in the proposition. By taking $\tilde J = J^*$, $\tilde Q = Q^*$ in the latter inequality, and using also Lemma 2.1, we obtain the second inequality (2.2). □

[1] Here we are using a nonexpansiveness property of the minimization map: for any Q_1, Q_2, $\tilde Q_1$, $\tilde Q_2$, we have $\|\min\{Q_1,Q_2\} - \min\{\tilde Q_1,\tilde Q_2\}\|_\infty \le \max\{\|Q_1-\tilde Q_1\|_\infty, \|Q_2-\tilde Q_2\|_\infty\}$. To see this, write, for every (i,u), $Q_m(i,u) \le \max\{\|Q_1-\tilde Q_1\|_\infty, \|Q_2-\tilde Q_2\|_\infty\} + \tilde Q_m(i,u)$, $m=1,2$, take the minimum of both sides over m, exchange the roles of Q_m and $\tilde Q_m$, and take the maximum over (i,u).

Equation (2.2) shows a worst-case bound on the distance from $F_{J,\nu}Q$ to Q^*, which depends on the distance between (J, Q) and (J^*, Q^*) and involves a contraction factor α. As will be shown later, this property of $F_{J,\nu}$ will ensure the convergence of our algorithms regardless of the choices of policies ν, and thereby allow a lot of freedom in algorithm design. We may contrast Equation (2.2) with a similar bound corresponding to the pure policy evaluation mapping F_μ of Equation (1.3),

$$\|F_\mu Q - Q_\mu\|_\infty \le \alpha \|Q - Q_\mu\|_\infty,$$

which involves Q_μ rather than Q^*.

2.2. A prototype algorithm and its qualitative behavior. Here is a prototype algorithm that will serve as the starting point for our versions of asynchronous modified PI and stochastic Q-learning algorithms in §§3 and 4. It generates a sequence of pairs (J_k, Q_k), starting from an arbitrary pair (J_0, Q_0). Given (J_k, Q_k), we select an arbitrary randomized policy ν_k and an arbitrary positive integer m_k, and we obtain the next pair (J_{k+1}, Q_{k+1}) as follows.

Iteration k of Q-Learning/Enhanced PI Algorithm:

(1) Generate Q_{k+1} with m_k iterations involving the mapping F_{J_k,ν_k}, with ν_k and J_k held fixed:

$$Q_{k+1} = F^{m_k}_{J_k,\nu_k} Q_k. \tag{2.4}$$

(2) Update J_{k+1} by

$$J_{k+1}(i) = \min_{u\in U(i)} Q_{k+1}(i,u), \quad \forall\, i = 1, \ldots, n. \tag{2.5}$$

Depending on the choices of ν_k and m_k, the algorithm's behavior is intermediate between VI and PI. It will be shown in the next subsection that regardless of these choices, the algorithm converges to (J^*, Q^*). Thus, there is a lot of freedom in choosing ν_k and m_k to affect the qualitative behavior of the algorithm, which we discuss now.

In one extreme case, the algorithm can behave as PI. If we let $m_k = \infty$, then Q_{k+1} is the Q-factor Q_{J_k,ν_k} of the associated stopping problem (the unique fixed point of F_{J_k,ν_k}, which can be obtained by solving the stopping problem directly instead of solving it by using VI), and the algorithm takes the form

$$J_{k+1}(i) = \min_{u\in U(i)} Q_{J_k,\nu_k}(i,u), \quad \forall\, i = 1, \ldots, n. \tag{2.6}$$

If in addition ν_k is chosen to be a deterministic policy μ_k that attains the minimum in the equation

$$\mu_k(i) = \arg\min_{u\in U(i)} Q_k(i,u), \quad \forall\, i = 1, \ldots, n, \tag{2.7}$$

with ν_0 being some deterministic policy μ_0 satisfying $J_0 \ge J_{\mu_0}$, then $Q_1 = Q_{J_0,\mu_0}$ is also the (exact) Q-factor vector of μ_0 (since $J_0 \ge J_{\mu_0}$), so μ_1 as generated by Equation (2.7), is the policy generated from μ_0 by exact policy improvement for the original MDP. Similarly, it can be shown by induction that for $m_k = \infty$ and $\nu_k = \mu_k$, the algorithm generates the same sequence of policies as exact PI for the original MDP.

In the other extreme case, the algorithm can behave as VI. We describe two different types of choices of ν_k and m_k for which this can happen. In the first setting, $m_k = 1$ for all k. Then,

$$\begin{aligned} Q_{k+1}(i,u) &= \sum_{j=1}^n p_{ij}(u)\biggl(g(i,u,j) + \alpha \sum_{v\in U(j)} \nu_k(v\mid j) \min\Bigl\{\min_{v'\in U(j)} Q_k(j,v'),\, Q_k(j,v)\Bigr\}\biggr) \\ &= \sum_{j=1}^n p_{ij}(u)\Bigl(g(i,u,j) + \alpha \min_{v\in U(j)} Q_k(j,v)\Bigr), \quad \forall\, u \in U(i),\ \ i = 1, \ldots, n, \end{aligned}$$

so an iteration of the algorithm coincides with the VI algorithm in Q-factor space, $Q_{k+1} = FQ_k$.

The second setting is important and illuminates the qualitative dimension of the algorithm associated with the randomized policies ν_k. Suppose ν_k is chosen to differ from the deterministic policy μ_k of Equation (2.7). As mentioned in §2.1, in model-free learning, there is often a need to try policies significantly different from the well-tested ones, for the sake of exploring the state-control space. We refer loosely to this difference, as expressed for example by the probabilities $\sum_{v\in U(j),\, v\neq\mu_k(j)} \nu_k(v\mid j)$, as the degree of *exploration* embodied by ν_k;

the use of this term will be explained further in §7. As an illustration, let us suppose that $m_k = \infty$, and ν_k is chosen to assign positive probability to only nonoptimal controls, so that $\nu_k(\mu^*(j) \mid j) = 0$ for all j and optimal policies μ^*. Then since $J_k \to J^*$ and $Q_{J_k,\nu_k} = Q_{k+1} \to Q^*$ (as we will show shortly), we have, for all j and sufficiently large k, $J_k(j) < Q_{J_k,\nu_k}(j, v)$ for all v with $\nu_k(v \mid j) > 0$, so that

$$\begin{aligned} J_{k+1}(i) &= \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\Bigl(g(i,u,j) + \alpha \sum_{v\in U(j)} \nu_k(v \mid j) \min\{J_k(j), Q_{J_k,\nu_k}(j,v)\}\Bigr) \\ &= \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha J_k(j)), \quad \forall\, i = 1,\ldots,n. \end{aligned}$$

Thus the algorithm, for sufficiently large k, reduces to synchronous Q-learning/VI for the original MDP, even though $m_k = \infty$, and produces the same results as with the choice $m_k = 1$ (or any value of m_k). The above characteristic of the algorithm provides also an intuitive view of the guaranteed convergence of the algorithm, to be proved shortly; that is, even with adversarial choices of the various quantities involved, the VI character of the algorithm will ensure its convergence. This is also essentially the case with asynchronous distributed and stochastic versions of the algorithm discussed in the subsequent sections, although the behavior of those algorithms are more complex.

Generally between the above two extremes, the iteration (2.4)–(2.5) on one hand resembles in some ways the classical modified PI, where policy evaluation is approximated with a finite number m_k of value iterations, with the case $m_k = 1$ corresponding to value iteration/synchronous Q-learning, and the case $m_k = \infty$ corresponding to (exact) policy iteration. On the other hand, the choice of ν_k also affects the qualitative character of our algorithm, as the preceding arguments illustrate. With little exploration (approaching the extreme case where ν_k is the deterministic policy (2.7)) our algorithm tends to act nearly like modified PI (or exact PI for $m_k = \infty$). With substantial exploration (approaching the extreme case where $\nu_k(\mu_k(j) \mid j) = 0$ for any policy μ_k generated according to Equation (2.7)) it tends to act nearly like Q-learning/VI, regardless of the value of m_k. (This reasoning also suggests that with substantial exploration it may be better to use small values of m_k.)

2.3. Convergence based on contraction on joint cost and Q-factor space. The convergence analysis given below is general and applies to any choice of the policies $\{\nu_k\}$ and the numbers $\{m_k\}$ of policy evaluation iterations in the algorithm (2.4)–(2.5). It shows that the algorithm converges with a rate of convergence that is at least geometric. First, from a repeated application of the contraction property of $F_{J,\nu}$ given by Proposition 2.1, we have the following lemma.

Lemma 2.2. *For all ν, J, $\tilde{J}$, Q, $\tilde{Q}$, and $m \geq 1$, we have*

$$\|F^m_{J,\nu} Q - F^m_{\tilde{J},\nu} \tilde{Q}\|_\infty \leq \alpha \max\{\|J - \tilde{J}\|_\infty, \|Q - \tilde{Q}\|_\infty\}.$$

Let us now introduce a mapping corresponding to an iteration of the algorithm (2.4)–(2.5) and on the joint space of (J, Q):

$$G_{\nu,m}(J, Q) = (MF^m_{J,\nu} Q, F^m_{J,\nu} Q), \tag{2.8}$$

where $MF^m_{J,\nu} Q$ is the vector with components

$$(MF^m_{J,\nu} Q)(i) = \min_{u\in U(i)} (F^m_{J,\nu} Q)(i,u), \quad i = 1,\ldots,n. \tag{2.9}$$

Here and later, we denote by M the minimization operation over $u \in U(i)$, for each i, and we note that for all Q, $\tilde{Q}$, we have[2]

$$\|MQ - M\tilde{Q}\|_\infty \leq \|Q - \tilde{Q}\|_\infty. \tag{2.10}$$

We now show the contraction property that underlies the convergence of our algorithm.

Lemma 2.3. *For all ν and $m \geq 1$, $G_{\nu,m}$ is a sup-norm contraction with modulus α, and (J^*, Q^*) is its unique fixed point.*

[2] For a proof, write $Q(i,u) \leq \|Q - \tilde{Q}\|_\infty + \tilde{Q}(i,u)$, $\forall\, u \in U(i)$, $i = 1,\ldots,n$, take the minimum of both sides over $u \in U(i)$, exchange the roles of Q and $\tilde{Q}$, and take the maximum over i.

Proof. Consider any ν and $m \geq 1$. By Lemma 2.1 $F^m_{J^*,\nu} Q^* = Q^*$, so $G_{\nu,m}(J^*, Q^*) = (MQ^*, Q^*) = (J^*, Q^*)$, where the last equality follows from the relation $J^*(i) = \min_{u\in U(i)} Q^*(i, u)$ for all states i (see Equation (1.1)), which is $J^* = MQ^*$.

To show the contraction property, for any (J_1, Q_1) and $(\tilde{J}_1, \tilde{Q}_1)$, let $(J_2, Q_2) = G_{\nu,m}(J_1, Q_1)$ and $(\tilde{J}_2, \tilde{Q}_2) = G_{\nu,m}(\tilde{J}_1, \tilde{Q}_1)$. By Lemma 2.2 and Equation (2.10),

$$\|Q_2 - \tilde{Q}_2\|_\infty \leq \alpha \|(J_1, Q_1) - (\tilde{J}_1, \tilde{Q}_1)\|_\infty \qquad \text{and} \qquad \|J_2 - \tilde{J}_2\|_\infty = \|MQ_2 - M\tilde{Q}_2\|_\infty \leq \|Q_2 - \tilde{Q}_2\|_\infty.$$

Hence,

$$\|(J_2, Q_2) - (\tilde{J}_2, \tilde{Q}_2)\|_\infty = \max\{\|J_2 - \tilde{J}_2\|_\infty, \|Q_2 - \tilde{Q}_2\|_\infty\} \leq \alpha \|(J_1, Q_1) - (J_2, Q_2)\|_\infty,$$

establishing the contraction property of $G_{\nu,m}$. □

The following convergence result is a consequence of the preceding proposition.

Proposition 2.2. *For any choice of (J_0, Q_0), $\{\nu_k\}$, and $\{m_k\}$, a sequence $\{(J_k, Q_k)\}$ generated by the algorithm (2.4)–(2.5) converges to (J^*, Q^*), and the rate of convergence is geometric. Furthermore, for all k after some index $\bar{k}$, the policies μ_k obtained from the minimization (2.5),*

$$\mu_k(i) \in \arg\min_{u\in U(i)} Q_k(i, u), \qquad \forall\, i = 1, \ldots, n,$$

are optimal.

Proof. It follows from Lemma 2.3 that $\{(J_k, Q_k)\}$ converges to (J^*, Q^*) geometrically. The optimality of μ_k for sufficiently large k follows from the convergence $Q_k \to Q^*$, since a policy μ^* is optimal if and only if $\mu^*(i)$ minimizes $Q^*(i, u)$ over $U(i)$ for all i. □

3. Deterministic asynchronous versions of the algorithm. The modified PI-like algorithm (2.4)–(2.5) may be viewed as synchronous in the sense that the Q-factors of all state-control pairs are simultaneously updated at each iteration. The contraction properties of the underlying mappings can be used to establish the convergence of the algorithm under far more irregular conditions. In particular, we consider in this section asynchronous updating of Q-factors and state costs corresponding to blocks of components, and we discuss in §4 model-free sampled versions, which do not require the explicit knowledge of $p_{ij}(u)$ and the calculation of expected values.

In standard asynchronous versions of PI for Q-factors (see Equations (1.5) and (1.6)), the updates of μ and Q are executed selectively, for only some of the states and state-control pairs. The algorithm generates a sequence of pairs (Q_k, μ_k), starting from an arbitrary pair (Q_0, μ_0) as follows:

$$Q_{k+1}(i, u) = \begin{cases} (F_{\mu_k} Q_k)(i, u) & \text{if } (i, u) \in R_k, \\ Q_k(i, u) & \text{if } (i, u) \notin R_k, \end{cases} \tag{3.1}$$

$$\mu_{k+1}(j) = \begin{cases} \arg\min_{v\in U(j)} Q_k(j, v) & \text{if } j \in S_k, \\ \mu_k(j) & \text{if } j \notin S_k, \end{cases} \tag{3.2}$$

where R_k and S_k are subsets of state-control pairs and states, respectively, one of which may be empty. (For an analysis and discussion, see, e.g., Bertsekas and Tsitsiklis [11, Section 2.2], Bertsekas [6, Section 1.3.3].) Relative to ordinary VI for Q-factors, the advantage is that the minimization in Equation (3.2) is performed only at times when S_k is nonempty, and only for the states in S_k (rather than at each iteration, and for all states). This is the generic advantage that modified PI has over ordinary VI: the policy improvement iterations of the form (3.2) can be performed much less frequently than the policy evaluation iterations of the form (3.1).

Unfortunately, the convergence of the asynchronous PI (3.1)–(3.2) to Q^* is questionable in the absence of additional restrictions; some assumption, such as $F_{\mu_0} Q_0 \leq Q_0$, is required for the initial policy μ_0 and vector Q_0 (see Williams and Baird [49] for a proof and counterexamples to convergence, or Bertsekas and Tsitsiklis [11, Proposition 2.5], Bertsekas [6, Proposition 1.3.5]). The restriction $F_{\mu_0} Q_0 \leq Q_0$ can be satisfied by adding to Q_0 a sufficiently large multiple of the unit vector (this is true only for the discounted DP model, and may not be possible for other types of DP models). The need for it, however, indicates that the convergence properties of the algorithm (3.1)–(3.2) are fragile and sensitive to the assumptions, which may cause convergence difficulties in both its deterministic and its stochastic simulation-based variants. In particular, no related convergence results or counterexamples are currently known for the case where the expected value of Equations (3.1) is replaced by a single sample in a stochastic approximation-type update.

In a corresponding asynchronous version of our algorithm (2.4)–(2.5), again Q is updated selectively, for only some of the state-control pairs, and J is also updated at some iterations and for some of the states. There may also be a policy μ that is maintained and updated selectively at some of the states. This policy may be used to generate a randomized policy ν that enters the algorithm in a material way. However, the algorithm is valid for any choice of ν, so its definition need not involve the policy μ and the method in which it is used to update ν (we will later give an example of an updating scheme for μ and ν). Specifically, our asynchronous algorithm, stated in general terms, generates a sequence of pairs (J_k, Q_k), starting from an arbitrary pair (J_0, Q_0). Given (J_k, Q_k), we obtain the next pair (J_{k+1}, Q_{k+1}) as follows.

Asynchronous Policy Iteration:

Select a randomized policy ν_k, a subset R_k of state-control pairs, and a subset of states S_k such that $R_k \cup S_k \neq \varnothing$. Generate Q_{k+1} according to

$$Q_{k+1}(i,u) = \begin{cases} (F_{J_k,\nu_k} Q_k)(i,u) & \text{if } (i,u) \in R_k, \\ Q_k(i,u) & \text{if } (i,u) \notin R_k, \end{cases} \tag{3.3}$$

and generate J_{k+1} according to

$$J_{k+1}(i) = \begin{cases} \min_{u \in U(i)} Q_k(i,u) & \text{if } i \in S_k, \\ J_k(i) & \text{if } i \notin S_k. \end{cases} \tag{3.4}$$

Similar to modified PI and its asynchronous version, when the above method is applied, typically for many iterations the sets S_k are empty and only the update (3.3) is performed on Q-factors. As mentioned in §2.2, ν_k may be selected in special ways so that it gives the algorithm a PI character, which can then be compared with (synchronous or asynchronous) modified PI for Q-factors, such as the one of Equations (3.1) and (3.2). For an example of such an algorithm, assume that a policy μ_k is also maintained, which defines ν_k (so ν_k is the deterministic policy μ_k). The algorithm updates Q according to

$$Q_{k+1}(i,u) = \begin{cases} (F_{J_k,\mu_k} Q_k)(i,u) & \text{if } (i,u) \in R_k, \\ Q_k(i,u) & \text{if } (i,u) \notin R_k, \end{cases} \tag{3.5}$$

and it updates J and μ according to

$$J_{k+1}(j) = \begin{cases} \min\limits_{v \in U(j)} Q_k(j,v) & \text{if } j \in S_k, \\ J_k(j) & \text{if } j \notin S_k, \end{cases} \qquad \mu_{k+1}(j) = \begin{cases} \arg\min\limits_{v \in U(j)} Q_k(j,v) & \text{if } j \in S_k, \\ \mu_k(j) & \text{if } j \notin S_k, \end{cases} \tag{3.6}$$

where R_k and S_k are subsets of state-control pairs and states, respectively, one of which may be empty.

We may view Equation (3.5) as a policy evaluation iteration for the state-control pairs in R_k, and Equation (3.6) as a policy improvement iteration only for the states in S_k. In comparing the new algorithm (3.5)–(3.6) with the known algorithm (3.1)–(3.2), we see that the essential difference is that Equation (3.5) involves the use of J_k and the minimization on the right-hand side, whereas Equation (3.1) does not. As we will show in the following proposition, this precludes the kind of anomalous behavior that is exhibited in the Williams and Baird [49] counterexamples mentioned earlier. Mathematically, the reason for this may be traced to the presence of the cost vector J in Equation (3.3) and its special case Equation (3.5), and the sup-norm contraction in the space of (J, Q), which underlies iterations (3.3)–(3.4) and (3.5)–(3.6) (see Proposition 2.1, Lemma 2.3, and also Proposition 4.1).

Note that for computational efficiency, one may give preference for inclusion in the set R_k to the state-control pairs $(j, \mu_k(j))$, $j = 1, \ldots, n$, and one may choose the sets R_k and S_k in specific ways to avoid storing all Q-factors.[3] But these implementation details, which we will not go into here, are not requirements for the convergence result of the subsequent proposition.

[3] If one is interested in computing just the optimal costs $J^*(i)$ and not the optimal Q-factors $Q^*(i,u)$, one may only maintain the Q-factors $Q(j, \mu_k(j))$, because these are the only ones needed to perform the update (3.5). The other Q-factors can be calculated just before the update (3.6) is performed for the states in S_k, and the results need not be stored except for $Q(j, \mu_{k+1}(j))$, $j \in S_k$.

The convergence result below bears similarity to general convergence results for asynchronous distributed DP and related algorithms involving sup-norm contractions (see Bertsekas [3, 4], Bertsekas and Tsitsiklis [12, Section 6.2]).

Proposition 3.1. *Assume that each pair (i, u) is included in the set R_k infinitely often, and each state i is included in the set S_k infinitely often. Then any sequence $\{(J_k, Q_k)\}$ generated by the algorithm (3.3)–(3.4) converges to (J^*, Q^*).*

Proof. Let $\{k_j\}$ and $\{\hat{k}_j\}$ be sequences of iteration indices such that $k_0 = 0$, $k_j < \hat{k}_j < k_{j+1}$ for $j = 0, 1, \ldots,$ and for all j, each (i, u) is included in $\bigcup_{k=k_j}^{\hat{k}_j-1} R_k$ at least once, and each i is included in $\bigcup_{k=\hat{k}_j}^{k_{j+1}-1} S_k$ at least once. Thus, between iterations k_j and $\hat{k}_j$, each component of Q is updated at least once, and between iterations $\hat{k}_j$ and k_{j+1}, each component of J is updated at least once.

By using Proposition 2.1, we have, for all k,

$$|Q_{k+1}(i, u) - Q^*(i, u)| \le \alpha \max\{\|J_k - J^*\|_\infty, \|Q_k - Q^*\|_\infty\}, \quad \forall\, (i, u) \in R_k, \tag{3.7}$$

$$Q_{k+1}(i, u) = Q_k(i, u), \quad \forall\, (i, u) \notin R_k. \tag{3.8}$$

Also, by using the nonexpansive property of the minimization operation (see Equation (2.10)), we have, for all k,

$$|J_{k+1}(i) - J^*(i)| \le \|Q_k - Q^*\|_\infty, \quad \forall\, i \in S_k, \tag{3.9}$$

$$J_{k+1}(i) = J_k(i), \quad \forall\, i \notin S_k. \tag{3.10}$$

From these relations, it follows that

$$\max\{\|J_{k+1} - J^*\|_\infty, \|Q_{k+1} - Q^*\|_\infty\} \le \max\{\|J_k - J^*\|_\infty, \|Q_k - Q^*\|_\infty\}, \quad \forall\, k = 0, 1, \ldots. \tag{3.11}$$

For each $k \in [\hat{k}_j, k_{j+1}]$, we have from Equations (3.7) and (3.8), for all $u \in U(i)$, $i = 1, \ldots, n$,

$$|Q_k(i, u) - Q^*(i, u)| \le \alpha \max\{\|J_{\tilde{k}(i,u,k)} - J^*\|_\infty, \|Q_{\tilde{k}(i,u,k)} - Q^*\|_\infty\}, \tag{3.12}$$

where $\tilde{k}(i, u, k)$ is the last iteration index between k_j and k when the component $Q(i, u)$ is updated. Since each component of Q is updated at least once between iterations k_j and $k \in [\hat{k}_j, k_{j+1}]$, using also Equation (3.11), it follows that

$$\|Q_k - Q^*\|_\infty \le \alpha \max\{\|J_{k_j} - J^*\|_\infty, \|Q_{k_j} - Q^*\|_\infty\}, \quad \forall\, j = 0, 1, \ldots, \quad k \in [\hat{k}_j, k_{j+1}]. \tag{3.13}$$

Since each component of J is updated at least once between iterations $\hat{k}_j$ and k_{j+1}, we have from Equations (3.9) and (3.10) that

$$|J_{k_{j+1}}(i) - J^*(i)| \le \|Q_{\tilde{k}(i)} - Q^*\|_\infty, \quad \forall\, i = 1, \ldots, n,$$

where $\tilde{k}(i)$ is the last iteration index between $\hat{k}_j$ and k_{j+1} when the component $J(i)$ is updated, so from Equation (3.13), it follows that

$$\|J_{k_{j+1}} - J^*\|_\infty \le \alpha \max\{\|J_{k_j} - J^*\|_\infty, \|Q_{k_j} - Q^*\|_\infty\}, \quad \forall\, j = 0, 1, \ldots. \tag{3.14}$$

Combining Equations (3.13) and (3.14), we obtain

$$\max\{\|J_{k_{j+1}} - J^*\|_\infty, \|Q_{k_{j+1}} - Q^*\|_\infty\} \le \alpha \max\{\|J_{k_j} - J^*\|_\infty, \|Q_{k_j} - Q^*\|_\infty\}, \quad \forall\, j = 0, 1, \ldots,$$

so $\max\{\|J_{k_j} - J^*\|_\infty, \|Q_{k_j} - Q^*\|_\infty\} \to 0$ as $j \to \infty$, i.e., $(J_{k_j}, Q_{k_j}) \to (J^*, Q^*)$ as $j \to \infty$. Using also Equation (3.11), this implies that the entire sequence $\{(J_k, Q_k)\}$ converges to (J^*, Q^*). □

4. Stochastic iterative versions of the algorithm.

In this section we consider stochastic iterative algorithms, which are patterned after the classical Q-learning algorithm of Watkins [48] (see also Tsitsiklis [42]), as well as optimistic and modified PI methods (Bertsekas and Tsitsiklis [11, Section 5.4]). We will compare our algorithm with the classical Q-learning algorithm, whereby we generate a sequence of state-control pairs $\{(i_k, u_k) \mid k = 0, 1, \ldots\}$ by any probabilistic mechanism that guarantees that each pair (i, u) appears infinitely often with probability 1, and at each time k, we generate a successor state j_k according to the distribution $p_{i_k j}(u_k)$, $j = 1, \ldots, n$, and we update only the Q-factor of (i_k, u_k),

$$Q_{k+1}(i_k, u_k) = (1 - \gamma_{(i_k, u_k), k}) Q_k(i_k, u_k) + \gamma_{(i_k, u_k), k} \Big(g(i_k, u_k, j_k) + \alpha \min_{v \in U(j_k)} Q_k(j_k, v) \Big), \tag{4.1}$$

while leaving all other components of Q_k unchanged: $Q_{k+1}(i, u) = Q_k(i, u)$ for all $(i, u) \neq (i_k, u_k)$. The positive stepsizes $\gamma_{(i_k, u_k), k}$ may depend on the current pair (i_k, u_k) and must satisfy assumptions that are standard in stochastic approximation methods (i.e., must diminish to 0 at a suitable rate). There are also distributed asynchronous versions of the algorithm (4.1), where $Q_k(j_k, v)$ may be replaced by $Q_{\tau_{k,v}}(j_k, v)$, where $k - \tau_{k,v}$ may be viewed as a nonnegative integer "delay" that depends on k and v, as discussed by Tsitsiklis [42] and other sources on asynchronous stochastic approximation methods such as Tsitsiklis et al. [46], Bertsekas and Tsitsiklis [12], Borkar [16], Abounadi et al. [1], and Borkar [17].

In §4.1 we present three model-free optimistic PI algorithms, which update a cost vector J in addition to the Q-factor vector Q, similar to the algorithms of §§2 and 3. We refer to these algorithms as Algorithms I–III, and we briefly describe them below:

(I) This algorithm resembles the classical Q-learning algorithm (4.1), but requires less overhead per iteration (the minimization over $u \in U(j)$ is replaced by a simpler operation). It also bears similarity with a natural form of partially optimistic TD(0) algorithm, which is unreliable, as discussed by Bertsekas and Tsitsiklis [11, Section 5.4]; in contrast to that algorithm, our Algorithm I has guaranteed convergence.

(II) This algorithm parallels the asynchronous PI method (3.3)–(3.4) of the preceding section, but is model free and uses a single sample per state instead of computing an expected value.

(III) This algorithm generalizes the first two in several ways and allows, among others, "delayed" components of state costs and Q-factors in its iteration. The extra generality is helpful in addressing implementations in an asynchronous distributed computing system, and also sets the stage for an abstraction of the algorithms that facilitates the convergence analysis, as we discuss in §4.2.

We find it useful to present Algorithms I and II first, because they are simpler and comparable in performance to Algorithm III for many situations, and they help to motivate the more general Algorithm III. We establish the convergence of the latter algorithm using the asynchronous stochastic approximation-type convergence framework of Tsitsiklis [42] in §4.3.

In what follows, all the variables involved in the algorithms (states, state-control pairs, costs of states, Q-factors, policies, sets of indexes that determine which components are updated, etc.) are to be viewed as random variables defined on a common probability space. Specific technical assumptions about their probabilistic properties will be given at the appropriate points later.

The method for selecting components of Q to update is left largely unspecified in the formal description of the algorithms because it does not affect our convergence result, as long as all the components are updated infinitely often. However, for efficiency one may give preference to the state-control pairs (j, v) with $\nu_k(v \mid j) > 0$ (see related discussion in §3). Similarly, we do not discuss in detail the choice of the randomized policies ν_k, which is primarily dictated by exploration concerns that tend to be problem specific. The choice of ν_k also affects the character of the algorithm (the extent to which it resembles the algorithm VI or modified PI, as discussed in §2.2), and this should be balanced against the desire for exploration. Some natural choices of the policies ν_k and subsets of components for update are illustrated for specific example problems in §5.

4.1. Some model-free optimistic policy iteration algorithms. Similar to classical Q-learning (4.1), our first algorithm generates a sequence of state-control pairs $\{(i_k, u_k) \mid k = 0, 1, \ldots\}$ and updates only the Q-factor of (i_k, u_k) at iteration k, using a positive stepsize $\gamma_{(i_k, u_k), k}$. At selected iterations $k \in K_J$, it also updates a single component of J, where K_J is an infinite subset of iteration indices (which need not be predetermined, but may depend on algorithmic progress). The algorithm may choose ν_k arbitrarily for each k and with dependence on (i_k, u_k), but one possibility is to maintain a policy μ_k that is updated at selected states simultaneously with J, and then use $\nu_k = \mu_k$, similar to algorithm (3.5)–(3.6). Furthermore, the controls u_k may be generated in accordance with ν_k; this gives the algorithm a modified/optimistic PI character. The states i_{k+1} may be generated according to $p_{i_k j}(u_k)$, as in some optimistic PI methods, although this is not essential for the convergence of the algorithm. Compared to the preceding Q-learning algorithm (4.1), the algorithm has an advantage similar to the one that modified PI has over VI (less overhead because it does not require the minimization over all controls $v \in U(j)$ at every iteration). In particular, given the pair (J_k, Q_k), the algorithm obtains (J_{k+1}, Q_{k+1}) as follows.

Model-Free Optimistic Policy Iteration I:

(1) Select a state-action pair (i_k, u_k). If $k \in K_J$, update J_k according to

$$J_{k+1}(j) = \begin{cases} \min_{v \in U(j)} Q_k(j, v) & \text{if } j = i_k, \\ J_k(j) & \text{if } j \neq i_k; \end{cases} \tag{4.2}$$

otherwise, leave J_k unchanged ($J_{k+1} = J_k$).

(2) Select a stepsize $\gamma_{(i_k, u_k), k} \in (0, 1]$ and a policy $\nu_{(i_k, u_k), k}$. Generate a successor state j_k according to the distribution $p_{i_k j}(u_k)$, $j = 1, \ldots, n$, and generate a control v_k according to the distribution $\nu_{(i_k, u_k), k}(v \mid j_k)$, $v \in U(j_k)$.

(3) Update the (i_k, u_k)th component of Q according to

$$Q_{k+1}(i_k, u_k) = (1 - \gamma_{(i_k, u_k), k}) Q_k(i_k, u_k) + \gamma_{(i_k, u_k), k}\big(g(i_k, u_k, j_k) + \alpha \min\{J_k(j_k), Q_k(j_k, v_k)\}\big), \tag{4.3}$$

and leave all other components of Q_k unchanged: $Q_{k+1}(i, u) = Q_k(i, u)$ for all $(i, u) \neq (i_k, u_k)$.

The preceding algorithm (Algorithm I) has similarities with the partially optimistic TD(0) algorithm, discussed by Bertsekas and Tsitsiklis [11, Section 5.4]. The latter algorithm updates only J (rather than (J, Q)) using TD(0), and also maintains a policy that is updated at selected iterations. However, its convergence properties are dubious, as discussed by Bertsekas and Tsitsiklis [11, p. 231] (see also Tsitsiklis [43]). In contrast, we will show that our algorithm above has satisfactory convergence properties.

We now give another stochastic iterative algorithm, which parallels the asynchronous PI method (3.3)–(3.4) of §3. Given the pair (J_k, Q_k), the algorithm obtains (J_{k+1}, Q_{k+1}) as follows.

Model-Free Optimistic Policy Iteration II:

Select a subset R_k of state-control pairs and a subset of states S_k such that $R_k \cup S_k \neq \varnothing$.

Update J_k according to

$$J_{k+1}(i) = \begin{cases} \min_{u \in U(i)} Q_k(i, u) & \text{if } i \in S_k, \\ J_k(i) & \text{if } i \notin S_k. \end{cases} \tag{4.4}$$

For each $\ell = (i, u) \in R_k$, select a stepsize $\gamma_{\ell, k} \in (0, 1]$ and a policy $\nu_{\ell, k}$, and:

(1) Generate a successor state j_k according to the distribution $p_{ij}(u)$, $j = 1, \ldots, n$, and generate a control v_k according to the distribution $\nu_{\ell, k}(v \mid j_k)$, $v \in U(j_k)$.

(2) Update the (i, u)th component of Q_k according to

$$Q_{k+1}(i, u) = (1 - \gamma_{\ell, k}) Q_k(i, u) + \gamma_{\ell, k}\big(g(i, u, j_k) + \alpha \min\{J_k(j_k), Q_k(j_k, v_k)\}\big). \tag{4.5}$$

Leave all other components of Q_k unchanged: $Q_{k+1}(i, u) = Q_k(i, u)$ for all $(i, u) \notin R_k$.

In the preceding algorithm (Algorithm II), the successor state-control pair (j_k, v_k) corresponding to the different pairs $\ell = (i, u) \in R_k$ are different random variables. We have used the same notation for simplicity. Compared with Algorithm I, the chief difference in Algorithm II is that it allows multiple components of J and Q to be updated at each iteration. Compared with the deterministic asynchronous version (3.3)–(3.4), the chief difference is that selected components of Q are updated using a single sample in place of the expected value that defines $F_{J_k, \nu_{\ell, k}}$ (see Equations (3.3) and (3.5)). Such updates must satisfy certain properties, to be discussed in what follows, so that the error due to simulation noise will vanish in the limit.

It is convenient to express the next algorithm (Algorithm III) explicitly as an algorithm that operates in the joint space of the pair (J, Q). We denote $x_k = (J_k, Q_k)$ and introduce outdated information in updating x_k. This is natural for asynchronous distributed computation, in which case each component ℓ may be associated with a processor, which keeps at time k a local, outdated version of x_k, denoted by $x_k^{(\ell)}$. We introduce outdated information not just for more generality, but also to facilitate the association with the algorithmic framework of Tsitsiklis [42], which we will use in our convergence proof. In particular, $x_k^{(\ell)}$ has the form

$$x_k^{(\ell)} = (x_{1,\tau_{1,k}^\ell}, \dots, x_{m,\tau_{m,k}^\ell}), \tag{4.6}$$

where the nonnegative difference $k - \tau_{j,k}^\ell$ indicates a "communication delay" relative to the "current" time k for the jth component of x at the processor updating component ℓ ($j, \ell = 1, \dots, m$, with m being the sum of the number of states and the number of state-control pairs). We write $x_k^{(\ell)}$ in terms of its components J and Q as

$$x_k^{(\ell)} = (J_k^{(\ell)}, Q_k^{(\ell)}). \tag{4.7}$$

We will require later that $\lim_{k\to\infty} \tau_{j,k}^\ell = \infty$ for all ℓ and j, but the exact values of $\tau_{j,k}^\ell$ are immaterial and need not even be known to the processor.

In the following Algorithm III, we can use outdated information to update J and Q, and the choice of the policy ν at time k may depend on the successor state j_k in addition to the history of the algorithm up to time k. To be more precise, let I_k be an information vector, a random variable that consists of the entire history of the algorithm up to time k (this includes the stepsizes $\gamma_{\ell,t}$, the index sets S_t and R_t selected for cost and Q-factor updates, the results of the updates, and the delays $t - \tau_{j,t}^\ell$, at all times $t \le k$). We will assume that the selection of the policy is based on (I_k, j_k), where j_k is the successor state generated according to probabilities $p_{ij}(u)$, similar to Algorithm II.

Model-Free Optimistic Policy Iteration III:

Select a subset R_k of state-control pairs, and a subset of states S_k such that $R_k \cup S_k \neq \varnothing$. For each $\ell \in R_k \cup S_k$, choose a stepsize $\gamma_{\ell,k} \in (0,1]$ and times $\tau_{j,k}^\ell \le k$, $j = 1, \dots, m$. Let $(J_k^{(\ell)}, Q_k^{(\ell)})$ be as defined in Equations (4.6) and (4.7). Update J_k according to

$$J_{k+1}(i) = \begin{cases} (1-\gamma_{\ell,k})J_k(i) + \gamma_{\ell,k} \min\limits_{u\in U(i)} Q_k^{(\ell)}(i,u), \quad \text{with } \ell = i, & \text{if } i \in S_k, \\ J_k(i) & \text{if } i \notin S_k. \end{cases} \tag{4.8}$$

For each $\ell = (i,u) \in R_k$:

(1) Generate a successor state $j_{\ell,k}$ according to the distribution $p_{ij}(u)$, $j = 1, \dots, n$. Select a policy $\nu_{\ell, I_k, j_{\ell,k}}$ based on the information $(I_k, j_{\ell,k})$, and generate a control $v_{\ell,k}$ according to the distribution $\nu_{\ell, I_k, j_{\ell,k}}(v \mid j_{\ell,k})$, $v \in U(j_{\ell,k})$.

(2) Update the (i,u)th component of Q_k according to

$$Q_{k+1}(i,u) = (1-\gamma_{\ell,k})Q_k(i,u) + \gamma_{\ell,k}\big(g(i,u,j_{\ell,k}) + \alpha \min\{J_k^{(\ell)}(j_{\ell,k}), Q_k^{(\ell)}(j_{\ell,k}, v_{\ell,k})\}\big). \tag{4.9}$$

Leave all other components of Q_k unchanged: $Q_{k+1}(i,u) = Q_k(i,u)$ for all $(i,u) \notin R_k$.

In Algorithms II and III, we typically let S_k be empty for many iterations and choose S_k to be a small set when it is nonempty, so that we update the components J less frequently than the components Q. (This is consistent with our experiments in §5.) Like Algorithm I, when compared with their classical Q-learning counterparts (see Tsitsiklis [42]), Algorithms II and III also have the advantage of lower overhead per iteration, because the minimization of Q-factors over a full set of feasible controls is performed infrequently.

4.2. A general algorithmic model. As preparation for an analytically more convenient description of Algorithm III, we introduce some notation. Let $\mathcal{M}$ denote the set of all stationary (deterministic or randomized) policies. For each $\nu \in \mathcal{M}$, define an operator L^ν on the space of (J, Q) by

$$(\tilde{J}, \tilde{Q}) = L^\nu(J, Q), \tag{4.10}$$

where

$$\tilde{J}(i) = \min_{u\in U(i)} Q(i,u), \quad i = 1, \dots, n, \quad \tilde{Q} = F_{J,\nu} Q. \tag{4.11}$$

Denote the ℓth component of the mapping L^ν by L_ℓ^ν, where $\ell = 1, \dots, m$. As can be seen from Equation (4.11), if ℓ corresponds to the ith component of J, then $L_\ell^\nu(J,Q) = \min_{u\in U(i)} Q(i,u)$, whereas if ℓ corresponds to the (i,u)th component of Q, then $L_\ell^\nu(J,Q) = (F_{J,\nu}Q)(i,u)$.

We note that for a given $\ell = (i, u) \in R_k$, the policy $\nu_{\ell, I_k, j_{\ell,k}}$ is a measurable $\mathcal{M}$-valued random variable with respect to the σ-field $\sigma(I_k, j_{\ell,k})$ generated by $(I_k, j_{\ell,k})$ (because it is selected with knowledge of $(I_k, j_{\ell,k})$). We introduce the $\sigma(I_k)$-measurable $\mathcal{M}$-valued random variable $\bar\nu_{\ell, I_k} = \{\bar\nu_{\ell, I_k}(v \mid j) \mid v \in U(j), j = 1, \dots, n\}$, which is the conditional distribution of v corresponding to the joint distribution $P(j_{\ell,k} = j,\ v_{\ell,k} = v \mid I_k)$, i.e.,

$$P(j_{\ell,k} = j, v_{\ell,k} = v \mid I_k) = p_{ij}(u)\bar\nu_{\ell, I_k}(v \mid j), \quad \forall\, v \in U(j), \quad j = 1, \dots, n. \tag{4.12}$$

(If $\ell = (i, u)$ and j is such that $p_{ij}(u) = 0$, we have $P(j_{\ell,k} = j,\ v_{\ell,k} = v \mid I_k) = 0$ for all $v \in U(j)$, and we may define $\bar\nu_{\ell, I_k}(v \mid j)$ to be any distribution over $U(j)$, for example, the uniform distribution.) Note that if in Algorithm III $\nu_{\ell, I_k, j_{\ell,k}}(\cdot \mid j)$ is chosen before $j_{\ell,k}$ is generated, then $\bar\nu_{\ell, I_k}$ coincides with $\nu_{\ell,k}$; this is the case in Algorithm II.

We can now express Algorithm III in a compact form using the mappings L^ν of Equations (4.10) and (4.11). It can be equivalently written as

$$x_{\ell, k+1} = (1 - \gamma_{\ell,k})x_{\ell,k} + \gamma_{\ell,k}\big(L_\ell^{\bar\nu_{\ell, I_k}}(x_k^{(\ell)}) + w_{\ell,k}\big), \tag{4.13}$$

where

(a) if $\ell = (i, u) \in R_k$, we have $\gamma_{\ell,k} \in (0, 1]$, and $w_{\ell,k}$ is a noise term given by

$$w_{\ell,k} = g(i, u, j_{\ell,k}) + \alpha \min\{J_k^{(\ell)}(j_{\ell,k}), Q_k^{(\ell)}(j_{\ell,k}, v_{\ell,k})\} - \big(F_{J_k^{(\ell)}, \bar\nu_{\ell, I_k}} Q_k^{(\ell)}\big)(i, u) \tag{4.14}$$

(see Equations (4.9) and (4.11), and noticing that $L_\ell^{\bar\nu_{\ell, I_k}}(x_k^{(\ell)}) = (F_{J_k^{(\ell)}, \bar\nu_{\ell, I_k}} Q_k^{(\ell)})(i, u)$);

(b) if $\ell \in S_k$, we have $\gamma_{\ell,k} \in (0, 1]$, $w_{\ell,k} = 0$, and $\bar\nu_{\ell, I_k}$ is immaterial (see Equations (4.8) and (4.11));

(c) if $\ell \notin R_k \cup S_k$, we have $\gamma_{\ell,k} = 0$ and $w_{\ell,k} = 0$.

With $\gamma_{\ell,k}$ defined for all ℓ and k, the sets R_k, S_k may also be specified implicitly by those $\gamma_{\ell,k}$ that are positive.

4.3. Convergence analysis. In our convergence analysis of the general algorithm (4.8)–(4.9), equivalently given in (4.13)–(4.14), we use extensions of two results from Tsitsiklis [42]. The latter work analyzed boundedness and convergence of algorithms of the form (4.13) with the exception that there is only a single contraction mapping L in place of $L^{\bar\nu_{\ell, I_k}}$. The results of Tsitsiklis [42], however, extend to the case with multiple mappings, if these mappings are contraction mappings with respect to the same norm and have the same fixed point. (Such an extension is given in Bertsekas and Tsitsiklis [11, Section 4.3] for the case without communication delays.)

Thus, the first step of our convergence proof is to establish a common contraction property of L^ν for all stationary policies ν. Define a weighted sup-norm $\|\cdot\|_\zeta$ on the space of (J, Q) by

$$\|(J, Q)\|_\zeta = \max\left\{\frac{\|J\|_\infty}{\xi}, \|Q\|_\infty\right\}, \tag{4.15}$$

where ξ is a positive scalar such that

$$\xi > 1, \quad \alpha\xi < 1. \tag{4.16}$$

Proposition 4.1. *Let $\|\cdot\|_\zeta$ and ξ be given by Equations (4.15) and (4.16), respectively, and let $\beta = \max\{\alpha\xi, 1/\xi\} < 1$. For all stationary policies ν, (J^*, Q^*) is the unique fixed point of the mapping L^ν given by Equations (4.10)–(4.11), and we have*

$$\|L^\nu(J, Q) - L^\nu(J', Q')\|_\zeta \le \beta\|(J, Q) - (J', Q')\|_\zeta \tag{4.17}$$

for all pairs (J, Q) and (J', Q').

Proof. At the beginning of the proof of Proposition 2.2 we showed that (J^*, Q^*) is a fixed point of L^ν for all ν. The uniqueness of the fixed point will be implied by Equation (4.17), which we now prove. Let $(\tilde J, \tilde Q) = L^\nu(J, Q)$ and $(\tilde J', \tilde Q') = L^\nu(J', Q')$. By Proposition 2.1, we have

$$\begin{aligned}\|\tilde Q - \tilde Q'\|_\infty &\le \alpha \max\{\|J - J'\|_\infty, \|Q - Q'\|_\infty\}\\ &= \alpha \max\left\{\xi \cdot \frac{\|J - J'\|_\infty}{\xi}, \|Q - Q'\|_\infty\right\}\\ &\le \alpha \max\left\{\xi \cdot \frac{\|J - J'\|_\infty}{\xi}, \xi \cdot \|Q - Q'\|_\infty\right\}\\ &= \alpha\xi \cdot \|(J, Q) - (J', Q')\|_\zeta,\end{aligned} \tag{4.18}$$

where we used $\xi > 1$ to derive the second inequality. We also have

$$\|\tilde{J} - \tilde{J}'\|_\infty \le \|Q - Q'\|_\infty,$$

which implies that

$$\begin{aligned}\frac{\|\tilde{J} - \tilde{J}'\|_\infty}{\xi} &\le \frac{1}{\xi} \cdot \|Q - Q'\|_\infty \\ &\le \frac{1}{\xi} \cdot \max\left\{ \frac{\|J - J'\|_\infty}{\xi}, \|Q - Q'\|_\infty \right\} \\ &= \frac{1}{\xi} \cdot \|(J, Q) - (J', Q')\|_\zeta. \end{aligned} \tag{4.19}$$

Equations (4.18) and (4.19) imply the desired property (4.17):

$$\begin{aligned}\|(\tilde{J}, \tilde{Q}) - (\tilde{J}', \tilde{Q}')\|_\zeta &= \max\left\{ \frac{\|\tilde{J} - \tilde{J}'\|_\infty}{\xi}, \|\tilde{Q} - \tilde{Q}'\|_\infty \right\} \\ &\le \max\{\alpha\xi, 1/\xi\} \cdot \|(J, Q) - (J', Q')\|_\zeta \\ &= \beta \|(J, Q) - (J', Q')\|_\zeta. \quad \square \end{aligned}$$

We now specify conditions on the variables involved in the algorithm (4.13)–(4.14). Our conditions parallel the assumptions given in Tsitsiklis [42, Assumptions 1–3], which are standard for asynchronous stochastic approximation. We use the shorthand "w.p.1" for "with probability 1." The first condition is a mild, natural requirement for the delays.

Condition 4.1. *For every ℓ and j, $\lim_{k\to\infty} \tau^\ell_{j,k} = \infty$ w.p.1.*

The next condition is mainly about the noise terms $w_{\ell,k}$. Let $(\Omega, \mathcal{F}, P)$ be the common probability space on which all the random variables involved in the algorithm are defined, and let $\{\mathcal{F}_k, k \ge 0\}$ be an increasing sequence of subfields of $\mathcal{F}$.

Condition 4.2. (a) *x_0 is $\mathcal{F}_0$-measurable.*
(b) *For every ℓ corresponding to a component of Q and every k, $w_{\ell,k}$ is $\mathcal{F}_{k+1}$-measurable.*
(c) *For every j, ℓ, and k, $\gamma_{\ell,k}$, $\tau^\ell_{j,k}$ and $\bar{\nu}_{\ell,I_k}$ are $\mathcal{F}_k$-measurable.*
(d) *For every ℓ corresponding to a component of Q and every k,*

$$E[w_{\ell,k} \mid \mathcal{F}_k] = 0.$$

(e) *There exist (deterministic) constants A and B such that for every ℓ corresponding to a component of Q and every k,*

$$E[w^2_{\ell,k} \mid \mathcal{F}_k] \le A + B \max_j \max_{\tau \le k} |x_{j,\tau}|^2.$$

The next condition deals with the stepsize variables.

Condition 4.3. (a) *For every ℓ,*

$$\sum_{k\ge 0} \gamma_{\ell,k} = \infty, \quad w.p.1.$$

(b) *For every ℓ corresponding to a component of Q,*

$$\sum_{k\ge 0} \gamma^2_{\ell,k} < \infty, \quad w.p.1.$$

Condition 4.3(a) implies that all components of J and Q are updated infinitely often, which is also part of the assumptions of Proposition 3.1. A simple way to choose stepsize sequences $\{\gamma_{\ell,k}\}$ that satisfy Condition 4.3 is to define them using a positive scalar sequence $\{\gamma_k\}$ which diminishes to 0 at a suitable rate (e.g., $O(1/k)$): For all $\ell \in R_k$, let $\gamma_{\ell,k}$ have a common value γ_k, and select all state-control pairs (i, u) "comparably often" in the sense that the fraction of times (i, u) is selected for iteration is nonzero in the limit (see Borkar [17]).

There are two insignificant differences between the preceding conditions and the assumptions in Tsitsiklis [42, Assumptions 1–3]. First, Condition 4.2(c) is imposed on the random variables $\bar{\nu}_{\ell,I_k}$, which do not appear in

Tsitsiklis [42]. Second, Conditions 4.2(d) and 4.2(e) are imposed on the noise terms $w_{\ell,k}$, which are involved in the updates of components of Q only (for components of J, there is no noise ($w_{\ell,k}=0$) in the updates and these conditions are trivially satisfied). For the same reason, Condition 4.3(b), a standard condition for bounding asymptotically the error due to noise, is also imposed on the components of Q only (in Tsitsiklis [42], Condition 4.3(b) is imposed on all components of x).[4]

We now verify that by its definition, the algorithm (4.13)–(4.14) satisfies Condition 4.2. Let $\mathcal{F}_k=\sigma(I_k)$. Then Conditions 4.2(a)–(c) are satisfied by the definition of the algorithm; in particular, note that $\bar{\nu}_{\ell,I_k}$ is, by definition, $\mathcal{F}_k$-measurable (see Equation (4.12)). We verify Conditions 4.2(d) and (e), similar to the standard Q-learning case given in Tsitsiklis [42]. Let $\ell=(i,u)\in R_k$ and $(j_{\ell,k},v_{\ell,k})$ be the corresponding successor state-control pair. From the way $j_{\ell,k}$ is generated, it is seen that

$$E[g(i,u,j_{\ell,k})\mid \mathcal{F}_k]=\sum_{j=1}^n p_{ij}(u)g(i,u,j).$$

From the way $(j_{\ell,k},v_{\ell,k})$ is generated and the definition of $\bar{\nu}_{\ell,I_k}$ (see Equation (4.12)), we have

$$E\big[\min\big\{J_k^{(\ell)}(j_{\ell,k}),\,Q_k^{(\ell)}(j_{\ell,k},v_{\ell,k})\big\}\mid \mathcal{F}_k\big]=\sum_{j=1}^n p_{ij}(u)\sum_{v\in U(j)}\bar{\nu}_{\ell,I_k}(v\mid j)\min\big\{J_k^{(\ell)}(j),\,Q_k^{(\ell)}(j,v)\big\}.$$

Taking conditional expectation in Equation (4.14) and using the preceding two equations, we obtain

$$\begin{aligned}E[w_{\ell,k}\mid \mathcal{F}_k]&=\sum_{j=1}^n p_{ij}(u)\Bigg(g(i,u,j)+\alpha\sum_{v\in U(j)}\bar{\nu}_{\ell,I_k}(v\mid j)\min\{J_k^{(\ell)}(j),\,Q_k^{(\ell)}(j,v)\}\Bigg)-\big(F_{J_k^{(\ell)},\bar{\nu}_{\ell,I_k}}Q_k^{(\ell)}\big)(i,u)\\&=0,\end{aligned}$$

so Condition 4.2(d) is satisfied. It can also be seen that we may write $w_{\ell,k}=Z_1+Z_2$ with

$$\begin{aligned}Z_1&=g(i,u,j_{\ell,k})-E[g(i,u,j_{\ell,k})\mid \mathcal{F}_k],\\Z_2&=\alpha\min\big\{J_k^{(\ell)}(j_{\ell,k}),\,Q_k^{(\ell)}(j_{\ell,k},v_{\ell,k})\big\}-E\big[\alpha\min\big\{J_k^{(\ell)}(j_{\ell,k}),\,Q_k^{(\ell)}(j_{\ell,k},v_{\ell,k})\big\}\mid \mathcal{F}_k\big],\end{aligned}$$

where the first expectation is over $j_{\ell,k}$ and the second is over $(j_{\ell,k},v_{\ell,k})$. Since the number of state-control pairs is finite, the variance of $g(i,u,j_{\ell,k})$ can be bounded by a constant C for all (i,u): $E[Z_1^2\mid \mathcal{F}_k]\le C$.[5] The conditional variance of $\min\{J_k^{(\ell)}(j_{\ell,k}),\,Q_k^{(\ell)}(j_{\ell,k},v_{\ell,k})\}$, conditioned on $\mathcal{F}_k$, is bounded by the square of the largest absolute value that this random variable can possibly take, so

$$E[Z_2^2\mid \mathcal{F}_k]\le \alpha^2\max_j\max_{\tau\le k}|x_{j,\tau}|^2.$$

Thus, using also the Cauchy-Schwarz inequality, we have

$$\begin{aligned}E[w_{\ell,k}^2\mid \mathcal{F}_k]&\le C+\alpha^2\max_j\max_{\tau\le k}|x_{j,\tau}|^2+2\sqrt{C\cdot\alpha^2\max_j\max_{\tau\le k}|x_{j,\tau}|^2}\\&\le A+B\max_j\max_{\tau\le k}|x_{j,\tau}|^2,\quad \forall\,k,\quad \ell\in R_k,\end{aligned}$$

for some deterministic constants A and B, so Condition 4.2(e) is satisfied.

Proposition 4.2. *Under Conditions* 4.1 *and* 4.3, *any sequence* $\{x_k\}$ *with* $x_k=(J_k,Q_k)$ *generated by the model-free optimistic PI algorithm* (4.13)–(4.14) (*or equivalently,* (4.8)–(4.9)) *converges to* $x^*=(J^*,Q^*)$ *with probability* 1.

[4] Assumption 3(b) in Tsitsiklis [42] is slightly different than Condition 4.3(b): it is $\sum_{k\ge 0}\gamma_{\ell,k}^2<C$ w.p.1, for some deterministic constant C, instead of C being ∞. However, this change in the stepsize condition only affects one technical lemma (Lemma 1) in Tsitsiklis [42], and by strengthening that lemma so that its conclusions hold under the weaker condition 4.3(b), the rest of the analysis given in Tsitsiklis [42] remains essentially intact under the latter condition. The additional analysis just mentioned can be found in Bertsekas and Tsitsiklis [11] (Proposition 4.1 and Example 4.3, pp. 141–143; see also Section 4.3.6 therein). In this paper, to keep the references brief, we will not repeat the above in our citations, and we will only refer to Tsitsiklis [42] for various conclusions that originally appeared there and now hold under the weaker stepsize Condition 4.3(b).

[5] If instead of a scalar, $g(i,u,j)$ is also treated as random, then one may impose a finite variance condition on it.

Proof. We have shown that Condition 4.2 is satisfied by the algorithm (4.13)–(4.14), so under the assumption of the proposition, we have that all Conditions 4.1–4.3 hold. We apply the analysis of Tsitsiklis [42], and in particular, the proofs of Theorems 1 and 3 of that reference. The two theorems imply the boundedness of $\{x_k\}$ and the convergence of $\{x_k\}$ to x^* with probability 1, respectively, for iterates of the form

$$x_{\ell,k+1} = (1-\gamma_{\ell,k})x_{\ell,k} + \gamma_{\ell,k}\big(L_\ell(x_k^{(\ell)}) + w_{\ell,k}\big),$$

where L is a contraction mapping with fixed point x^*, under assumptions that parallel Conditions 4.1–4.3 with minor differences, which we address below (in our algorithm, there are multiple contraction mappings L^ν that share the same fixed point, and the Condition 4.3(b) is satisfied only for ℓ corresponding to components of Q).

First, for a contraction mapping L with modulus β and with respect to a weighted sup-norm $\|\cdot\|_\zeta$, L enters in the proofs of Theorems 1 and 3 of Tsitsiklis [42], only via the two inequalities

$$\|L(x)\|_\zeta \le \beta\|x\|_\zeta + D, \quad \forall\ x, \tag{4.20}$$

where D is some constant, and

$$\|L(x)-x^*\|_\zeta \le \beta\|x-x^*\|_\zeta, \quad \forall\ x. \tag{4.21}$$

Implications of these inequalities are used to bound $L_\ell(x_k^{(\ell)})$ in the iterates $x_{\ell,k+1}$ for each sample path from a set of probability one.

Second, in the proofs of Theorems 1 and 3 of Tsitsiklis [42], the effect of the noise $\{w_{\ell,k}\}$ on $\{x_{\ell,k}\}$ for each component ℓ is analyzed in two lemmas, Lemmas 1 and 2, under Conditions 4.2(b)–(e) and 4.3 for that particular component. It is only in those two places that Condition 4.3(b) for a component is used. The rest of the analysis for Theorems 1 and 3 relies only indirectly on Condition 4.3(b) through the two lemmas.

In our case, the inequalities (4.20) and (4.21) are satisfied by all $L^{\bar{\nu}_{\ell,t_k}}$ for the same $\|\cdot\|_\zeta, \beta, D$, and $x^* = (J^*, Q^*)$, as established in Proposition 4.1. Moreover, when ℓ corresponds to a component of J, although the stepsizes $\gamma_{\ell,k}$ are not restricted by Condition 4.3(b), because the noise terms $w_{\ell,k}, k \ge 0$ are always zero, Lemmas 1 and 2 of Tsitsiklis [42] trivially hold without Condition 4.3(b) for such ℓ. It then follows that Lemmas 1 and 2 hold for all components ℓ of x in our case. We can thus apply the proofs of the two theorems of Tsitsiklis [42] with $L_\ell(x_k^{(\ell)})$ replaced by $L_\ell^{\bar{\nu}_{\ell,t_k}}(x_k^{(\ell)})$ to establish the convergence to x^* with probability 1 for the sequence $\{x_k\}$ generated by the algorithm (4.13)–(4.14). □

5. Computational experiments.

In this section we illustrate the behavior of our algorithms of §§3 and 4 with three numerical examples. In summary, our experiments confirm the results of the theoretical analysis. In particular:

(1) Our asynchronous PI algorithms of §3 converge under conditions where the classical algorithm fails.

(2) Our Q-learning algorithms of §4 exhibit comparable convergence (in terms of number of iterations) to the standard Q-learning algorithm, with substantially less overhead each time Q_k is updated (because the minimization involved is simpler).

5.1. Williams and Baird [49] counterexample.

Williams and Baird [49] provided several examples in which the initial conditions and the order of updating the components (i.e., the sets R_k and S_k) are chosen so that the sequence of Q-factors generated by the asynchronous modified PI algorithm (3.1)–(3.2) oscillates and fails to converge. In Figure 1 we compare the behavior of three asynchronous algorithms for one such example. This is Example 2 of Williams and Baird [49], which involves six states and two controls, with the discount factor equal to 0.9; for a description of this example that is adapted to the Q-learning format of the present paper, see Bertsekas [7]. The three algorithms are the algorithm (3.1)–(3.2), the algorithm (3.3)–(3.4), and the nonstochastic version of the Q-learning algorithm (1.4). All of them follow the same order of updates and use the same subsets R_k and S_k as described by the example, except that the nonstochastic Q-learning algorithm does not use S_k because it operates on Q-factors only. Our experiments with algorithm (3.3)–(3.4) involved two special choices of the policies ν_k, yielding two variant algorithms. The first variant is the one of Equations (3.5) and (3.6), and its updates are shown in the second column. The second variant involves a deterministic policy ν_k selected randomly according to the uniform distribution, and its updates are shown in the third column.

Figure 1 shows the values of Q_k for a fixed state-control pair, which is indicated at the beginning of each row. It can be seen that our algorithm converges as predicted by the theoretical analysis, and so does Q-learning.

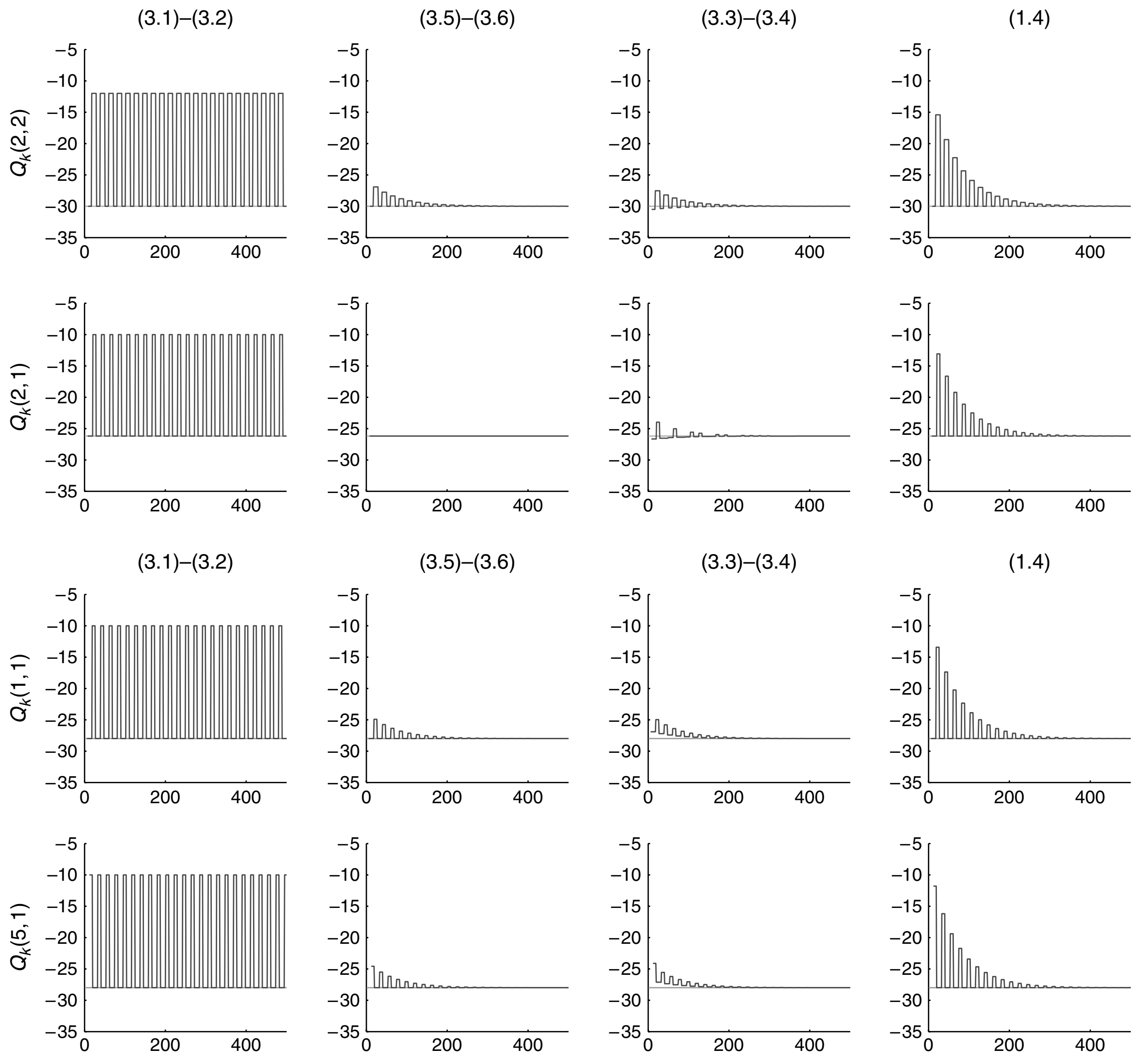

FIGURE 1. Illustration of performance on Example 2 of Williams and Baird [49] for the algorithm (3.1)–(3.2), the algorithm (3.5)–(3.6), the algorithm (3.3)–(3.4) with random selection of ν_k, and the nonstochastic version of the Q-learning algorithm (1.4).
Note. The plots give the values of four Q-factors as functions of iteration number, with the desired limit values indicated by horizontal lines.

5.2. Dynamic location example. We next compare the stochastic optimistic PI algorithms of §4 on a dynamic location problem adapted from the book by Puterman [35, Problem 3.17, pp. 70–71, attributed to Rosenthal et al. [36]]. A repairman moves between 10 sites according to certain stationary transition probabilities, and a trailer carrying supplies to the repairman may be relocated to any of the sites. The problem is to dynamically relocate the trailer, with knowledge of the locations of the repairman and the trailer, so as to minimize the expected discounted total cost. We chose the discount factor to be 0.98. We define the one-stage costs as follows: at each stage, if the repairman and the trailer are at sites d_r and d_e, respectively, and the trailer is moved to site $\tilde{d}_e$, then the cost is $|d_r - d_e| + |d_e - \tilde{d}_e|/2$. Regarding the repairman's transition probabilities, if the repairman is at site d_r, he next moves to any site $d_r \leq d \leq 10$ with equal probability unless $d_r = 10$, in which case he moves to site 1 with probability 3/4 and stays at site 10 with probability 1/4.

In this problem there are 10^2 states (d_r, d_e), $d_r, d_e = 1, \ldots, 10$, corresponding to the locations of the repairman and the trailer, and 10 controls $\tilde{d}_e = 1, \ldots 10$, corresponding to the next location of the trailer. So there are 10^3 Q-factors, which we denote by $Q((d_r, d_e), \tilde{d}_e)$. Because the movement of the repairman is uncontrolled, if he moves from site d_r to $\tilde{d}_r$, this transition can be used to update simultaneously the Q-factors $Q((d_r, d_e), \tilde{d}_e)$ for all possible locations and moves of the trailer, $d_e, \tilde{d}_e = 1, \ldots, 10$. The simulation results we present below are obtained in this way. In particular, we simulate a single trajectory of sites $s_0, s_1, \ldots$ visited by the repairman. Simultaneously, as the trajectory is being generated, we apply the optimistic PI algorithm II

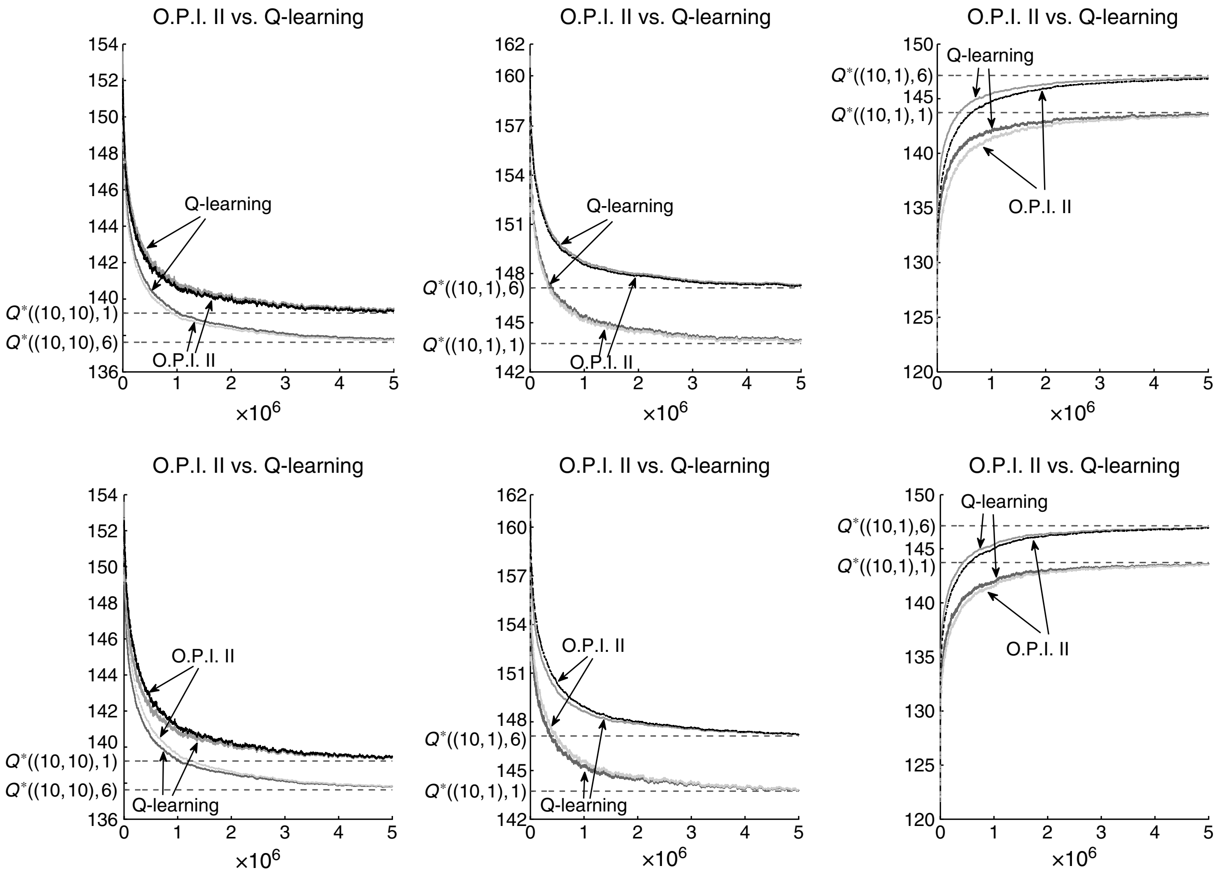

FIGURE 2. Comparison of the optimistic PI algorithm II and Q-learning for the dynamic location problem.
Note. The two rows correspond to two variants of the algorithm resulting from different choices of the policies $\nu_{\ell,k}$, with the second choice involving more exploration than the first one.

(Equations (4.4) and (4.5)), in which, for updating Q-factors at iterations $k = 0, 1, \ldots,$ we let the set R_k of state-control pairs be $\{(s_k, d_e, \tilde{d}_e) \mid d_e, \tilde{d}_e = 1, \ldots, 10\}$, whereas we update J_k once every 50 iterations, with the corresponding set R_k of states being $\{(s_k, d_e) \mid d_e = 1, \ldots, 10\}$. We use the stepsize $\gamma_{\ell,k} = (10+k)^{-0.55}$.

Figure 2 compares the algorithm (4.4)–(4.5) with ordinary Q-learning. Both algorithms use the same stepsize sequence, the same trajectory of the repairman's move, and the same set R_k for the block of Q-factors to be updated at each iteration. Each subfigure corresponds to a simulation run and shows the values of Q_k at two state-control pairs generated by the two algorithms. The two pairs consist of the same state with two different controls, one being optimal and the other nonoptimal for that state. Together with the true values, they are indicated on the vertical axis of each subfigure. The optimistic PI algorithm II is designated by "O.P.I. II." In Figure 2 we present results with two variants of the algorithm, which differ in the choice of the policy $\nu_{\ell,k}$. In the first variant (shown in the top row of Figure 2), $\nu_{\ell,k} = \nu_k$, a deterministic policy, initially chosen randomly and maintained throughout iterations. Its components $\nu_k(i), i \in S_k$ are updated to be the controls that attain the minima in Equation (4.4), whenever we update J_k. In the second variant (shown in the bottom row of Figure 2), we let $\nu_{\ell,k}$ be a deterministic policy chosen randomly according to the uniform distribution.

As the figures show, our optimistic PI algorithm behaves similar to Q-learning for both choices of $\nu_{\ell,k}$, even though it has about 90% percent less computation overhead in the minimization operations than Q-learning. We also run the algorithms with initial values Q_0 and J_0 well below the optimal. The results are shown in the third column of Figure 2, from which it can be seen that the updates of our algorithm tend to produce smaller values and increase more slowly than Q-learning. This can be attributed to the minimization operation in Equation (4.5). It is a phenomenon that may arise frequently and may be addressed by a form of interpolation between $Q(j, v)$ and $\min\{J(j), Q(j, v)\}$ that preserves the convergence of the algorithm. We refer to Bertsekas and Yu [14] for a proposal of such a device within a related context.

5.3. Automobile replacement example. We now compare the stochastic algorithms of §4 with Q-learning on the classical automobile replacement problem from Howard [27]. The problem is to decide when to replace

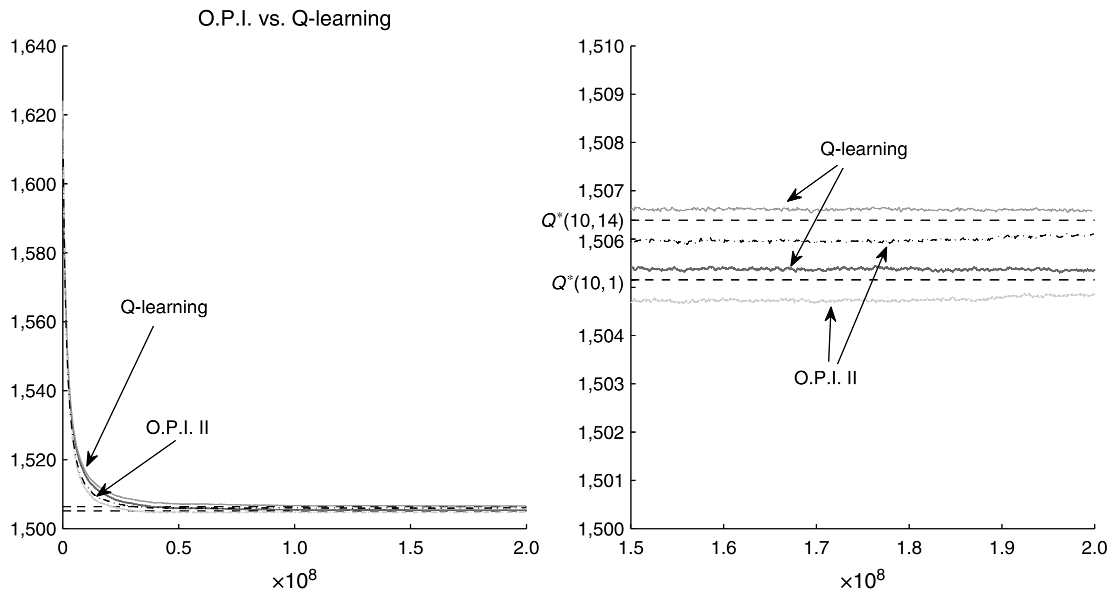

FIGURE 3. Comparison of the optimistic PI algorithm II and Q-learning for the automobile replacement problem.

a car as it ages, given that the cost and value of a car decrease with its age, whereas the operating expense and the probability of breaking down increase with its age. Decisions of whether to keep the car or to trade it for another car are made at three-month intervals. We have 41 states corresponding to the age of a car: a new car is at state/age 1, a three-month-old car at state/age 2, and so on; but if a car breaks down or if it is more than 10 year old, then it is at state/age 41. We have 41 controls: control 1 represents keeping the car, whereas controls $u = 2, \ldots, 41$ represent trading the car for a car at state/age $u - 1$, i.e., a $(u-2) \times$ three-month-old car. For our experiments, we set the discount factor $\alpha = 0.999$ and scaled down the prices/costs so that 1 unit represents \$100. We found that the optimal policy in this case is to keep the car if it is at any of the states 4–26, and to trade it for a $3\frac{1}{4}$-year-old car (state 14) otherwise.

We run the optimistic PI algorithms II and III, and the ordinary Q-learning algorithm under comparable conditions. In particular, all the algorithms have access to the prices and operating costs of cars at all ages, and they all simulate a trajectory of state-control pairs $(i_0, u_0), (i_1, u_1), \ldots,$ where i_{k+1} is generated according to the transition model $p_{ij}(u)$ with $i = i_k$, $u = u_k$, and u_k is generated by some randomized policy to be described

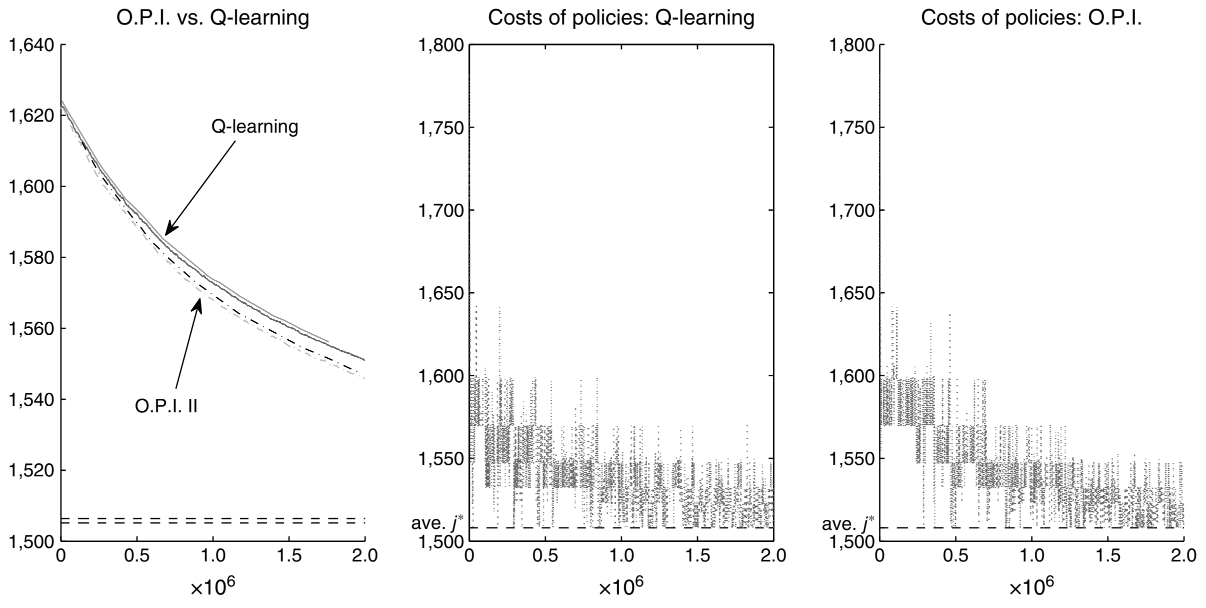

FIGURE 4. Comparison of the optimistic PI algorithm II and Q-learning at the early phase for the automobile replacement problem.

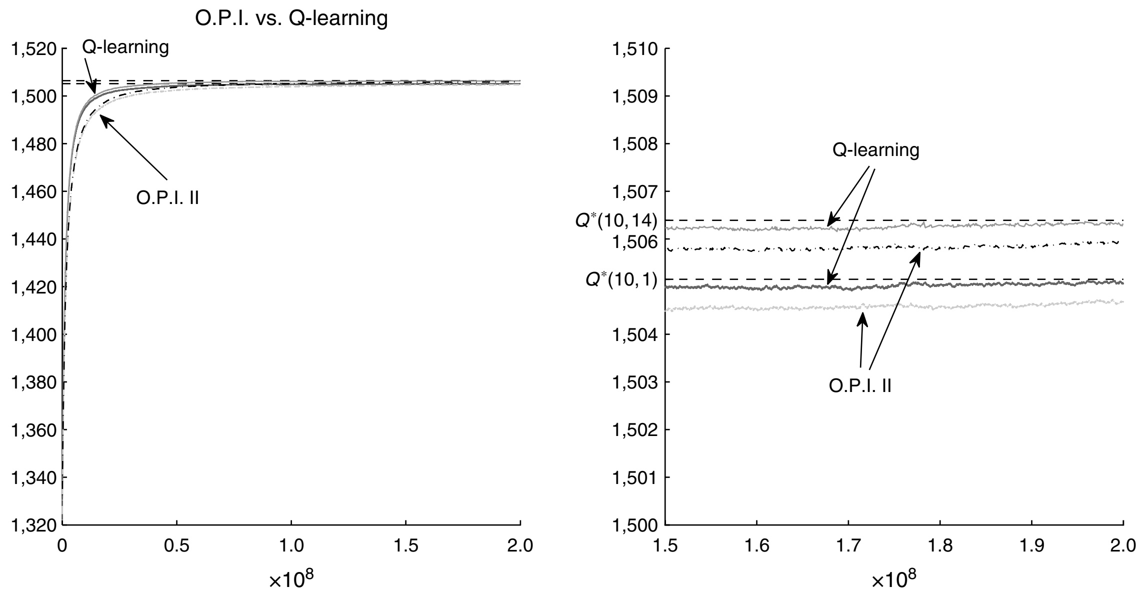

FIGURE 5. Comparison of the optimistic PI algorithm II and Q-learning for the automobile replacement problem.

shortly. To make computation more efficient, at iteration k, based on the value of (i_k, u_k), we update multiple Q-factors using the subsequent transition to i_{k+1}. More specifically, given that the control u_k instantly makes the age of the car at hand $\bar{i}$, we let the set R_k of state-control pairs, at which the algorithms update the Q-factors, to include (i) the state $\bar{i}$ with the control to keep the car, and (ii) all states i with the control to trade the car at hand for a car of age $\bar{i}$. In the Q-factor updates, we use the stepsize $\gamma_{\ell,k} = (100 + k/10^4)^{-0.8}$.

The controls u_k, $k \geq 0$, are generated as follows. All of the algorithms maintain a deterministic policy μ_k. At iteration k, $u_k = \mu_k(i_k)$ with probability 0.7, whereas with probability 0.3, u_k is chosen randomly uniformly from a set of reasonable controls (which excludes those obviously inferior decisions to trade for an older car that does not result in any instant benefit). Once every 50 iterations, the algorithms update the policy μ_k at 10 randomly chosen states and set the controls at those states to be the ones minimizing the respective $Q(i, u)$ over u. The optimistic PI algorithms also update the costs J_k at these chosen states, which form the set S_k.

In the experiments shown below, all the algorithms start with $i_0 = 41$ and the initial policy μ_0, which is to always keep the car if it is not at state/age 41, and to buy a new car only then. The initial J_0 and Q_0 are calculated using this policy and the prices/costs given by the model, assuming that a car never breaks.

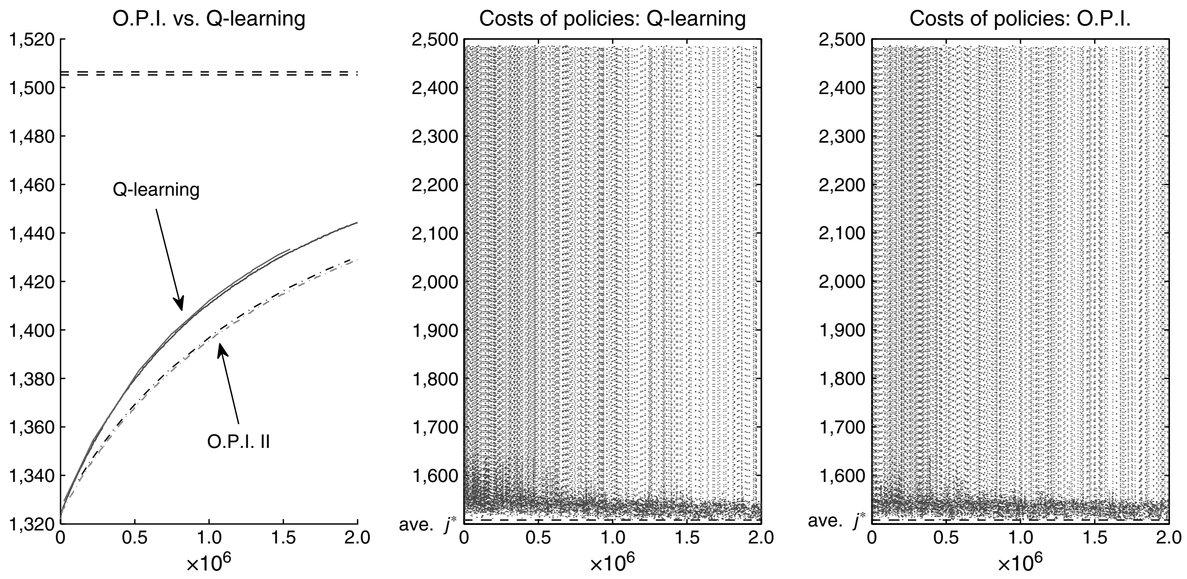

FIGURE 6. Comparison of the optimistic PI algorithm II and Q-learning at the early phase for the automobile replacement problem.

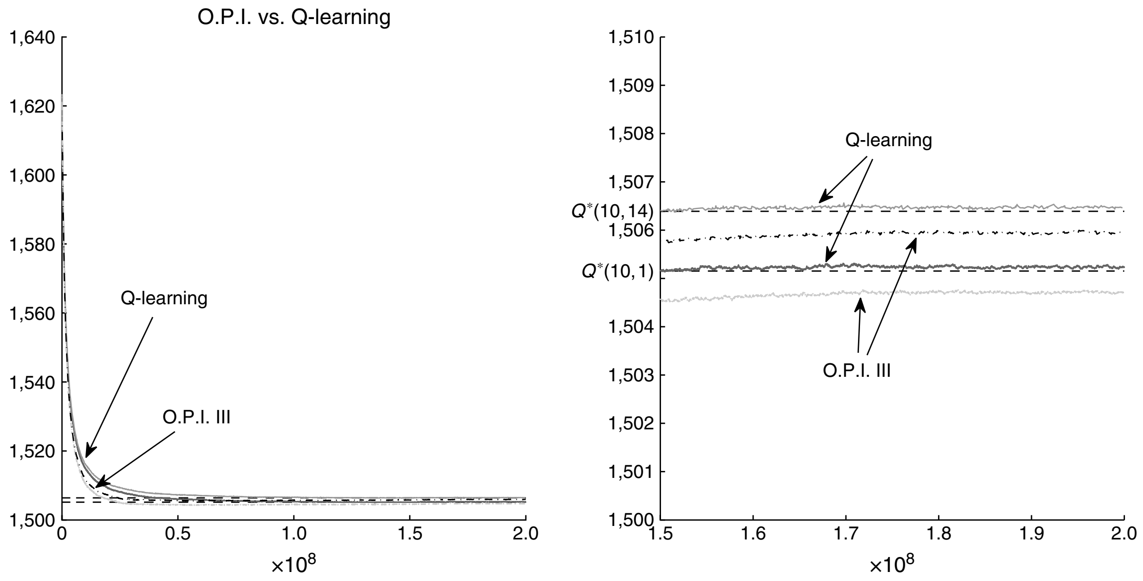

FIGURE 7. Comparison of the optimistic PI algorithm III and Q-learning for the automobile replacement problem.

Figures 3 and 4 compare Q-learning and the optimistic PI algorithm II (Equations (4.4) and (4.5), designated by "O.P.I. II" in the figures). In the latter algorithm, the policies $\nu_{\ell,k}$ used for the Q-factor updates (4.5) are μ_k. Figure 3 shows the iterates $Q_k(10,1)$ and $Q_k(10,14)$ generated by the two algorithms. (In the experiments, only $Q_k(i_k,u_k)$ are recorded, and interpolation is used to generate the curves shown in this and the following figures.) It can be seen that the optimistic PI algorithm behaves similar to Q-learning, even though in each Q-factor update, it only compares two values instead of 41 values in the minimization operation. The right subfigure shows that near convergence, the optimistic PI algorithm tends to approach the true values from below and converges more slowly than Q-learning. Again this can be attributed to the minimization operation in Equation (4.5) (see our earlier discussion on interpolation at the end of §5.2).

Figure 4 shows that during the early phase when the Q-factors are still far from the true values (shown in the left subfigure), the policies μ_k generated by both algorithms improve rapidly. In the middle and right subfigures, we plot the averaged cost $\frac{1}{41}\sum_i J_{\mu_k}(i)$ of the policies μ_k for the two algorithms; the averaged optimal value, $\frac{1}{41}\sum_i J^*(i)$, is indicated on the vertical axis. The averaged cost of the initial policy μ_0 is approximately 1,731.

We observe that this rapid policy improvement at the early phase depends on how the initial Q_0 and J_0 are chosen. For comparison, Figures 5 and 6 illustrate the behavior of both algorithms when Q_0 and J_0 are

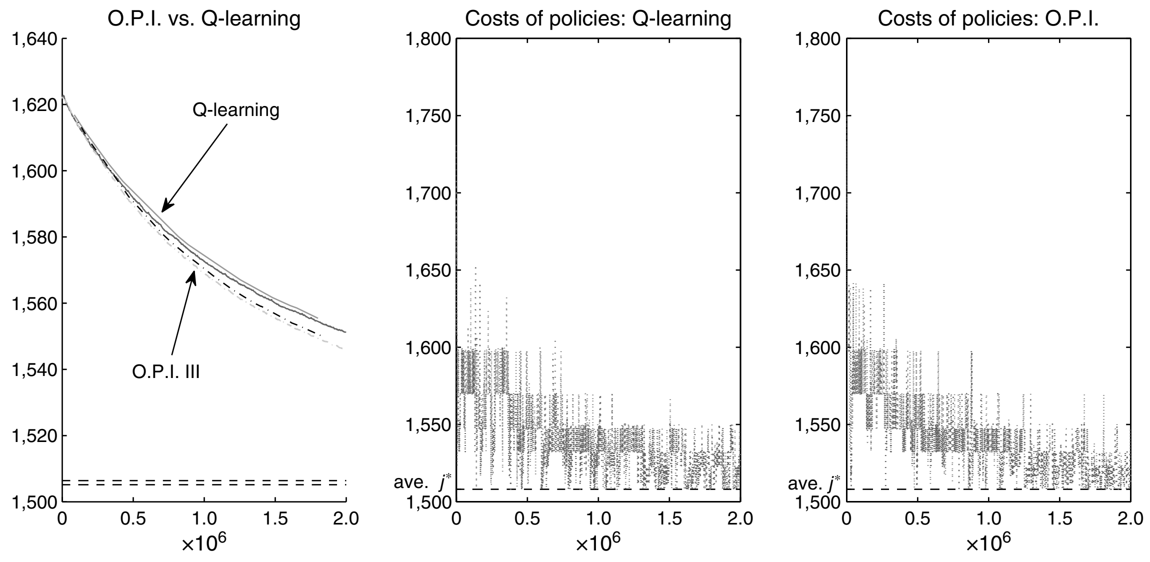

FIGURE 8. Comparison of the optimistic PI algorithm III and Q-learning at the early phase for the automobile replacement problem.

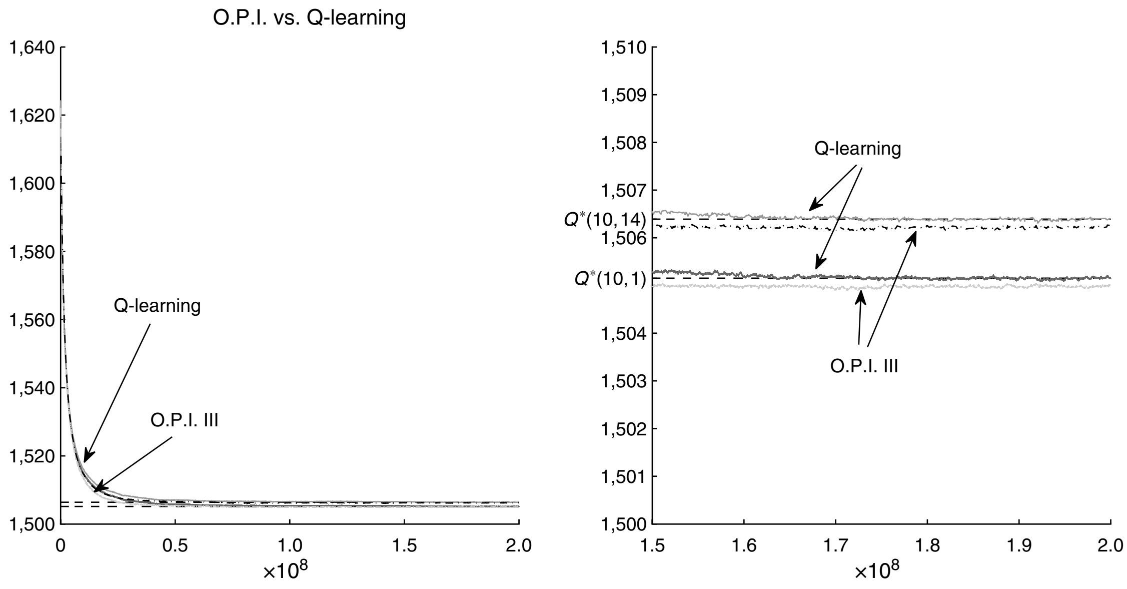

FIGURE 9. Comparison of the optimistic PI algorithm III and Q-learning for the automobile replacement problem.

shifted by a negative constant to make them well below the optimal values. Figure 6 shows wild oscillation in the performance of the policies μ_k generated by both algorithms during the early phase. The tendency of the optimistic PI algorithm to generate smaller values can also be observed in Figure 5.

We also run the same experiments to compare standard Q-learning and the optimistic PI algorithm III (Equations (4.8) and (4.9), designated by "O.P.I. III" in the figures). In the latter algorithm, we use a constant stepsize $\gamma_{\ell,k} = 0.5$ to update J_k via Equation (4.8), and we test two variants with different choices of the policies $\nu_{\ell, I_k, j_{\ell,k}}$ in the Q-factor updates (4.9). In the first variant, we set $\nu_{\ell, I_k, j_{\ell,k}}$ to be the policy $\mu_{\tilde{k}}$, $\tilde{k} < k$, prior to the most recent policy update that gives the present μ_k. The results are shown in Figures 7 and 8. In the second variant, $\nu_{\ell, I_k, j_{\ell,k}}$ depends also on $j_{\ell,k} = i_{k+1}$, the subsequent state of the car. If the latter is no less than 30, $\nu_{\ell, I_k, j_{\ell,k}} = \mu_k$; otherwise, $\nu_{\ell, I_k, j_{\ell,k}}$ is the randomized policy, which, with equal probability, follows μ_k or applies a control randomly selected from the set of reasonable controls. The results are shown in Figures 9 and 10. It can be seen that the behavior of the algorithm in both cases is similar to the one described above for Algorithm II.

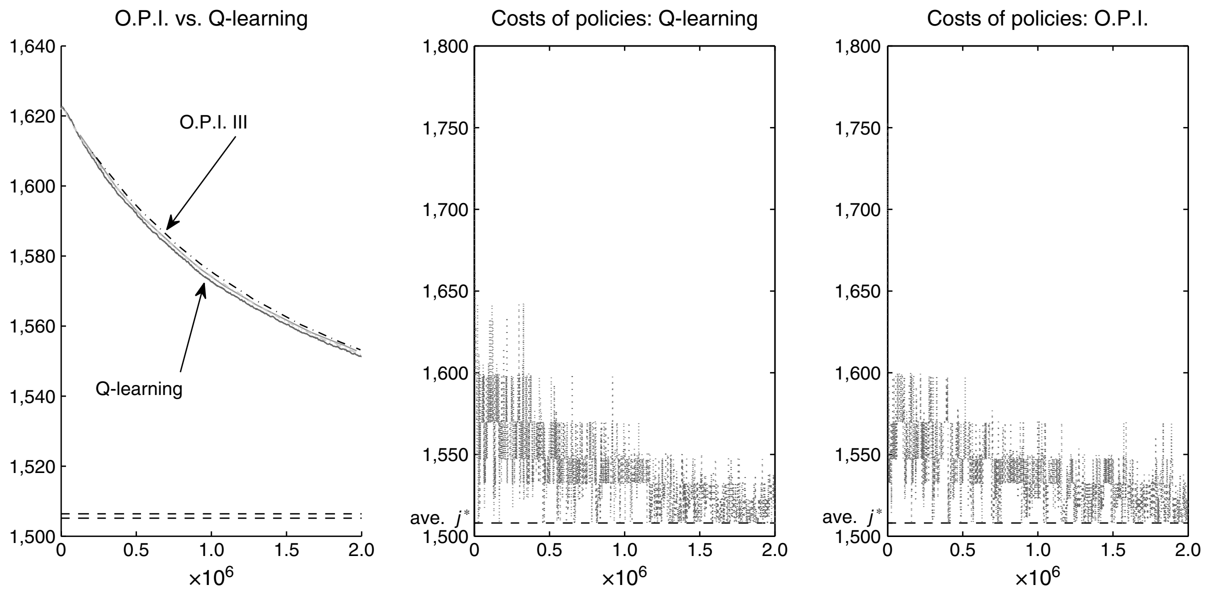

FIGURE 10. Comparison of the optimistic PI algorithm III and Q-learning at the early phase for the automobile replacement problem.

6. Error bounds for approximate implementations. In this section, we discuss the effect of approximations on the algorithm of §2, and we derive a bound on the suboptimality of certain policies in terms of approximation error. In particular, we consider performing the iteration $Q_{k+1} = F^{m_k}_{J_k,\nu_k} Q_k$ (see Equation (2.4)) approximately and generate a sequence $\{Q_k\}$ such that

$$\|Q_{k+1} - F^{m_k}_{J_k,\nu_k} Q_k\|_\infty \le \delta, \tag{6.1}$$

for some $\delta > 0$ and a sequence of positive integers $\{m_k\}$. We update J_k according to

$$J_{k+1}(i) = \min_{u\in U(i)} Q_{k+1}(i,u), \quad \forall\, i = 1,\ldots,n, \tag{6.2}$$

or more generally, we find J_{k+1} such that

$$\left| J_{k+1}(i) - \min_{u\in U(i)} Q_{k+1}(i,u)\right| \le \epsilon, \quad \forall\, i = 1,\ldots,n, \tag{6.3}$$

for some $\epsilon > 0$. As before, we let the randomized policy ν_{k+1} be chosen in any desirable way. We analyze the performance of the deterministic policies $\{\mu_{k+1}\}$, where $\mu_{k+1}(i)$ is assumed to attain the minimum within ϵ in Equation (6.2) for all states i, i.e.,

$$Q_{k+1}(i,\mu_{k+1}(i)) \le \min_{u\in U(i)} Q_{k+1}(i,u) + \epsilon, \quad \forall\, i = 1,\ldots,n. \tag{6.4}$$

These would be the deterministic policies of interest when the algorithm is terminated at iteration $k+1$.

Several sources can contribute to the error terms δ and ϵ, which are generally unknown. Collectively, Equations (6.1), (6.3), and (6.4) cover a variety of approximation scenarios, to which the error bound we derive in this section is applicable. Here are some examples:

(a) When $F^{m_k}_{J_k,\nu_k} Q_k$ is calculated using simulation in model-free learning by algorithms of §4 or calculated by the parallel asynchronous algorithms of §3, δ incorporates the error due to simulation noise and lack of synchronization. (The interval $[k, k+1]$ here corresponds to the time interval between two consecutive updates of the stopping cost vector J and the policy ν in those algorithms.)

(b) When the control space is large, imprecise minimization of $Q_{k+1}(i,u)$ in Equation (6.3) may be desirable, and the discrepancy can be absorbed into the error term ϵ.

(c) With compact representation, the sequences $\{Q_k\}$ and/or $\{J_k\}$ may be restricted to be within some parametrized families of functions, and δ and ϵ reflect the size of the approximation error.

(d) The policy sequence $\{\mu_k\}$ may be restricted in a certain subset of structured policies that have a compact representation in the policy space, whereas policies ν_k may be chosen based on the corresponding μ_k and also the space limits in the computation. Positivity of the scalar ϵ in Equation (6.4) then reflects the fact that, because of the compact representation of μ_{k+1}, we do not have $Q_{k+1}(i,\mu_{k+1}(i)) = J_{k+1}(i)$, even when Equation (6.2) holds and the algorithm implementation is synchronous.

In §7 we will discuss a specific simulation-based method of computing $F^{m_k}_{J_k,\nu_k} Q_k$ with compact representation.

Our bound on the performance of μ_k, given in the following proposition, is similar to what is generally viewed as the standard bound for the performance of approximate PI (Bertsekas and Tsitsiklis [11, Proposition 6.2]). Our analysis also holds when m_k may be equal to ∞, in which case Equation (6.1) is replaced by

$$\|Q_{k+1} - Q_{J_k,\nu_k}\|_\infty \le \delta.$$

Proposition 6.1. *Assume that for some $\delta, \epsilon \ge 0$ and each $k \ge 0$, Equation (6.1) holds for some positive integer m_k, and Equations (6.3) and (6.4) also hold. Then, for any stationary policy μ that is a limit point of $\{\mu_k\}$, we have*

$$\|J_\mu - J^*\|_\infty \le \frac{2(\delta+\epsilon)}{(1-\alpha)^2}.$$

We prove Proposition 6.1 through two lemmas.

Lemma 6.1. *Given (J,ν) and $\delta, \epsilon \ge 0$, let Q, $\hat{Q}$, and $m \ge 1$ be such that*

$$\|\hat{Q} - F^m_{J,\nu} Q\|_\infty \le \delta,$$

and let $\hat{J}$ satisfy

$$\left| \hat{J}(i) - \min_{u\in U(i)} \hat{Q}(i,u)\right| \le \epsilon, \quad \forall\, i = 1,\ldots,n.$$

Then,

$$\|\hat{J} - J^*\|_\infty - \epsilon \le \|\hat{Q} - Q^*\|_\infty \le \alpha \max\{\|Q - Q^*\|_\infty,\ \|J - J^*\|_\infty\} + \delta.$$

Proof. Using the triangle inequality, the fact $Q^* = F^m_{J^*,\nu} Q^*$, and Lemma 2.2, we have

$$\|\hat{Q} - Q^*\|_\infty - \|\hat{Q} - F^m_{J,\nu} Q\|_\infty \le \|F^m_{J,\nu} Q - Q^*\|_\infty \le \alpha \max\{\|Q - Q^*\|_\infty, \|J - J^*\|_\infty\},$$

which, together with the assumption $\|\hat{Q} - F^m_{J,\nu} Q\|_\infty \le \delta$, implies the right-hand side of the desired inequality. The left-hand side follows from the generic inequality (2.10) and the triangle inequality. □

For any policy μ, we denote by T_μ the mapping defined by

$$(T_\mu J)(i) = \sum_{j=1}^n p_{ij}(\mu(i))(g(i, \mu(i), j) + \alpha J(j)), \quad \forall\, i = 1, \dots, n.$$

Lemma 6.2. *Given Q, let μ be a policy such that $\mu(i)$ attains within ϵ the minimum of $Q(i, u)$ over $u \in U(i)$ for all states i. Then,*

$$\|J_\mu - J^*\|_\infty \le \frac{2(\|Q - Q^*\|_\infty + \epsilon)}{1 - \alpha}.$$

Proof. Under the assumption, there exists $\hat{Q}$ with $\|\hat{Q} - Q\|_\infty \le \epsilon$ such that $\mu(i)$ attains the minimum of $\hat{Q}(i, u)$ over $u \in U(i)$ for all states i. By the triangle inequality, $\|\hat{Q} - Q^*\|_\infty \le \|Q - Q^*\|_\infty + \epsilon$. Hence, to prove the proposition, it is sufficient to consider μ and Q such that μ attains the minimum of $Q(i, u)$ for all states i, and to prove the inequality

$$\|J_\mu - J^*\|_\infty \le \frac{2\|Q - Q^*\|_\infty}{1 - \alpha}.$$

Let $\beta = \|Q - Q^*\|_\infty$, and let μ^* be an optimal policy. We have, for all states i,

$$|Q(i, \mu(i)) - Q^*(i, \mu(i))| \le \beta, \qquad |Q(i, \mu^*(i)) - Q^*(i, \mu^*(i))| \le \beta.$$

Note that

$$Q^*(i, \mu(i)) = (T_\mu J^*)(i), \qquad Q^*(i, \mu^*(i)) = (T_{\mu^*} J^*)(i) = J^*(i),$$

and by the definition of μ,

$$Q(i, \mu(i)) \le Q(i, \mu^*(i)).$$

Combining these relations, we have, for all states i,

$$(T_\mu J^*)(i) - J^*(i) \le (T_\mu J^*)(i) - Q(i, \mu(i)) + Q(i, \mu^*(i)) - J^*(i) \le \beta + \beta = 2\beta.$$

Using the standard bound $\|J_\mu - J^*\|_\infty \le \|T_\mu J^* - J^*\|_\infty / (1 - \alpha)$, the desired inequality follows. □

Proof of Proposition 6.1. Let

$$\beta_k = \max\{\|Q_k - Q^*\|_\infty, \|J_k - J^*\|_\infty\}.$$

By applying Lemma 6.1 with $J = J_k$, $\hat{J} = J_{k+1}$, $Q = Q_k$, $\hat{Q} = Q_{k+1}$, we have

$$\|J_{k+1} - J^*\|_\infty - \epsilon \le \|Q_{k+1} - Q^*\|_\infty \le \alpha \max\{\|Q_k - Q^*\|_\infty, \|J_k - J^*\|_\infty\} + \delta = \alpha\beta_k + \delta. \tag{6.5}$$

From Equation (6.5) and Lemma 6.2,

$$\|J_{\mu_{k+1}} - J^*\|_\infty \le \frac{2(\alpha\beta_k + \delta + \epsilon)}{1 - \alpha}. \tag{6.6}$$

Also, by the definition of β_{k+1}, Equation (6.5) implies that $\beta_{k+1} \le \alpha\beta_k + \delta + \epsilon$, and by iteration

$$\beta_{k+1} \le \alpha^{k+1}\beta_0 + (\alpha^k + \alpha^{k-1} + \cdots + 1)(\delta + \epsilon),$$

so that $\limsup_{k\to\infty} \beta_k \le (\delta + \epsilon)/(1 - \alpha)$. Using this relation in Equation (6.6), and taking the limit along a subsequence of μ_k that converges to a stationary policy μ, we obtain

$$\|J_\mu - J^*\|_\infty \le \frac{2(\delta + \epsilon)}{(1 - \alpha)^2}. \quad □$$

7. Basis function approximation and exploration. In §§3–5, we discussed Q-learning algorithms with lookup table representation, and we showed how our asynchronous deterministic and stochastic Q-learning/modified PI algorithms have improved convergence properties, while maintaining the reduced overhead advantage of modified PI. We now consider large problems and simulation-based approximate PI algorithms, where it is common to represent Q-factors and state costs compactly through a relatively small set of basis functions (or features); see, e.g., Bertsekas and Tsitsiklis [11], Sutton and Barto [39]. We will discuss two subjects: (i) a specific implementation of our Q-learning algorithm with function approximation, which uses as a subroutine the Q-learning algorithm of Tsitsiklis and Van Roy [45] for approximately solving optimal stopping problems, and (ii) the issue of exploration and how our approach can address it more naturally compared to standard PI approaches.

7.1. Function approximation. It is straightforward to combine function approximation with a version of our algorithm, which at each iteration carries out the steps described by Equations (6.1)–(6.4), by invoking the algorithm of Tsitsiklis and Van Roy [45] to approximately solve an optimal stopping problem. Let us provide some details of the latter algorithm within our context.

As in §2, we view $Q_{J_k,\nu_k}(i,u)$ as the Q-factor of an optimal stopping problem that corresponds to the action of not stopping at pair (i,u). We approximate $Q_{J_k,\nu_k}(i,u)$ using a linear approximation architecture of the form

$$\tilde{Q}(i,u)=\phi(i,u)'r, \quad \forall\, u\in U(i), \quad i=1,\ldots,n. \tag{7.1}$$

Here, $\phi(i,u)'$ is a row vector of s features whose inner product $\tilde{Q}(i,u)$ with a column vector of weights $r\in\Re^s$ provides an approximation for the Q-factor of (i,u).[6] In the typical policy evaluation phase, we start with an initial weight vector $\bar{r}_0$ (for example, one obtained from the preceding policy evaluation phase). We generate a single simulated trajectory $\{(i_0,u_0),(i_1,u_1),\ldots\}$ corresponding to an unstopped system, i.e., using transition probabilities from (i_t,u_t) to (i_{t+1},u_{t+1}) given by

$$p_{i_t i_{t+1}}(u_t)\nu_k(u_{t+1}\mid i_{t+1}).$$

Following the transition $((i_t,u_t),(i_{t+1},u_{t+1}))$, we update $\bar{r}_t$ by

$$\bar{r}_{t+1}=\bar{r}_t-\gamma_t\phi(i_t,u_t)q_t, \tag{7.2}$$

where q_t is the temporal difference

$$q_t=\phi(i_t,u_t)'\bar{r}_t-g(i_t,u_t,i_{t+1})-\alpha\min\{J_k(i_{t+1}),\phi(i_{t+1},u_{t+1})'\bar{r}_t\}, \tag{7.3}$$

and γ_t is a positive stepsize that diminishes to 0 (e.g., $\gamma_t=O(1/t)$). While generating the sequence $\{\bar{r}_t\}$, J_k and ν_k are held fixed. Moreover, ν_k can be selected arbitrarily, and may be flexibly used to encode exploration, i.e., induce a trajectory $\{(i_0,u_0),(i_1,u_1),\ldots\}$ with balanced representation of all state-control pairs; see the discussion in the subsequent §7.2.

Upon convergence, the preceding algorithm yields a weight vector r_{k+1} (the limit of $\{\bar{r}_t\}$) and an approximation Q_{k+1} to Q_{J_k,ν_k} of the form $Q_{k+1}(i,u)=\phi(i,u)'r_{k+1}$. Combined with the update rule of Equation (6.2),

$$J_{k+1}(i)=\min_{u\in U(i)}Q_{k+1}(i,u), \quad \forall\, i=1,\ldots,n,$$

or the ϵ-approximation version of Equation (6.3),

$$\left|J_{k+1}(i)-\min_{u\in U(i)}Q_{k+1}(i,u)\right|\le\epsilon, \quad \forall\, i=1,\ldots,n,$$

it yields an approximate PI method. We may also terminate the iterations (7.2) well before convergence of $\{\bar{r}_t\}$, giving the algorithm an optimistic character.

We note that instead of the algorithm (7.2), one may use related alternative algorithms to approximate $Q_{J,\nu}$. One scaled version of (7.2) is proposed by Choi and Van Roy [24]:

$$\bar{r}_{t+1}=\bar{r}_t-\gamma_t D_t^{-1}\phi(i_t,u_t)q_t, \tag{7.4}$$

[6] We do not discuss the important issue of selection of $\phi(i,u)$, but we note the possibility of its optimal choice within some restricted class by using gradient and random search algorithms (see Menache et al. [31] and Yu and Bertsekas [54] for recent work on this subject).

where D_t is a positive definite scaling matrix. For our purposes, to keep overhead per iteration low, it is important that D_t is chosen to be diagonal, and Choi and Van Roy [24] suggests suitable simulation-based choices. Alternative iterative optimal stopping algorithms are given by Yu and Bertsekas [52, 53], which have faster convergence properties, but require more overhead per iteration because they require a sum of a number of past temporal differences in the right-hand side of Equation (7.4).

When the number of states is large, J_k and ν_k cannot be stored explicitly, so they must either be generated when needed, or like Q_{J_k,ν_k}, they must be represented compactly through an approximation architecture. Without going into details, let us discuss briefly some of these possibilities and their impact on the computational overhead of the algorithm. We may choose J_k from a linear or nonlinear parametric family of functions. We may also choose ν_k from a parametric family of policies. A particular case of interest is where ν_k is chosen based on the policy μ_k obtained by minimization (see Equation (6.2)):

$$\mu_k(i) \in \arg\min_{u\in U(i)} Q_k(i,u) = \arg\min_{u\in U(i)} \phi(i,u)'r_k, \quad \forall\, i = 1,\ldots,n,$$

giving the method a PI flavor. In particular, ν_k may be a convex combination of μ_k and a simple randomized policy, with the weight on μ_k reflecting special considerations (e.g., exploration as discussed in §7.2). Since μ_k can be viewed as a policy parametrized by r_k and the controls $\mu_k(i)$ can be generated online for states encountered in the simulation, the policy ν_k can also be generated online and need not be stored. With this scheme, ν_k is also represented compactly; however, a drawback is that the reduced overhead advantage of modified PI is lost, because of the minimization operation involved in generating μ_k. Alternatively, before iteration $k+1$ starts, we may choose ν_k among certain structured policies that have a compact representation in the policy space, while taking into account μ_k. In this case, the reduced overhead advantage of modified PI is maintained.

Finally, we note that the convergence properties of the function approximation algorithm we described above may be quite complicated, not only because Q_{k+1} is just an approximation to Q_{J_k,ν_k}, but also because when Q-factor approximations of the form (7.1) are used, policy oscillations may occur, a phenomenon described by Bertsekas and Tsitsiklis [11, Section 6.4] (see also Bertsekas [8, 9, Section 6.3]). However, the error bound of §6 holds for this algorithm, although the associated approximation parameters δ and ϵ are generally unknown. (For characterizations of the approximation error of the method (7.2)–(7.3), see Tsitsiklis and Van Roy [45] and Van Roy [47].)

7.2. Exploration. Approximate PI methods constitute one of the major methodologies for approximate DP (see, e.g., the books by Bertsekas and Tsitsiklis [11], Sutton and Barto [39], Gosavi [26], Cao [22], Chang et al. [23], Meyn [32], Powell [33], Buşoniu et al. [20]; the textbook by Bertsekas [6] and its online chapter (Bertsekas [8]) provide a recent treatment and up-to-date references). These methods build on the standard PI framework and incorporate within it function approximation and stochastic approximation techniques, which are necessary for large problems and model-free learning of optimal control. For the approximate policy evaluation step, popular choices are the so-called TD methods, such as TD(λ) (Sutton [38]), least squares policy evaluation (LSPE(λ)) methods (Bertsekas and Ioffe [10]), and least squares temporal difference (LSTD(λ)) methods (Bradtke and Barto [19], Boyan [18]). Aggregation methods (Jaakkola et al. [28, 29], Gordon [25], Tsitsiklis and Van Roy [44], Baras and Borkar [2], Bertsekas [5, 8]) have also been used for policy evaluation. The goal of these algorithms is to approximate the costs $J_\mu(i)$ or Q-factors $Q_\mu(i,u)$ of a deterministic policy μ with linear cost function approximations of the form $\phi(i)'r$ or $\phi(i,u)'r$, respectively (see Equation (7.1)).

A critical issue in model-free learning is exploration. It is relatively simple to estimate the costs of a given policy μ, using state trajectories of the Markov chain induced by μ. The policy improvement step, however, could easily break down due to cost function approximation error at states that are rarely visited or unreachable under μ and the initial conditions of the Markov chain. Thus, it is desirable to adequately explore the state-control space beyond the part easily accessible with the policy that we want to evaluate. The exploration issue is more acute in real-time learning: if the learning agent can only visit a small subset of states frequently when starting from the initial state and following μ, the agent can be totally unaware of the larger outside world or poorly estimate the policy costs there. Such pathologies can be avoided in simulation-based PI where an adequate number of transitions can be simulated for any states, but even there, exploration mechanisms affect algorithm efficiency and are an issue of concern.

For approximate PI methods, a common approach to deal with exploration is an off-policy strategy (using the terminology of Sutton and Barto [39]; see also Precup et al. [34]), whereby we occasionally generate transitions involving randomly selected controls rather than the ones dictated by μ, and we then adjust the policy evaluation

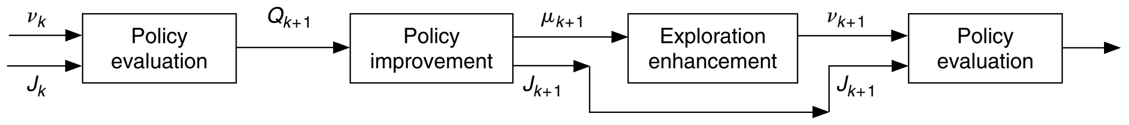

FIGURE 11. Illustration of exploration-enhanced PI algorithm.

Notes. The kth policy evaluation consists of approximate solution of the optimal stopping problem involving the randomized policy ν_k and the threshold/stopping cost J_k. It is followed by policy improvement that produces a new deterministic policy μ_{k+1}, which forms the basis for constructing the new randomized policy ν_{k+1} using some exploration mechanism.

procedures, especially those based on multistage costs, to take this into account. For example, some LSTD(λ)-type policy evaluation algorithms use the richer set of data thus obtained together with importance sampling to guarantee convergence (Bertsekas and Yu [13], Yu [50, 51]). But they can encounter problems due to high simulation noise variance, and they also require the solution of a linear projected equation of dimension equal to the number of features, so they may be unsuitable for problems where many features are required for good performance. Some other policy evaluation algorithms, such as LSPE(λ) and TD(λ), may encounter convergence difficulties because a contraction mapping property on which they rely, and which holds for trajectories generated by μ, now need not hold (see, e.g., Bertsekas [11, Example 6.7], which provides an instance of divergence of TD(0)). Aggregation methods do not rely on a contraction property, but they are restricted in the types of basis functions that they can use (see, e.g., the discussion in Bertsekas [8, 9]). Thus, it is an interesting question to derive TD(λ)-like exploration-enhanced approximate policy evaluation methods that have sound convergence properties and are suitable for architectures with a large number of features. As an example of recent efforts in this direction, stochastic gradient-based optimization has also been suggested in combination with the TD(0) approximation framework to tackle the convergence issue (Sutton et al. [40, 41], Maei et. al. [30]). Although future research may add some more remediation techniques to approximate PI with exploration enhancements, it is reasonable to question whether the standard PI framework itself is sufficiently flexible to handle exploration in all cases of concern.

Our methodology of this paper provides an alternative framework for the incorporation of exploration mechanisms in PI-like algorithms. By blending value and policy iteration (through the optimal stopping formulation of policy evaluation), and by separating cost approximation J_k from Q-factor approximation Q_k, our approach supports exploration naturally and removes at the technical level the conflict between exploration and policy evaluation that is present in approximate PI methods. As illustrated in Figure 11, to incorporate exploration, we may exploit the freedom to choose arbitrarily the distributions/policies ν_k that define the optimal stopping problems for the approximation algorithm described in §7.1. In particular, we may use ν_k that is a random mixture of a certain policy μ_k of interest and another policy that induces exploration and enables visits to state-control pairs that are impossible/unlikely to generate under μ_k. This is similar to the off-policy approach, except that at each policy evaluation phase, regardless of how much exploration is carried out, there is no convergence difficulty in solving the optimal stopping problem, despite the fact that the algorithm is TD(0)-like. Compared to the LSTD-based off-policy approach mentioned earlier, our method is better suited for an approximation architecture with a large number of features.

The structure of our algorithm can also be exploited in a somewhat different way to provide improved estimation of the costs of various states, thereby providing exploration enhancement of a kind that is different from the one described above. In particular, suppose that in some problems, upper bounds $\hat{J}(i)$ to the true optimal costs $J^*(i)$ are known or can be calculated for some states i, for example, by using some extra simulation with either the current policy, some exploration policy, or some other heuristic policy. Then the stopping costs $J_k(i)$ in our algorithms may be replaced by $\min\{J_k(i), \hat{J}(i)\}$ for those states i, with potentially substantial performance improvement. Thus, the flexibility of our approach can be valuable when problem-specific knowledge or extra simulation/exploration can be used to calculate suitable upper bounds to state costs.

8. Conclusions.

We have developed a new Q-learning algorithm for discounted MDP. In its lookup table form, the algorithm admits interesting deterministic and stochastic asynchronous implementations, akin to modified PI, with sound convergence properties and less overhead per iteration over the classical Q-learning algorithm. In its compact representation/approximate form, the algorithm addresses in a new way the critical issue of exploration in the context of simulation-based approximations.

The main idea of this paper is to replace the standard linear system of equations used for policy evaluation with an optimal stopping problem that may be solved inexactly, with a finite number of Q-learning/VI iterations. This idea finds application in other settings, beyond the discounted MDP framework of this paper, such as semi-Markov and minimax discounted problems, as well as some undiscounted problems of the stochastic shortest path type. Our paper Bertsekas and Yu [14] deals with related deterministic algorithms and addresses such problems within a more general and unifying formulation. Our paper Yu and Bertsekas [55] considers stochastic shortest path problems and develops results that parallel those in the present paper.

The optimal stopping framework for policy evaluation may also be combined with other exact and approximate policy iteration ideas, involving, for example, aggregation, interpolation, and distributed multiagent implementations. Some of these ideas have been explored in Bertsekas and Yu [14] within a related context.

Acknowledgments. The research of Dimitri Bertsekas was supported by the National Science Foundation [Grant ECCS-0801549] and by the Air Force [Grant FA9550-10-1-0412]. This work was done in part when Huizhen Yu was with the Department of Computer Science and the Helsinki Institute for Information Technology, University of Helsinki. Her research was supported in part by the Academy of Finland [Grant 118653 (ALGODAN)] and by the Air Force [Grant FA9550-10-1-0412].

References

[1] Abounadi, J., D. P. Bertsekas, V. Borkar. 2002. Stochastic approximation for nonexpansive maps: Application to Q-learning algorithms. *SIAM J. Control Optim.* **41** 1–22.
[2] Baras, J. S., V. S. Borkar. 2000. A learning algorithm for Markov decision processes with adaptive state aggregation. *Proc. IEEE CDC* **4** 3351–3356.
[3] Bertsekas, D. P. 1982. Distributed dynamic programming. *IEEE Trans. Automatic Control* **AC-27** 610–616.
[4] Bertsekas, D. P. 1983. Asynchronous distributed computation of fixed points. *Math. Programming* **27** 107–120.
[5] Bertsekas, D. P. 2005. *Dynamic Programming and Optimal Control*, 3rd ed, Vol. 1. Athena Scientific, Belmont, MA.
[6] Bertsekas, D. P. 2007. *Dynamic Programming and Optimal Control*, 3rd ed, Vol. 2. Athena Scientific, Belmont, MA.
[7] Bertsekas, D. P. 2010. Williams-Baird counterexample for Q-factor asynchronous policy iteration. http://web.mit.edu/dimitrib/www/Williams-Baird_Counterexample.pdf.
[8] Bertsekas, D. P. 2011. Approximate dynamic programming. http://web.mit.edu/dimitrib/www/dpchapter.html.
[9] Bertsekas, D. P. 2011. Approximate policy iteration: A survey and some new methods. *J. Control Theory Appl.* **9** 310–335.
[10] Bertsekas, D. P., S. Ioffe. 1996. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Laboratory for Information and Decision Systems, Report LIDS-P-2349, MIT, Cambridge, MA.
[11] Bertsekas, D. P., J. N. Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.
[12] Bertsekas, D. P., J. N. Tsitsiklis. 1997. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, Belmont, MA.
[13] Bertsekas, D. P., H. Yu. 2009. Projected equation methods for approximate solution of large linear systems. *J. Comput. Appl. Math.* **227** 27–50.
[14] Bertsekas, D. P., H. Yu. 2010. Asynchronous distributed policy iteration in dynamic programming. *Allerton Conf. on Communication, Control, and Computing, Allerton, IL*, 1368–1375.
[15] Bhatnagar, S., K. M. Babu. 2008. New algorithms of the Q-learning type. *Automatica* **44** 1111–1119.
[16] Borkar, V. S. 1998. Asynchronous stochastic approximations. *SIAM J. Control Optim.* **36** 840–851; correction note in ibid. **38** 662–663.
[17] Borkar, V. S. 2008. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge University Press, New York.
[18] Boyan, J. A. 2002. Technical update: Least-squares temporal difference learning. *Machine Learn.* **49** 1–15.
[19] Bradtke, S. J., A. G. Barto. 1996. Linear least-squares algorithms for temporal difference learning. *Machine Learn.* **22** 33–57.
[20] Buşoniu, L., R. Babuška, B. De Schutter, D. Ernst. 2010. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, New York.
[21] Canbolat, P. G., U. G. Rothblum. 2012. (Approximate) iterated successive approximations algorithm for sequential decision processes. *Annals Oper. Res.* Forthcoming.
[22] Cao, X. R. 2007. *Stochastic Learning and Optimization: A Sensitivity-Based Approach*. Springer, New York.
[23] Chang, H. S., M. C. Fu, J. Hu, S. I. Marcus. 2007. *Simulation-Based Algorithms for Markov Decision Processes*. Springer, New York.
[24] Choi, D. S., B. Van Roy. 2006. A generalized Kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynam. Systems: Theory Appl.* **16** 207–239.
[25] Gordon, G. J. 1995. Stable function approximation in dynamic programming. *Proc. 12th Internat. Conf. Machine Learning*, Morgan Kaufmann, San Francisco, 261–268.
[26] Gosavi, A. 2003. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Springer-Verlag, New York.
[27] Howard. 1960. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.
[28] Jaakkola, T., M. I. Jordan, S. P. Singh. 1994. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Comput.* **6** 1185–1201.
[29] Jaakkola, T., S. P. Singh, M. I. Jordan. 1995. Reinforcement learning algorithm for partially observable Markov decision problems. *Adv. Neural Inform. Processing Systems* **7** 345–352.
[30] Maei, H. R., C. Szepesvari, S. Bhatnagar, D. Silver, D. Precup, R. S. Sutton. 2009. Convergent temporal-difference learning with arbitrary smooth function approximation. *The 23rd Annual Conf. Neural Information Processing Systems, Vancouver, BC*, 1204–1212.
[31] Menache, I., S. Mannor, N. Shimkin. 2005. Basis function adaptation in temporal difference reinforcement learning. *Ann. Oper. Res.* **134** 215–238.
[32] Meyn, S. 2007. *Control Techniques for Complex Networks*. Cambridge University Press, New York.
[33] Powell, W. B. 2007. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, New York.
[34] Precup, D., R. S. Sutton, S. Dasgupta. 2001. Off-policy temporal-difference learning with function approximation. *Proc. 18th Internat. Conf. Machine Learn.,* Morgan Kaufmann, San Francisco, 417–424.

[35] Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York.

[36] Rosenthal, R. E., J. A. White, D. Young. 1978. Stochastic dynamic location analysis. *Management Sci.* **24** 645–653.

[37] Rothblum, U. G. 1979. Iterated successive approximation for sequential decision processes. J. W. B. van Overhagen, H. C. Tijms, eds. *Stochastic Control and Optimization*. Vrije Universiteit, Amsterdam.

[38] Sutton, R. S. 1998. Learning to predict by the methods of temporal differences. *Machine Learn.* **3** 9–44.

[39] Sutton, R. S., A. G. Barto. 1998. *Reinforcement Learning*. MIT Press, Cambridge, MA.

[40] Sutton, R. S., C. Szepesvari, H. R. Maei. 2008. A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. *The 22nd Annual Conf. Neural Inform. Processing Systems, Vancouver, BC*, 1609–1616.

[41] Sutton, R. S., H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvari, E. Wiewiora. 2009. Fast gradient-descent methods for temporal-difference learning with linear function approximation. *Proc. 26th Internat. Conf. Machine Learn.*, Omnipress, Madison, WI, 993–1000.

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

[43] Tsitsiklis, J. N. 2002. On the convergence of optimistic policy iteration. *J. Machine Learn. Res.* **3** 59–72.

[44] Tsitsiklis, J. N., B. Van Roy. 1996. Feature-based methods for large-scale dynamic programming. *Machine Learn.* **22** 59–94.

[45] Tsitsiklis, J. N., B. Van Roy. 1999. Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing financial derivatives. *IEEE Trans. Automatic Control* **44** 1840–1851.

[46] Tsitsiklis, J. N., D. P. Bertsekas, M. Athans. 1986. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. Aut. Control* **AC-31** 803–812.

[47] Van Roy, B. 2010. On regression-based stopping times. *Discrete Event Dynam. Systems: Theory Appl.* **20** 307–324.

[48] Watkins, C. J. C. H. 1989. Learning from delayed rewards. Ph.D. thesis, Cambridge University, Cambridge, UK.

[49] Williams, R. J., L. C. Baird. 1993. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. Report NU-CCS-93-11, College of Computer Science, Northeastern University, Boston.

[50] Yu, H. 2010. Convergence of least squares temporal difference methods under general conditions. *Proc. 27th Internat. Conf. Machine Learn.*, Omnipress, Madison, WI, 1207–1214.

[51] Yu, H. 2010. Least squares temporal difference methods: An analysis under general conditions. Technical Report C-2010-39, Department Computer Science, University of Helsinki, Helsinki, Finland.

[52] Yu, H., D. P. Bertsekas. 2007. A least squares Q-learning algorithm for optimal stopping problems. Report 2731, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

[53] Yu, H., D. P. Bertsekas. 2007. Q-learning algorithms for optimal stopping based on least squares. *Proc. European Control Conf.* (*ECC*), *Kos, Greece*, 2368–2375. 3

[54] Yu, H., D. P. Bertsekas. 2009. Basis function adaptation methods for cost approximation in MDP. *Proc. IEEE Symp. Approximate Dynamic Programming and Reinforcement Learn., Nashville, TN.*

[55] Yu, H., D. P. Bertsekas. 2011. Q-learning and policy iteration algorithms for stochastic shortest path problems. Report 2871, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.