AsyncQVI: Asynchronous-Parallel Q-Value Iteration for Reinforcement Learning with Near-Optimal Sample Complexity

Yibo Zeng\textsuperscript{a}, Fei Feng\textsuperscript{b}, Wotao Yin\textsuperscript{b}

\textsuperscript{a}School of Mathematical Sciences, Fudan University, Shanghai 200433, China
\textsuperscript{b}Department of Mathematics, University of California, Los Angeles, CA 90095, US

Abstract

In this paper, we propose AsyncQVI, an asynchronous-parallel Q-value iteration for Reinforcement Learning problems. Given such a problem with $|S|$ states, $|A|$ actions, and a discounted factor $\gamma \in (0,1)$, AsyncQVI uses memory of size $O(|S|)$ and returns an $\varepsilon$-optimal policy with probability at least $1-\delta$ using

$$\hat{O}\left(\frac{|S||A|}{(1-\gamma)^5\varepsilon^2 \log\left(\frac{1}{\delta}\right)}\right)$$

samples\textsuperscript{1}. AsyncQVI is also the first asynchronous-parallel algorithm for reinforcement learning with a convergence rate and a sample complexity. Its sample complexity nearly matches the theoretical lower bound. The relatively low memory footprint and parallel ability of AsyncQVI make it suitable for large-scale applications. In numerical tests, we compare AsyncQVI with four sample-based value iteration methods. The test results show AsyncQVI is highly efficient and achieves linear parallel speedup.

Keywords: Reinforcement Learning, Asynchronous-Parallel Algorithms, Q-value Iteration, Markov Decision Processes

1. Introduction

Reinforcement learning (RL) is a rapidly developing area of artificial intelligence \cite{1, 2, 3, 4}. With the advent of big-data RL applications, computational costs have increased significantly. Therefore, we resort to parallel computing techniques to accomplish RL tasks. Asynchronous (async) parallel iterative algorithms in solving RL, epitomized by \cite{5}, have recently gained increasing interests \cite{3, 4, 6}. Compared to synchronous (sync) parallel algorithms, where the agents must wait for the slowest agent to accomplish a task before they can all proceed to the next one, async-parallel algorithms allow agents to run continuously with little idling. Hence, async-parallel algorithms complete more tasks than their synchronous counterparts (though information delays and inconsistencies may negatively affect the task quality). Async-parallel algorithms have other advantages \cite{7}: the system is more tolerant of computing faults and communications glitches; it is also easy to incorporate new agents.

In this paper, we develop an async-parallel algorithm for the RL problems based on Discounted Infinite-Horizon Markov Decision Processes (DMDPs). DMDP is described by a tuple $(S, A, P, r, \gamma)$, where $S$ is a finite state space, $A$ is a finite action space, $P$ contains the transition probabilities, $r$ is the instant reward, and $\gamma \in (0,1)$ is a discounted factor. At each time step $t$, the controller

\textsuperscript{1}We use $\hat{O}$ to omit polylogarithmic factors, i.e., $\hat{O}(f) = O(f \cdot (\log f)^{O(1)})$. 

Preprint submitted to Elsevier
or the decision maker observes a state \( s_t \in \mathcal{S} \), and selects an action \( a_t \in \mathcal{A} \) according to a policy \( \pi \), where \( \pi \) maps a state to an action. The action leads the environment to transit to a next state \( s_{t+1} \) with probability \( p_{ij}^{a_t} \) and the controller to receive an instant reward \( r_{ij}^{a_t} \). Here, \( r_{ij}^{a_t} \) is a deterministic value given the transitional instance \((s_t, a_t, s_{t+1})\). If only \( s_t \) and \( a_t \) are specified, \( r_{ij}^{a_t} \) is a random variable and \( r_{ij}^{a_t} = r_{ij}^{a_t} \) with probability \( p_{ij}^{a_t} \). Given a policy \( \pi : \mathcal{S} \rightarrow \mathcal{A} \), we call \( \mathbf{v}^\pi \in \mathbb{R}^{|\mathcal{S}|} \) the state-value vector of \( \pi \):

\[
\mathbf{v}^\pi := [v^\pi_1, v^\pi_2, \ldots, v^\pi_{|\mathcal{S}|}]^\top, \quad v^\pi_i := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_{s_t,a_t} s_0 = i],
\]

where the sequence \{\( s_0, a_0, s_1, a_1, \ldots, s_t, a_t \ldots \)\} consists of the state-action transitions generated by the DMDP under \( \pi \), i.e., \( \pi_{s_t} = a_t \), and the expectation \( \mathbb{E}[\cdot] \) is taken over the random trajectory. The problem DMDP seeks for an optimal policy \( \pi^* \) in the following problem:

\[
\max_{\pi} \mathbf{v}^\pi, \quad \forall i \in \mathcal{S}.
\]

The objective of RL is to solve the above problem without knowing the underlying DMDP, in particular, without the transition probability \( P \) and the instant reward \( r \). Since we can no longer compute the value expectation in RL, we must rely on transitional samples. Depending on the application, we have access to samples either taken over trajectories or returned by a generative model. Specifically, given any state-action pair \((i, a)\), a generative model returns a next state \( j \) with probability \( p_{ij}^a \) and the corresponding instant reward \( r_{ij}^a \). One can repeatedly call it with the same input \((i, a)\). Our algorithm must access a generative model; as a benefit, the algorithm uses only \( O(|S|) \) memory and a nearly optimal sample complexity.

We use notation \( \mathbf{p}_i^a := [p_{i1}^a, p_{i2}^a, \ldots, p_{i|S|}^a] \) and \( \mathbf{r}_i^a := \sum_{j \in \mathcal{S}} p_{ij}^a r_{ij}^a \) and assume, without loss of generality, \( r_{ij}^a \in [0, 1], \forall i, j \in \mathcal{S}, a \in \mathcal{A} \). We let \( \mathbf{v}^* \) denote the optimal value vector associated with an optimal policy \( \pi^* \). To measure the quality of a policy, we introduce

**Definition 1.1 (\( \varepsilon \)-optimal policy).** We call a policy \( \pi \) \( \varepsilon \)-optimal if \( \|\mathbf{v}^* - \mathbf{v}^\pi\|_\infty \leq \varepsilon \).

This paper introduces the algorithm Asynchronous-Parallel Q-Value Iteration (AsyncQVI) for RL, the first RL algorithm that is async-parallel and has a sample complexity. We show it returns an \( \varepsilon \)-optimal policy \( \pi \) with probability at least \( 1 - \delta \) using

\[
\tilde{O}\left( \frac{|\mathcal{S}||\mathcal{A}|}{(1 - \gamma)\delta^2 \varepsilon^2} \log \left( \frac{1}{\delta} \right) \right)
\]
samples, provided that each coordinate is updated at least once within \( O(|\mathcal{S}||\mathcal{A}|) \) time and the async delay is bounded also by \( O(|\mathcal{S}||\mathcal{A}|) \). [8] established the lower bound on the sample complexity of any RL problem with the generative model:

\[
\Omega\left( \frac{|\mathcal{S}||\mathcal{A}|}{(1 - \gamma)\delta^2 \varepsilon^2} \log \left( \frac{1}{\delta} \right) \right),
\]

for finding an \( \varepsilon \)-optimal policy \( \pi \) with probability at least \( 1 - \delta \). Therefore, our result nearly matches the lower bound up to dependence on \( (1 - \gamma) \) and logarithmic factors. AsyncQVI requires only \( O(|\mathcal{S}|) \) memory, which is minimal possible (without using dimension reduction) to store \( \pi : \mathcal{S} \rightarrow \mathcal{A} \). With its near-optimal sample complexity, minimal memory requirement, and asynchronous-parallel implementation, AsyncQVI is a very competitive algorithm for practical RL. The AsyncQVI package can be accessed from Github at [https://github.com/uclaopt/AsyncQVI](https://github.com/uclaopt/AsyncQVI).

**Notation:** We write a scalar in italic type, a vector or a matrix in boldface, and their components with subscripts. For example, \( \mathbf{v} \) and \( v_i \) are a vector and its \( i \)th component, respectively.
2. Related Works

[9] proposed async-parallel Dynamic Programming (DP) methods to solve DMDP problems. They established and analyzed basic asynchronous models, which are characterized by coordinate-wise update and asynchronous delay. This seminal work inspires a mass of algorithms for DMDPs through async-parallel methods.

Later on, instead of assuming full knowledge of DMDPs, which is often unrealistic under practical circumstances, the literature focuses on RL algorithms and uses sampling models to have access to DMDPs. [5] proposes Asynchronous-Parallel Q-learning using trajectory samples and ensures convergence. Although the convergence rate and the sample complexity were obtained for several single-threaded RL algorithms in [8, 10, 11, 12, 13, 14], there have been no such results for async-parallel algorithms.

Recently, another sampling model, the generative model has been proposed by [15]. This model is a simulator which takes any state-action pair \((s, a)\) as input and returns a next state \(j\) with probability \(p_{ij}\) and the corresponding instant reward \(r_{ij}\). Although the generative model is a stronger assumption than trajectory sampling, it is natural and practical when some data have been collected. Several algorithms based on the generative model have been proposed [8, 11, 12, 13, 15, 16].

Our async-parallel algorithm is also based on the generative model and derives stronger results compared to previous async-parallel DP or RL algorithms. We prove linear convergence and obtain a sample complexity, provided that both the time interval between two consecutive updates to each coordinate and the asynchronous delay are bounded. We also reduce memory requirement of our algorithm from \(O(|S||A|)\) to \(O(|S|)\). In Table 1, we compare related async-parallel DP methods or RL algorithms for DMDPs. Some papers [10, 11, 12] use the word “asynchronous” for coordinate-wise updates methods. Since those methods are single-threaded, their authors do not analyze stale information or async delay.

Although our sample complexity is slightly greater than [8, 13], the smaller memory footprint \(O(S)\) and the async-parallel implementation of our AsyncQVI are important for solving large scale RL problems. See Table 2 for related RL algorithms with the generative model.

In another line of work [17, 18], async-parallel algorithms were developed for fixed-point problems that are nonexpansive in a Hilbert space. (In comparison, our algorithm is based on a contraction in the \(\ell_\infty\) norm.) In their setting, one must choose a step size that depends on the maximum async delay [17] or the statistics of async delays [18]. In contrast, our algorithm does not use a step size and can be implemented without the accurate knowledge of async delay. Hence, AsyncQVI is relatively easier to implement.

Table 1: Related Async-Parallel DP Methods or RL Algorithms for DMDPs, where Delay: Asynchronous Delay; Rate: Convergence Rate; S.C.: Sample Complexity; Memory: Memory Space; B: Bounded; U: Unbounded.

| Algorithms          | Methods | Delay | Rate | S.C.     | Memory          | References |
|---------------------|---------|-------|------|----------|-----------------|------------|
| Totally Async QVI   | DP      | U\(^2\) | –    | N/A      | \(O(|S||A|)\)  | [9]        |
| Partially Async QVI | DP      | B     | \(-\) | N/A      | \(O(|S||A|)\)  | [9]        |
| Async Q-learning    | RL      | U\(^2\) | –    | –        | \(O(|S||A|)\)  | [5]        |
| AsyncQVI            | RL      | B     | \(\sqrt{\ }\) | \(\sqrt{\ }\) | \(O(|S|)\)    | This Paper |

\(^2\)Under the assumption: for all \(i, j\), \(\lim_{t \to \infty} \tau_{ij}(t) = \infty\) holds with probability 1.

\(^3\) [9, Section 6.3.5] does propose another Generic Partially Async Algorithm with convergence rate analysis. Yet, all coordinates are updated at each iteration and, thus, coordinate-wise update does not hold.
Table 2: Related RL Algorithms with the Generative Model.

| Algorithms            | Async | Sample Complexity | Memory | References |
|-----------------------|-------|-------------------|--------|------------|
| Variance-Reduced VI   | ×     | $\tilde{O}\left(\frac{|S||A|}{(1-\gamma)^2\epsilon^2}\log\frac{1}{\delta}\right)$ | $\mathcal{O}(|S||A|)$ | [13]        |
| Variance-Reduced QVI  | ×     | $\tilde{O}\left(\frac{|S||A|}{(1-\gamma)^2\epsilon^2}\log\frac{1}{\delta}\right)$ | $\mathcal{O}(|S||A|)$ | [8]         |
| AsyncQVI              | √     | $\tilde{O}\left(\frac{|S||A|}{(1-\gamma)^2\epsilon^2}\log\frac{1}{\delta}\right)$ | $\mathcal{O}(|S|)$ | This Paper |

3. Preliminaries

In this section, we review several key results on Q-value iteration and async-parallel algorithms.

3.1. Q-value Iteration

Given a DMDP $(S, A, P, r, \gamma)$ and a policy $\pi$, we define the action-value vector $Q^\pi$ with entries

$$Q^\pi_{i,a} = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_{s_0,...,s_t} \big| s_0 = i, a_0 = a\right].$$

For an optimal policy $\pi^*$, we let $Q^*$ denote the corresponding optimal action-value vector. It was shown in [19] that $Q^*$ satisfies the Bellman equation:

$$Q^*_{i,a} = \mathbb{E}[r_{s_0,...,s_t} + \gamma \max_{a' \in A} Q^*_{s_{t+1},a'} | s_t = i, a_t = a].$$

From $Q^*$, we obtain

$$\pi^*(i) = \arg\max_a Q^*_{i,a}, \quad v^*_i = \max_a Q^*_{i,a}, \quad \forall i \in S.$$

Hence, to derive an optimal policy $\pi^*$, it suffices to compute the optimal action-value vector $Q^*$.

It is well known that there is a fixed-point structure underlying Eq. (1). Specifically, given a vector variable $Q \in \mathbb{R}^{|S||A|}$, define the Q-value operator $T : \mathbb{R}^{|S||A|} \to \mathbb{R}^{|S||A|}$ as

$$[TQ]_{i,a} = \sum_{j=1}^{|S|} p_{ij} r_{ij} + \gamma \sum_{j=1}^{|S|} p_{ij} \max_{a'} Q_{j,a'},$$

where $Q_{i,a}$ is the $((i-1) \times |A| + a)$th component of $Q$ with $i \in \{1, \ldots, |S|\}$ and $a \in \{1, \ldots, |A|\}$. Lemma 3.1 below states that $T$ is a $\gamma$-contraction. Combining with Eq. (1), one can apply a fixed-point iteration for recovering $Q^*$.

Lemma 3.1. [20, Proposition 6.2.4] For any two vectors $Q, Q' \in \mathbb{R}^{|S||A|}$, it holds that

$$\|TQ - TQ'\|_\infty \leq \gamma \|Q - Q'\|_\infty.$$  \hspace{1cm} (3)

Consequently, there exists a unique fixed-point $Q^*$.

By now, we have converted a DMDP to a fixed-point problem. In the next subsection, we introduce the coordinate update model based on which we shall develop AsyncQVI.


3.2. Asynchronous-Parallel Coordinate Updates

Given a contraction $G : \mathbb{R}^n \to \mathbb{R}^n$ in the sense of (3), the fixed-point iteration

$$x(t + 1) = G(x(t)), \quad t \geq 0$$

converges linearly. Rewriting $Gx$ as $(G_1x, \ldots, G_nx)$, we call

$$x_i(t + 1) = \begin{cases} G_i(x(t)), & t \in T^i; \\ x_i(t), & t \notin T^i, \end{cases}$$

the coordinate update of $Gx$, where $x_i(t)$ is the $i$th coordinate of $x$ at iteration $t$ and

$$T^i = \{t \geq 0 : \text{coordinate } i \text{ is updated at iteration } t\}$$

is the set of iterations at which $x_i$ is updated.

We use a set of computing agents to perform coordinate updates in an async-parallel fashion. Unlike the typical parallel implementation where all the agents must wait for the slowest one to finish an update, async-parallel algorithms allow each agent to use the (possibly stale) information it has and complete more iterations within the same period of time, which is preferable for cases where the computing capacity is highly heterogeneous or the workload is far from balanced. See more discussions in [21].

We summarize a shared-memory async-parallel coordinate-update algorithm in Algorithm 1, where each agent first chooses one coordinate to update, then reads required information from global memory to its local cache, and finally uploads its computed result to the shared memory.

**Algorithm 1: Asynchronous-Parallel Coordinate Updates**

1. **Shared variables:** $x^0, L > 0, t \leftarrow 0$
2. **Private variable:** $\hat{x}$
3. **while** $t < L$, **every agent asynchronously do**
4. select $i \in \{1, 2, \cdots, n\}$ according to some criterion;
5. read (required) shared variable to local memory $\hat{x} \leftarrow x$;
6. perform an update $x_i \leftarrow G_i(\hat{x})$;
7. increment the global counter $t \leftarrow t + 1$;

By Line 6 in Algorithm 1, the $t$th update can be written as

$$x_i(t + 1) = \begin{cases} G_i(\hat{x}(t)), & t \in T^i; \\ x_i(t), & t \notin T^i, \end{cases}$$

$$\hat{x}(t) := [x_1(\tau_1(t)), \ldots, x_n(\tau_n(t))]^\top$$

(4)

where $x_j(\tau_j(t))$ is the most recent version of $x_j$ available at time $t$ that is used to compute $x_i(t + 1)$. We have that $0 \leq \tau_j(t) \leq t$. The difference $t - \tau_j(t)$ is called the delay.

In this paper, we adopt partial asynchronism [9]:

**Assumption 3.2** (Partial Asynchronism$^4$). For the async-parallel algorithm, there exists two positive integers $B_1, B_2$ (asynchronism measure) such that:

---

$^4$Assumption 1.1 in [9, Section 7.1] uses $B$ for both $B_1$ and $B_2$. Because $B_1$ and $B_2$ are different in practice, we keep them separate to derive a tighter bound. Further, we have dropped assumption (c) there to make our algorithm easier to implement.
For every $i$ and for every $t \geq 0$, at least one of the elements of the set \( \{t, t+1, \ldots, t+B_1-1\} \) belongs to $T_i$.

(b) There holds $t - B_2 < \tau_j(t) \leq t$, for all $j$ and all $t \geq 0$.

Assumption 3.2 (a) ensures that the time interval between consecutive updates to each coordinate is uniformly bounded by $B_1$ and (b) ensures that the communication delays are uniformly bounded by $B_2$. Note that when $B_1 = B_2 = 1$, the algorithm becomes synchronous. The partial asynchronism assumption is often easy to enforce in a practical implementation. Convergence under this assumption was established in [22].

**Proposition 3.3.** [22, Theorem 2] Consider the iterations Eq. (4) under Assumption 3.2. Suppose that $G$ is $\gamma$-contractive under infinity norm and $x^*$ is the fixed-point of $G$. Then

$$\|x(t) - x^*\|_\infty \leq \|x(0) - x^*\|_\infty \rho^{t-2B_1}$$

for all $t \geq B_1$, where $\rho = \gamma^{\frac{1}{B_1 + B_2 - 1}}$.

In big-data RL applications, the transition probability $P$ is usually sparse, so in a state $i$, the possible next states form a tiny subset $S_i \subset S$. Hence, we just require certain components of $Q$ when conducting async-parallel Q-value iteration, and we only need to bound the asynchronous delay over the smaller subset $S_i$. Therefore, we usually have $B_2 \ll B_1$, where $B_1 \geq |S||A|$. Hence, the convergence rate $\gamma^{\frac{1}{B_1 + B_2 - 1}}$ we obtain is significantly better than $\gamma^{\frac{1}{2(B_1 + B_2) - 1}}$ from [22, Theorem 2]; the proof is deferred to Appendix A.

**Remark 3.4 (Total Asynchronism).** Here we do not adopt the total asynchronism notion [9, Section 6.1, Assumption 1.1]. On one hand, it cannot provide convergence rate. Specifically, it allows arbitrarily long delays between revisits (though revisits are infinitely many), so no improvement can be said for finite iterations. On the other hand, partial async can avoid this case and be practically enforced [9, Section 7.1]. Hence, in this paper, we adopt partial asynchronism.

### 4. AsyncQVI: Asynchronous-Parallel Q-value Iteration

In this section, we present AsyncQVI and its convergence analysis.

AsyncQVI (Algorithm 2) is an asynchronous randomized version of Eq. (2). To develop AsyncQVI, we first apply the asynchronous framework (Algorithm 1) to Eq. (2), obtaining

$$Q_{i,a}(t+1) = \begin{cases} 
\sum_j p_{ij}^a r_{ij}^a + \gamma \sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}(t), & t \in T_{i,a}; \\
Q_{i,a}(t), & t \notin T_{i,a}.
\end{cases}$$

Since there is no knowledge of the transition probability, we approximate the expectations $\sum_j p_{ij}^a$ by random sampling (routine $APX$ in Algorithm 3). This is done by accessing a generative model $\mathcal{GM}$, which takes a state-action pair $(i,a)$ as input and returns a next state $j$ with probability $p_{ij}^a$ and the corresponding instant reward $r_{ij}^a$. So instead of (6), we have

$$Q_{i,a}(t+1) = \begin{cases} 
r + \gamma S(\mathcal{M}(t)) & t \in T_{i,a}; \\
Q_{i,a}(t), & t \notin T_{i,a}.
\end{cases}$$

\[5\] If we insist on assuming total async, a weaker rate result (compared with partial async) is possible with further assumption, e.g., [22, Theorem 3] is established under [22, Assumption 3]. However, it essentially makes total async like partial async. Further, in this case, our algorithm and analysis still work.
where \( r := \frac{1}{K} \sum_k r_{i,j}^a \) and \( S(\hat{Q}(t)) := \frac{1}{K} \sum_k \max_{a'} \hat{Q}_{j,a'}(t) \) are the empirical means of \( \sum_j p_{ij}^a r_{i,j}^a \) and \( \sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}(t) \), respectively. For the purpose of analysis, we tune the update slightly by substituting a small constant \( (1 - \gamma)\varepsilon/4 \) to Eq. (7). Consequently, AsyncQVI is mathematically modeled by the iteration

\[
Q_{i,a}(t + 1) = \begin{cases} 
  r + \gamma S(\hat{Q}(t)) - (1 - \gamma)\varepsilon/4 & t \in \mathcal{I}_{i,a}; \\
  Q_{i,a}(t), & t \notin \mathcal{I}_{i,a}.
\end{cases}
\]  

(8)

For memory efficiency, we do not form \( Q \in \mathbb{R}^{||S|| \cdot ||A||} \). Instead, since only the values \( \max_{a'} Q_{i,a'}(t) \) are used for update, we maintain two vectors \( v, \pi \in \mathbb{R}^{||S||} \); at each iteration \( t \), we ensure \( v_i(t) = \max_a Q_{i,a}(t), \pi_i(t) = \arg\max_a Q_{i,a}(t) \) and \( \hat{v}_j(t) = \max_{a'} \hat{Q}_{j,a'}(t) \). By this means, we reduce the memory complexity from \( \mathcal{O}(||S|| \cdot ||A||) \) to \( \mathcal{O}(||S||) \), which is of a great advantage in real applications.

**Algorithm 2:** AsyncQVI: Asynchronous-Parallel Q-value Iteration

- **Input:** \( \varepsilon \in (0, (1 - \gamma)^{-1}), \delta \in (0, 1), L, K; \)
- **Shared variables:** \( v \leftarrow 0, \pi \leftarrow 0, t \leftarrow 0; \)
- **Private variables:** \( \hat{v}, r, S, q \);
- **while** \( t < L, \) **every agent asynchronously do**
  - select state \( i_t \in S \) and action \( a_t \in A; \)
  - copy shared variable to local memory \( \hat{v} \leftarrow v; \)
  - \( (r, S) \leftarrow \text{APX}(i_t, a_t, \hat{v}, K); \)
  - \( q \leftarrow r + \gamma S - \frac{(1 - \gamma)\varepsilon}{4}; \)
  - if \( q > v_{i_t} \) then
    - mutex lock;
    - \( v_{i_t} \leftarrow q, \pi_{i_t} \leftarrow a_t; \)
    - mutex unlock;
  - increment the global counter \( t \leftarrow t + 1; \)
- **return** \( \pi \)

**Algorithm 3:** APX \( (i, a, v, K) \)

- **Input:** the generative model \( \text{GM} \) for DMDP \( (S, A, P, r, \gamma); \)
- **Input:** state \( i \), action \( a \), vector \( v \), and sample size \( K; \)
- **initialize** \( r \leftarrow 0, \) and \( S \leftarrow 0; \)
- **for** \( k = 1 : K \) **do**
  - sample \( (j, R) \leftarrow \text{GM}(i, a); \) \( \triangleright \) call \( \text{GM} \) for the next state \( j \) and reward \( r_{i,j}^a \)
  - increment \( r \leftarrow r + R \) and \( S \leftarrow S + v_j; \)
- **set sample averages** \( r \leftarrow \frac{r}{K}, \) and \( S \leftarrow \frac{S}{K}; \)
- **return** \( (r, S) \)

We make a few remarks.

**Remark 4.1 (Coordinate Selection).** To guarantee convergence, the coordinate should be selected to satisfy Assumption 3.2. In practice, however, if all agents have similar powers, one can simply apply either uniformly random or globally cyclic selections.

---

7
Remark 4.2 (Memory Lock). To ensure that $v_i$ and $\pi_i$ are indeed the maximum value and a maximizer of the vector $\hat{Q}_i$, respectively, one might acquire a write lock (e.g. mutex) during the updates of $v$ and $\pi$ (Line 10, Algorithm 2). Such an implementation ensures correctness but has very low overhead since the writing collision is rare when there are much more states than the agents.

4.1. Convergence Analysis

Next, we establish convergence for AsyncQVI. To distinguish different sequences, we let $\{Q^E(t)\}$ denote the asynchronous coordinate update sequence generated through Eq. (6), where the superscript represents the updates with real expectations. Specifically, if AsyncQVI produces a sequence

$$Q_{i,a}(t+1) = \begin{cases} r + \gamma S(\hat{Q}(t)) - (1-\gamma)\varepsilon/4 & t \in \mathcal{I}_{i,a}; \\ Q_{i,a}(t), & t \notin \mathcal{I}_{i,a}, \end{cases}$$

where $\hat{Q}(t) := [Q_{1,1}(\tau_{1,1}(t)), \ldots, Q_{|S||A|}(\tau_{|S||A|}(t))]^\top$. Then

$$Q^E_{i,a}(t+1) = \begin{cases} \bar{r}^a_i + \gamma \sum_j p^a_{ij} \max_{a'} \hat{Q}^E_{j,a'}(t), & t \in \mathcal{I}_{i,a}; \\ Q^E_{i,a}(t), & t \notin \mathcal{I}_{i,a}, \end{cases}$$

where $\hat{Q}^E(t) := [Q^E_{1,1}(\tau_{1,1}(t)), \ldots, Q^E_{|S||A|}(\tau_{|S||A|}(t))]^\top$. There are two things to notice:

(i) $\{Q^E(t)\}_{t=0}^L$ and $\{Q(t)\}_{t=0}^L$ have the same initial point;
(ii) at any iteration, $\{Q^E(t)\}_{t=0}^L$ shares exactly the same choice of coordinate $(i_t,a_t)$ and the same asynchronous delay with $\{Q(t)\}_{t=0}^L$.

These properties are important to our analysis. Recall that we assume partial asynchronism (Assumption 3.2) for AsyncQVI. Then Eq. (10) also meets Assumption 3.2. Hence, Eq. (10) converges following Lemma 3.1 and Proposition 3.3. Since Eq. (9) is an approximation of Eq. (10), we can leverage the convergence of Eq. (10) to establish the convergence of AsyncQVI. To this end, we first review Hoeffding’s Inequality to analyze the sampling error.

Lemma 4.3 (Hoeffding’s Inequality [23]). Let $X_1, \ldots, X_m$ be i.i.d real valued random variables with $X_j \in [a_j, b_j]$ and $Y = \frac{1}{m} \sum_{j=1}^m X_j$. For all $\varepsilon \geq 0$,

$$\mathbb{P} \left[ \left| Y - \mathbb{E}[Y] \right| \geq \varepsilon \right] \leq 2e^{-2m^2\varepsilon^2 / \sum_{j=1}^m (b_j - a_j)^2}.$$

By Hoeffding’s Inequality, the error between the sample averages produced by $\text{APX}$ and the true expectations can be controlled with enough number of samples. Specifically, we have:

Lemma 4.4. Given $i \in \{1, \ldots, |S|\}$, $a \in \{1, \ldots, |A|\}$, a vector $\hat{v} \in \mathbb{R}^{|S|}$ where $0 \leq \hat{v}_i \leq \frac{1}{\gamma}$, and a constant $L$, with $K = \left\lceil \frac{8}{(1-\gamma)^2} \log \left( \frac{4L}{\delta} \right) \right\rceil$ samples, $\text{APX}$ returns $r$ and $S$ satisfying

$$|r - \bar{r}^a_i| \leq \frac{(1-\gamma)^2 \varepsilon}{4}, \quad |S - \Phi^a_i \top \hat{v}| \leq \frac{(1-\gamma)\varepsilon}{4}$$

with probability at least $1 - \frac{\delta}{L}$.
**Proof.** As we explained before, both $r$ and $S$ are averages of $K$ i.i.d. samples with $E[r] = \sum_j p^a_{ij} r^a_{ij} := \bar{r}^a_i$ and $E[S] = \sum_j p^a_{ij} \hat{v}_j := p^a_i \hat{v}$. Since we assume $r^a_{ij} \in [0, 1]$, letting $K = \lceil \frac{8}{(1-\gamma)^2 \varepsilon^2 \log \left( \frac{4L}{\delta} \right)} \rceil$, we can obtain that
\[
\mathbb{P}\left[ |r - r^a_i| \geq \frac{(1-\gamma)^2 \varepsilon}{4} \right] \leq 2e^{-\frac{2K^2(1-\gamma)^2 \varepsilon^2}{4K}} \leq \frac{\delta}{2L};
\]
\[
\mathbb{P}\left[ |S - p^a_i \hat{v}| \geq \frac{(1-\gamma)\varepsilon}{4} \right] \leq 2e^{-\frac{2K^2(1-\gamma)^2 \varepsilon^2}{4K}} \leq \frac{\delta}{2L}.
\]
\[\square\]

In Lemma 4.4, the bound over the input vector $\hat{v}$ is valid for AsyncQVI since each instant reward $r^a_{ij}$ lies in $[0, 1]$. As this can be easily proved by induction, we skip the details.

**Corollary 4.5** (Sample Concentration). With $K = \lceil \frac{8}{(1-\gamma)^2 \varepsilon^2 \log \left( \frac{4L}{\delta} \right)} \rceil$, AsyncQVI generates a sequence $\{r(t), S(t)\}_{t=0}^{L-1}$ that satisfies
\[
|r(t) + \gamma S(t) - \bar{r}^a_{it} - \gamma p^a_{it} \hat{v}(t)| \leq \frac{(1-\gamma)\varepsilon}{4}, \forall 0 \leq t \leq L-1
\]
with probability at least $1 - \delta$.

**Proof.** For a fixed iteration $t$, by Lemma 4.4,
\[
|r(t) + \gamma S(t) - \bar{r}^a_{it} - \gamma p^a_{it} \hat{v}(t)| \leq |r(t) - \bar{r}^a_{it}| + |\gamma S(t) - p^a_{it} \hat{v}(t)| \leq \frac{(1-\gamma)\varepsilon}{4}
\]
holds with probability at least $1 - \frac{\delta}{T}$. Taking a union bound over all $0 \leq t \leq L-1$ iterations gives the desired result. \[\square\]

Corollary 4.5 indeed provides a control over a one-step approximation error between Eq. (9) and Eq. (10) given that $Q = Q^E$. However, for the two sequences $\{Q(t)\}$ and $\{Q^E(t)\}$ that only share the same initial point, the error can accumulate. To tackle this issue, we further utilize the $\gamma$-contraction property to weaken previously cumulative error. More specifically, if the newly made error and the previously accumulated error keep the ratio $(1 - \gamma) : 1$ for each iteration, the overall error remains $(1 - \gamma)\varepsilon + \gamma \varepsilon = \varepsilon$. By this means, we control the difference between $\{Q(t)\}$ and $\{Q^E(t)\}$ as shown in Proposition 4.6.

**Proposition 4.6.** Given the total iteration number $L$, accuracy parameters $\varepsilon$ and $\delta$, with $K = \lceil \frac{8}{(1-\gamma)^2 \varepsilon^2 \log \left( \frac{4L}{\delta} \right)} \rceil$, AsyncQVI can generate a sequence $\{Q(t)\}_{t=1}^{L}$ satisfying
\[
\|Q(t) - Q^E(t)\|_{\infty} \leq \frac{\varepsilon}{2}, \forall 1 \leq t \leq L
\]
with probability at least $1 - \delta$.

**Proof.** We denote by $\mathcal{E}_1$ the event
\[
\left\{ |r(t) + \gamma S(t) - \bar{r}^a_{it} - \gamma p^a_{it} \hat{v}(t)| \leq \frac{(1-\gamma)\varepsilon}{4}, \forall 0 \leq t \leq L-1 \right\}.
\]
By Corollary 4.5, $\mathcal{E}_1$ occurs with probability at least $1 - \delta$. Next, we condition on $\mathcal{E}_1$ and prove Eq. (11) by induction. The basic case is trivial. For the induction step, we analyze the scenario at
$t + 1$ as two cases. When $t \notin \mathcal{T}^{i,a}$, $|Q_{i,a}(t + 1) - Q_{i,a}^E(t + 1)| \leq \varepsilon / 2$ follows from the hypothesis, since Eqs. (9) and (10) gives that

$$Q_{i,a}(t + 1) - Q_{i,a}^E(t + 1) = Q_{i,a}(t) - Q_{i,a}^E(t).$$

When $t \in \mathcal{T}^{i,a}$, by Eq. (9), Eq. (10) and triangle inequality, we have that

$$|Q_{i,a}(t + 1) - Q_{i,a}^E(t + 1)| = |r(t) + \gamma S(t) - r_i^a - \sum_j p_{ij}^a \max_{a'} Q_{j,a'}^E(t)| \leq |r(t) + \gamma S(t) - r_i^a - \sum_j p_{ij}^a \hat{Q}_{j,a'}^E(t)| \leq |r(t) + \gamma S(t) - r_i^a - \sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}^E(t)| \leq |r(t) + \gamma S(t) - r_i^a - \sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}^E(t)|.$$

By definition of $E_1$ and the induction hypothesis, we further obtain that

$$|Q_{i,a}(t + 1) - Q_{i,a}^E(t + 1)| \leq \frac{(1 - \gamma)\varepsilon}{4} + \frac{(1 - \gamma)\varepsilon}{4} + \gamma \frac{\varepsilon}{2} = \frac{\varepsilon}{2},$$

which completes the proof. □

**Theorem 4.7** (Linear Convergence). Under Assumption 3.2, given accuracy parameters $\varepsilon$ and $\delta$, with $L = \left[2B_1 + \frac{B_1 + B_2 - 1}{1 - \gamma} \log \left(\frac{2}{(1 - \gamma)^\gamma}\right)\right]$ and $K = \left[\frac{8}{(1 - \gamma)^\gamma} \log \left(\frac{4L}{\delta}\right)\right]$, AsyncQVI can produce $Q(L) \in \mathbb{R}^{|S||A|}$ and $v(L) \in \mathbb{R}^{|S|}$ satisfying

$$\|Q^* - Q(L)\|_\infty \leq \varepsilon, \quad \|v^* - v(L)\|_\infty \leq \varepsilon$$

with probability at least $1 - \delta$.

**Proof.** By Proposition 3.3,

$$\|Q^* - Q^E(L)\|_\infty \leq (1 - \gamma)^{-1} p^{L-2B_1} = (1 - \gamma)^{-1} \gamma \frac{L-2B_1}{B_1+B_2-1}.$$

Notice that $\gamma = (1 - (1 - \gamma)) \leq e^{-(1-\gamma)}$. We have that

$$\|Q^* - Q^E(L)\|_\infty \leq (1 - \gamma)^{-1} e^{-(1-\gamma)} \gamma \frac{L-2B_1}{B_1+B_2-1} \leq \frac{\varepsilon}{2},$$

where the last inequality holds with $L = \left[2B_1 + \frac{B_1 + B_2 - 1}{1 - \gamma} \log \left(\frac{2}{(1 - \gamma)^\gamma}\right)\right]$. Then, by Proposition 4.6, with probability at least $1 - \delta$,

$$\|Q^E(L) - Q(L)\|_\infty \leq \frac{\varepsilon}{2}. \quad (13)$$

Inserting Eq. (13) back into Eq. (12) gives the desired result

$$\|Q^* - Q(L)\|_\infty \leq \|Q^* - Q^E(L)\|_\infty + \|Q^E(L) - Q(L)\|_\infty \leq \varepsilon.$$

Then one can check $\|v^* - v(L)\|_\infty \leq \varepsilon$ at ease. □
4.2. \( \varepsilon \)-optimal Policy

In the following theorem, we show that the vector \( \pi \) maintained through the iterations is an \( \varepsilon \)-optimal policy (see Definition 1.1); the proof is deferred to Appendix B. Using this theorem, we shall present the sample complexity of AsyncQVI in Corollary 4.9.

**Theorem 4.8.** Under Assumption 3.2, given accuracy parameters \( \varepsilon \) and \( \delta \), with

\[
L = \left\lceil \frac{2B_1 + B_1 + B_2 - 1}{1 - \gamma} \log \left( \frac{2}{(1 - \gamma) \varepsilon^2} \right) \right\rceil \quad \text{and} \quad K = \left\lceil \frac{8}{(1 - \gamma)^4 \varepsilon^2} \log \left( \frac{1}{\delta} \right) \right\rceil,
\]

AsyncQVI returns an \( \varepsilon \)-optimal policy \( \pi \) with probability at least \( 1 - \delta \).

**Corollary 4.9.** Under Assumption 3.2, AsyncQVI returns an \( \varepsilon \)-optimal policy \( \pi \) with probability at least \( 1 - \delta \) at the sample complexity

\[
\mathcal{O} \left( \frac{|S||A|}{(1 - \gamma)^5 \varepsilon^2 \log \left( \frac{1}{\delta} \right)} \right),
\]

provided that \( B_1 + B_2 = \mathcal{O}(|S||A|) \).

Moreover, given the complete knowledge of transition \( P \) and reward \( r \), one can build a generative model in \( \mathcal{O}(|S|^2|A|) \) prepossessing time [24], and the GM produces a sample in \( \mathcal{O}(1) \) arithmetic operations. In this sense, AsyncQVI also has the following computational complexity results.

**Corollary 4.10 (Near-Optimal Sample Complexity).** Under Assumption 3.2, AsyncQVI returns an \( \varepsilon \)-optimal policy \( \pi \) with probability at least \( 1 - \delta \) at the sample complexity

\[
\mathcal{O} \left( \frac{|S||A|}{(1 - \gamma)^5 \varepsilon^2 \log \left( \frac{1}{\delta} \right)} \right),
\]

provided that \( B_1 + B_2 = \mathcal{O}(|S||A|) \).

5. Numerical Experiments

5.1. Sailing Problem

To investigate the performance of AsyncQVI, we solve the sailing problem from [25] on a 100 \( \times \) 100 grid with 80000 states and 8 actions. Each state contains the sailor’s current position \((x, y)\) and the wind direction. Each action is one of the eight directions \{((0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1))\}. The goal is to reach the target position \((50, 50)\) at the lowest cost. Different from the original settings, we add more randomness to the system. Under the action \((\delta_x, \delta_y)\), the sailor will be further affected by two drift noises: a mild wind noise \( \mathcal{N}(0, \sigma_1^2) \) which occurs with probability 1 and a big vortex noise \( \mathcal{N}(0, \sigma_2^2) \) which occurs with a fairly small probability \( p \). So, the next position is

\[
(x + \delta_x + \mathcal{N}(0, \sigma_1^2), y + \delta_y + \mathcal{N}(0, \sigma_1^2)) \sim 1 - p, \quad \text{or} \quad (x + \delta_x + \mathcal{N}(0, \sigma_1^2 + \sigma_2^2), y + \delta_y + \mathcal{N}(0, \sigma_1^2 + \sigma_2^2)) \sim p.
\]
The wind direction at next time maintains its current direction with probability 0.3, changes 45
degrees to either direction with probability 0.2 each direction, changes 90 degrees to either direction
with probability 0.1 each, changes 135 degrees to either direction with probability 0.04 each, and
reverses direction with probability 0.02. We set the instant reward as

\[ d \times \left| \frac{\text{angle between wind and action directions}}{45} \right|, \]

where \( d \) is a constant hyperparameter. When the reward is lower, we can take it as a higher cost.
If the sailor reaches the target position, the reward is 1.

5.2. Implementation

We compare five algorithms with a sample oracle (SO): AsyncQVI, Asynchronous-Parallel Q-
learning with adaptive stepsize (AsyncQL-ADA)[5], Asynchronous-Parallel Q-learning with con-
tant stepsize (AsyncQL-CONST), Variance-reduced value iteration (VRVI)[13], and Variance-
reduced Q-value iteration (VRQVI)[8]. All algorithms and the SO are implemented in C++11.
We use the thread class and pthread.h for parallel computing.

The tests were performed with 20 threads running on two 2.5GHz 10-core Intel Xeon E5-2670v2
processors. We chose the optimal sample method (uniformly random, cyclic, Markovian sampling)
and optimal hyperparameters (sample number, iteration number, learning rate, exploration rate)
for each algorithm individually. The learning rate of AsyncQL-ADA was set as \( 1/t^{0.51} \) according
to its theoretical analysis, where \( t \) is the iteration number. Our code is available at

https://github.com/uclaopt/AsyncQVI.

5.3. Policy Evaluation

Given a policy, we let the agent start from a random initial state and take actions following
the policy for 200 steps. Then, we evaluate the policy by recording the total discounted rewards
(\( \gamma = 0.99 \)) and whether the agent reaches the target position (flag = 1 if so). We repeat 100
episodes of this process and calculate the average total discounted rewards and total flags. A policy
with higher rewards and flags is better.

We test with different randomness and rewards which represent various MDP settings. In the
first test, one-step transition rewards are dominated by rewards for reaching the target (\( d = 0.05 \)
is very small compared with 1) and only the wind noise is considered in positioning. The agent
mainly aims at finding the target, which is relatively easy with minor noises. This leads to a fast
convergence of policies with low sampling request and bold learning rate (Figure 1). In the second
test, with increasing transition rewards (\( d = 0.15 \)), the agent needs to take a more economical way
to reach the goal. This prolongs the learning process with more samples and more prudent learning
rate (Figure 2). The next two tests make the situation more complicated with a big vortex noise,
which gives rise to higher sampling numbers and more conservative learning rates (Figure 3 and
4). This phenomenon occurs in VRVI and VRQVI as well. We skip the detailed parameters here.

In these four tests, AsyncQVI and AsyncQL-CONST are almost equivalently outstanding in
terms of time and show an at least 10\times speedup compared with VRQVI and VRVI with 20 threads
running parallel. On the other hand, VRQVI and VRVI have lower sample complexities, especially
on complicated cases. The testing results verify our theory. In the sequel, we further analyze
the performance of AsyncQVI and AsyncQL-CONST and provide heuristics on how to set sample
number and learning rate.
Figure 1: $\sigma_1 = 0.1$, $p = 0$, $d = 0.05$. AsyncQVI: $K = 1$, AsyncQL-CONST: $\alpha = 1$.

Figure 2: $\sigma_1 = 0.1$, $p = 0$, $d = 0.15$. AsyncQVI: $K = 35$, AsyncQL-CONST: $\alpha = 0.125$.

Figure 3: $\sigma_1 = 0.1$, $p = 0.05$, $\sigma_2 = 1$, $d = 0.05$. AsyncQVI: $K = 10$, AsyncQL-CONST: $\alpha = 0.15$. 
5.4. Performance Analysis and Heuristics

Recall that AsyncQVI derives from the Q-value operator $T$ (see Eq. (2)). Let $T_\alpha := (1-\alpha)I+\alpha T$, where $\alpha$ is the learning rate. One can get AsyncQL-CONST through the same approach. What’s special is, AsyncQL-CONST only takes one sample each time. This seems to be a very inaccurate approximation and might cause devastating error. However, note that when applying $T_\alpha$, sample range in Lemma 4.3 scales down to $[\alpha a_j, \alpha b_j]$. For fixed $\delta$ and $\epsilon$, the requested sample number $m$ decreases quadratically with respect to $\alpha$, since $m \geq C\alpha^2 \log \left(\frac{1}{\delta}\right)$. Hence, when $\alpha$ is smaller, AsyncQL-CONST converges more stably. On the other hand, a tiny learning rate also leads to slow progress, since $T_\alpha$’s contractive factor $(1 - \alpha + \alpha \gamma)$ approaches 1. Similarly, for AsyncQVI, when the sample number $K$ is larger, it converges more stably but also more slowly. Therefore, we propose a trade-off heuristic of adaptively increasing the sample number or decreasing the learning rate. Specifically, in our test, we set $K_t = \min(\lfloor t^{0.175}\rfloor, 35)$ for AsyncQVI and $\alpha_t = \max(t^{-0.1}, 0.1)$ for AsyncQL-CONST, where $t$ is the iteration number. The results are depicted in Figures 5 and 6.

The above interpretation also shows that AsyncQL-CONST is a special case of AsyncQVI (with $T_\alpha$ and $K = 1$), which explains the similarity in their optimal performances. However, since AsyncQVI takes $\frac{1}{|A|}$ memory of AsyncQL, our algorithm is still preferable for high dimensional applications.

5.5. Parallel Performance

We also test the parallel speedup performance of AsyncQVI using 1, 2, 4, 8, and 16 threads (see Figure 7). The result demonstrates linear speedup.
5.6. Summary

AsyncQVI and AsyncQL-CONST have similar numerical performance, and they are faster than VRQVI, VRVI and AsyncQL-ADA. In general, async algorithms speed and scale up very well as the number of threads increases, and AsyncQVI is not an exception. On the other hand, AsyncQVI requires only $O(|S|)$ memory, which is much less than the $O(|S||A|)$ memory of the other three; recall Table 1. Therefore, AsyncQVI can solve much larger problem instances.

6. Conclusions

In this paper, we propose an async-parallel RL algorithm AsyncQVI. Under mild asynchronism conditions, our algorithm achieves near-optimal sample complexity and minimal memory requirement. To the best of our knowledge, AsyncQVI is the first async-parallel RL algorithm with convergence rate analysis and an explicit sample complexity.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, Computer Science (2013).

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, Human-level control through deep reinforcement learning, Nature 518 (2015) 529.

[3] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, et al., Massively parallel methods for deep reinforcement learning, arXiv preprint arXiv:1507.04296 (2015).
[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, pp. 1928–1937.

[5] J. N. Tsitsiklis, Asynchronous stochastic approximation and q-learning, Machine Learning 16 (1994) 185–202.

[6] M. Grounds, D. Kudenko, Parallel reinforcement learning with linear function approximation, in: Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning, Springer, 2008, pp. 60–74.

[7] D. P. Bertsekas, J. N. Tsitsiklis, Some aspects of parallel and distributed iterative algorithms: a survey, Automatica 27 (1991) 3–21.

[8] A. Sidford, M. Wang, X. Wu, L. Yang, Y. Ye, Near-optimal time and sample complexities for solving markov decision processes with a generative model, in: Advances in Neural Information Processing Systems, pp. 5192–5202.

[9] D. P. Bertsekas, J. N. Tsitsiklis, Parallel and distributed computation: numerical methods, volume 23, Prentice hall Englewood Cliffs, NJ, 1989.

[10] E. Even-Dar, Y. Mansour, Learning rates for q-learning, Journal of Machine Learning Research 5 (2003) 1–25.

[11] M. G. Azar, R. Munos, M. Ghavamzadeh, H. Kappen, Speedy q-learning, in: Advances in neural information processing systems.

[12] D. Kalathil, V. S. Borkar, R. Jain, Empirical q-value iteration, arXiv preprint arXiv:1412.0180 (2014).

[13] A. Sidford, M. Wang, X. Wu, Y. Ye, Variance reduced value iteration and faster algorithms for solving markov decision processes, in: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, pp. 770–787.

[14] C. Jin, Z. Allen-Zhu, S. Bubeck, M. I. Jordan, Is q-learning provably efficient?, in: Advances in Neural Information Processing Systems, pp. 4868–4878.

[15] M. Kearns, Y. Mansour, A. Y. Ng, A sparse sampling algorithm for near-optimal planning in large markov decision processes, Machine learning 49 (2002) 193–208.

[16] M. G. Azar, R. Munos, H. J. Kappen, Minimax pac bounds on the sample complexity of reinforcement learning with a generative model, Machine learning 91 (2013) 325–349.

[17] Z. Peng, Y. Xu, M. Yan, W. Yin, Arock: an algorithmic framework for asynchronous parallel coordinate updates, SIAM Journal on Scientific Computing 38 (2016) A2851–A2879.

[18] R. Hannah, W. Yin, On unbounded delays in asynchronous parallel fixed-point algorithms, Journal of Scientific Computing 76 (2018) 299–326.

[19] R. Bellman, On the theory of dynamic programming, Proceedings of the National Academy of Sciences 38 (1952) 716–719.
[20] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2014.

[21] R. Hannah, W. Yin, More iterations per second, same quality–why asynchronous algorithms may drastically outperform traditional ones, arXiv preprint arXiv:1708.05136 (2017).

[22] H. R. Feyzmahdavian, M. Johansson, On the convergence rates of asynchronous iterations, in: Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on, IEEE, pp. 153–159.

[23] W. Hoeffding, Probability inequalities for sums of bounded random variables, Journal of the American statistical association 58 (1963) 13–30.

[24] M. Wang, Randomized linear programming solves the discounted markov decision problem in nearly-linear (sometimes sublinear) running time, arXiv preprint arXiv:1704.01869 (2017).

[25] R. Vanderbei, Sailing strategies, an application involving stochastics, optimization, and statistics, 1996.
A. Missing Proof of Proposition 3.3

For analysis, we sort \( \mathcal{T}^i \) into a sequence \( \{t^*_k\}_{k \geq 0} \), where \( t^*_0 \) is the first element of \( \mathcal{T}^i \) and \( t^*_k \) is the \((k+1)\)th. Then Theorem A.1 bounds \( |x_i(t) - x^*_i| \) in a staircase decreasing way: \( |x_i(t) - x^*_i| \) will contract when \( t \in \mathcal{T}^i \), or equivalently, \( t = t^*_k \) for some \( k \).

**Theorem A.1** (Staircase Decreasing). Consider the iteration (4) under Assumption 3.2. Suppose that \( G \) is \( \gamma \)-contractive under infinity norm and \( x^* \) is the fixed-point of \( G \). For each \( t \geq 1 \) and \( i \in \{1, 2, \cdots, n\} \), if \( t \in (t^*_k, t^*_k+1] \) for some \( k \), then \( x_i(t) \) satisfies

\[
|x_i(t) - x^*_i| \leq \|x(0) - x^*\|_{\infty} \rho^{t^*_k-B_1},
\]

where \( \rho := \gamma \frac{1}{b_1 + b_2 - 1} \).

**Proof.** We first claim that for each \( t \geq B_1 \) and \( i \in \{1, 2, \cdots, n\} \), there exists some \( k \geq 0 \) such that \( t \in (t^*_k, t^*_k+1] \). This follows from Assumption 3.2 (a), where \( t^*_0 \leq B_1 - 1, \forall i \).

Now we prove Eq. (14) by induction. One could check

\[
\|x(t) - x^*\|_{\infty} \leq \|x(0) - x^*\|_{\infty}, \forall t \geq 0
\]
as a corollary of [22, Theorem 2] or by another induction. We skip the details here. Thus for the basic case,

\[
\max_{0 \leq t \leq B_1} \{\|x(t) - x^*\|_{\infty}\rho^{-t}\} \leq \max_{0 \leq t \leq B_1} \{\|x(0) - x^*\|_{\infty}\rho^{-t}\} \leq \|x(0) - x^*\|_{\infty}\rho^{-B_1},
\]

which gives that for each \( t \leq B_1 \) and \( i \in \{1, 2, \cdots, n\} \),

\[
|x_i(t) - x^*_i| \leq \|x(0) - x^*\|_{\infty}\rho^{t-B_1}.
\]

Since \( \rho^t \) is decreasing, we can further obtain that

\[
|x_i(B_1) - x^*_i| \leq \|x(0) - x^*\|_{\infty}\rho^{B_1},
\]

if \( B_1 \in (t^*_k, t^*_k+1] \) for some \( k \).

For the induction step, we assume that Eq. (14) holds for all \( t \geq B_1 \) up to some \( t' \). For a fixed \( i \in \{1, 2, \cdots, n\} \), suppose that \( t' \in (t^*_k, t^*_k+1] \) for some \( k \), then we analyze the scenario at \((t' + 1)\) as two cases.

**Case 1:** \( t' \notin \mathcal{T}^i \), i.e., we do not update coordinate \( i \) at iteration \( t' \). Hence, \( x_i(t' + 1) = x_i(t') \) and \( t' + 1 \in (t^*_k, t^*_k+1] \). Then Eq. (14) follows directly.

**Case 2:** \( t' \in \mathcal{T}^i \), i.e., the \( i \)th coordinate is updated at iteration \( t' \) and \( t' = t^*_k+1 \). Since \( G \) is \( \gamma \)-contractive under infinity norm, we have

\[
|x_i(t' + 1) - x^*_i| = |G_i(\tilde{x}(t)) - x^*_i| \leq \|G(\tilde{x}(t)) - x^*\|_{\infty}
\]

\[
\leq \gamma \max_j \left| x_j(t' + 1) - x^*_j \right|.
\]

For a fixed \( j \in \{1, 2, \cdots, n\} \), suppose that \( \tau_j(t') \in (t^*_k, t^*_k+1] \) for some \( k \). Then the induction hypothesis gives

\[
|x_j(\tau_j(t')) - x^*_j| \leq \|x(0) - x^*\|_{\infty}\rho^{t^*_k-B_1}.
\]

Since \( \tau_j(t') \leq t^*_k + B_1 \) by Assumption 3.2 (a) and \( \tau_j(t') \geq t' - B_2 + 1 \) by Assumption 3.2 (b), we obtain

\[
\gamma |x_j(\tau_j(t')) - x^*_j| \leq \gamma \|x(0) - x^*\|_{\infty}\rho^{t^*_k-B_1} \leq \|x(0) - x^*\|_{\infty}\rho^{t' - 2B_1 - 2B_1}
\]

\[
= \|x(0) - x^*\|_{\infty}\rho^{t^*_k+1-B_1},
\]

(16)
where the equality holds since $\gamma = \rho^{B_1+B_2-1}$ by definition and $t' = t_{k'+1}^i$. Notice that $t' + 1 \in (t_k^i, t_{k+1}^i, t_{k+2}^i]$. Inserting Eq. (16) back into Eq. (15) yields the desired result.

This completes the proof. □

Note that if $t \in (t_k^i, t_{k+1}^i]$, then $t_k^i + B_1 \geq t$ by Assumption 3.2 (a). Hence, Proposition 3.3 is a direct consequence of Theorem A.1.

B. Missing Proof of Theorem 4.8

After $L$ iterations, AsyncQVI returns a policy $\pi(L)$ with $\pi_i(L) = \text{argmax}_{a \in A} Q_i(a, L)$. To show that $\pi(L)$ is $\varepsilon$-optimal, we first define a policy operator.

**Definition B.1 (Policy Operator).** Given a policy $\pi$ and a vector $v \in \mathbb{R}^{|S|}$, the policy operator $T_\pi: \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|S|}$ is defined as

$$[T_\pi v]_i = r_i^{\pi_i} + \gamma p_{i,a}^{\pi_i} v = r_i^{\pi_i} + \gamma \sum_{j=1}^{|S|} p_{ij}^{\pi_i} v_j.$$  \hspace{1cm} (17)

**Proposition B.2 ($T_\pi$’s Properties).** Given a policy $\pi$, for any vectors $v, v' \in \mathbb{R}^S$,

(a) **Monotonicity:** if $v \leq v'$, then $T_\pi v \leq T_\pi v'$.

(b) **$\gamma$-Contraction:** $\|T_\pi v - T_\pi v'\|_\infty \leq \gamma \|v - v'\|_\infty$.

(c) $v^\pi$ is the fixed-point of $T_\pi$.

The proof is straightforward following the definition. We skip the details here.

**Lemma B.3.** [13] Given a policy $\pi$, for any vector $v \in \mathbb{R}^S$, if there exists a $v' \in \mathbb{R}^S$ such that $v' \leq v$ and $v \leq T_\pi v'$, then $v \leq v^\pi$.

**Proof.** By Proposition B.2 (a) and $v' \leq v$, we first have $T_\pi v' \leq T_\pi v$. Combining with $v \leq T_\pi v'$, we further obtain $v \leq T_\pi v$. By induction, one can check $v \leq T_n^\pi v$, $\forall n \in \mathbb{N}$. Moreover, since $T_\pi$ is a $\gamma$-contraction, $v^\pi = \lim_{n \rightarrow \infty} T_n^\pi v$. Hence, $v \leq \lim_{n \rightarrow \infty} T_n^\pi v = v^\pi$. □

Next, we consider the special case that $v(L)$ and $\pi(L)$ are both derived from AsyQVI with $\pi_i(L) = \text{argmax}_a Q_i(a, L), v_i(L) = \text{argmax}_a Q_i(a, L), \forall i \in S$.

If $\|v^* - v^\pi\|_\infty \leq \varepsilon$, then $\pi$ is $\varepsilon$-optimal. To achieve this, we first show that $v(L)$ satisfies Lemma B.3 (see Lemma B.4). Then with Theorem 4.7, $\|v^* - v^\pi\|_\infty \leq \|v^* - v(L)\|_\infty \leq \varepsilon$.

**Lemma B.4.** Under Assumption 3.2, AsyncQVI generates a sequence of $\{v(t)\}_{t=1}^L$ and $\{\pi(t)\}_{t=1}^L$ satisfying

$$v(t - 1) \leq v(t) \leq T_{\pi(t)} v(t - 1), \quad 1 \leq t \leq L$$

with probability at least $1 - \delta$.

**Proof.** By Corollary 4.5,

$$|r(t) + \gamma S(t) - r_{it}^{\pi_i} - \gamma p_{it}^{\pi_i} v(t)| \leq \frac{(1 - \gamma)\varepsilon}{4}, \quad 0 \leq t \leq L - 1$$

holds with probability at least $1 - \delta$. Denote by $E_2$ the event

$$\left\{ r(t) + \gamma S(t) - \frac{(1 - \gamma)\varepsilon}{4} \leq r_{it}^{\pi_i} + \gamma p_{it}^{\pi_i} v(t), \forall 0 \leq t \leq L - 1 \right\}.$$
Then $\mathcal{E}_2$ occurs with probability at least $1 - \delta$.

Now we condition on $\mathcal{E}_2$ and prove Eq. (18) by induction. For simplicity, we let $v(-1) = v(0) = 0$ and start our proof from $t = 0$. Then the basic case holds. For the induction step, suppose that Eq. (18) is true for all $t$ up to some $t'$. Recall that in AsyncQVI, for each iteration, whether $v_i$ or $\pi_i$ will be updates depends on the value of $Q_{i,a}$. We hence analyze the scenario at $(t' + 1)$ as two cases.

**Case 1:** $Q_{i',a'}(t' + 1) \leq v_{i'}(t')$. Then $v$ and $\pi$ will not be updated, i.e., $v(t' + 1) = v(t')$ and $\pi(t' + 1) = \pi(t')$. In this case, the inequality $v(t') \leq v(t' + 1)$ follows directly. For the other part, by induction hypothesis we have

$$v(t' + 1) = v(t') \leq T_{\pi(t')} v(t' - 1) = T_{\pi(t' + 1)} v(t' - 1)$$

where the last inequality comes from $v(t' - 1) \leq v(t')$ and the monotonicity of $T_{\pi(t' + 1)}$.

**Case 2:** $Q_{i',a'}(t' + 1) > v_{i'}(t')$. Then $\forall i \in S$,

**Case 2.1:** $i \neq i'$. In this case, $v_i(t' + 1) = v_i(t')$ and $\pi_i(t' + 1) = \pi_i(t')$. Hence, once again by induction hypothesis and $T_\pi$’s monotonicity, we obtain

$$v_i(t' + 1) = v_i(t') \leq [T_{\pi(t')} v(t' - 1)]_i = [T_{\pi(t' + 1)} v(t' - 1)]_i$$

By definition of $\mathcal{E}_2$, we obtain $v_i(t' + 1) \leq \bar{r}^a_i + \gamma S(t') - \frac{(1 - \gamma)\varepsilon}{4}$. Since $\pi_i(t' + 1) = a'$. Owing to $\hat{v}(t') \leq v(t')$ by induction hypothesis and the monotonicity of $T_{\pi(t' + 1)}$, we can complete our proof by

$$v_i(t' + 1) \leq [T_{\pi(t' + 1)} \hat{v}(t')]_i \leq [T_{\pi(t' + 1)} v(t')]_i.$$ 

Finally, combining the results of Lemma B.3, Lemma B.4 and Theorem 4.7, we can establish Theorem 4.8 at ease.