AsyncQVI: Asynchronous-Parallel Q-Value Iteration for Discounted Markov Decision Processes with Near-Optimal Sample Complexity

Yibo Zeng
Fudan University
Columbia University

Fei Feng
University of California,
Los Angeles

Wotao Yin
University of California,
Los Angeles

Abstract

In this paper, we propose AsyncQVI, an asynchronous-parallel Q-value iteration for discounted Markov decision processes whose transition and reward can only be sampled through a generative model. Given such a problem with $|\mathcal{S}|$ states, $|\mathcal{A}|$ actions, and a discounted factor $\gamma \in (0,1)$, AsyncQVI uses memory of size $\mathcal{O}(|\mathcal{S}|)$ and returns an ε-optimal policy with probability at least $1-\delta$ using

$$\tilde{\mathcal{O}}\big(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^5\varepsilon^2}\log(\frac{1}{\delta})\big)$$

samples.[1] AsyncQVI is also the first asynchronous-parallel algorithm for discounted Markov decision processes that has a sample complexity, which nearly matches the theoretical lower bound. The relatively low memory footprint and parallel ability make AsyncQVI suitable for large-scale applications. In numerical tests, we compare AsyncQVI with four sample-based value iteration methods. The results show that our algorithm is highly efficient and achieves linear parallel speedup.

1 Introduction

Markov Decision Processes (MDPs) are a fundamental model to encapsulate sequential decision making under uncertainty. They have been indepthly studied and successfully applied to many fields, especially Reinforcement Learning (RL). As a rapidly developing area of artificial intelligence, RL is being flourishingly combined with deep neural network (Mnih et al., 2015, 2016; Li, 2017) and used in many domains including games (Mnih et al., 2015; Silver et al., 2016), robotics (Kober et al., 2013), natural language processing (Young et al., 2018), finance (Deng et al., 2016), healthcare (Kosorok and Moodie, 2015) and so on. With the advent of big-data applications, computational costs have increased significantly. Therefore, parallel computing techniques have been applied to reduce RL solving time (Grounds and Kudenko, 2008; Nair et al., 2015). Recently, asynchronous (async) parallel algorithms have been widely researched in RL and gained empirical success (Mnih et al., 2016; Babaeizadeh et al., 2016; Gu et al., 2017; Stooke and Abbeel, 2018; Zhang et al., 2019). Compared to synchronous (sync) parallel algorithms, where the agents must wait for the slowest agent to finish its 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 (Bertsekas and Tsitsiklis, 1991): the system is more tolerant of computing faults and communications glitches; it is also easy to incorporate new agents.

In contrast to promising empirical results in async-parallel RL, its theoretical property has not been fully understood. In this paper, we are trying to mitigate the gap between theory and practice. Specifically, we will asynchronous-parallelly solve Discounted Infinite-Horizon Markov Decision Processes (DMDPs) which is not fully known in advance. A DMDP is described by a tuple $(\mathcal{S}, \mathcal{A}, \mathrm{P}, \mathrm{r}, \gamma)$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, P contains the transition probabilities, r is the collection of instant rewards, and $\gamma \in (0,1)$ is a discounted factor. At time step t, the controller 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 π,

[1] We use $\tilde{\mathcal{O}}$ to omit polylogarithmic factors, i.e., $\tilde{\mathcal{O}}(f) = \mathcal{O}(f \cdot (\log f)^{\mathcal{O}(1)})$.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

where π maps a state to an action. The action leads the environment to a next state s_{t+1} with probability $p^{a_t}_{s_t s_{t+1}}$. Meanwhile, the controller receives an instant reward $r^{a_t}_{s_t s_{t+1}}$. Here, $r^{a_t}_{s_t s_{t+1}}$ is a deterministic value given the transitional instance (s_t, a_t, s_{t+1}). If only s_t and a_t are specified, $r^{a_t}_{s_t}$ is a random variable and $r^{a_t}_{s_t} = r^{a_t}_{s_t s_{t+1}}$ with probability $p^{a_t}_{s_t s_{t+1}}$. Given a policy $\pi : \mathcal{S} \to \mathcal{A}$, we denote $\mathbf{v}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ the state-value vector of π. Specifically,

$$\mathbf{v}^\pi := [v^\pi_1, v^\pi_2, \cdots, v^\pi_{|\mathcal{S}|}]^\top,\ v^\pi_i := \mathbb{E}^\pi\big[\textstyle\sum_{t=0}^\infty \gamma^t r^{a_t}_{s_t s_{t+1}} | s_0 = i\big],$$

where the expectation is taken over the trajectory $(s_0, a_0, s_1, a_1, \ldots, s_t, a_t \ldots)$ following π, i.e. $a_t = \pi_{s_t}$. The objective is to seek for an optimal policy π^* such that $\mathbf{v}^\pi$ is maximized component-wisely.

In our setting, P and r are unknown, which is also the case for RL. Thus, an optimal policy cannot be obtained through dynamic programming approach but learned from transitional samples. Depending on the applications, one can have access to either trajectories samples or a *generative model*. Specifically, given any state-action pair (i, a), a generative model returns a next state j with probability p^a_{ij} and the instant reward r^a_{ij}. One can repeatedly call it with the same input (i, a). Although the generative model is a stronger assumption than trajectories samples, it is natural and practical in many cases. Our algorithm must access a generative model; as a benefit, the algorithm requires only $\mathcal{O}(|\mathcal{S}|)$ memory and achieves a nearly optimal sample complexity.

We use notation $\mathbf{p}^a_i := [p^a_{i1}, p^a_{i2}, \cdots, p^a_{i|\mathcal{S}|}]^\top$ and $\bar{r}^a_i := \sum_{j\in\mathcal{S}} p^a_{ij} r^a_{ij}$ and assume, without loss of generality, $r^a_{ij} \in [0, 1]$, $\forall\ i, j \in \mathcal{S}, a \in \mathcal{A}$. We let $\mathbf{v}^*$ denote the optimal value vector associated with an optimal policy π^*. A policy π is ε-optimal if $\|\mathbf{v}^* - \mathbf{v}^\pi\|_\infty \le \varepsilon$.

In this paper, we propose the algorithm Asynchronous-Parallel Q-Value Iteration (AsyncQVI), the first async-parallel RL algorithm that has a sample complexity result. AsyncQVI returns an ε-optimal policy with probability at least $1 - \delta$ using

$$\tilde{O}\Big(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^5\varepsilon^2}\log(\frac{1}{\delta})\Big)$$

samples[1], provided that each coordinate is updated at least once within $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ time and the async delay is bounded by $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$. Sidford et al. (2018a) established the lower bound on the sample complexity of any DMDP with a generative model as

$$\Omega\Big(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^3\varepsilon^2}\log(\frac{1}{\delta})\Big)$$

for finding an ε-optimal policy π with probability at least $1 - \delta$. Therefore, our result nearly matches the lower bound up to $(1-\gamma)^2$ and logarithmic factors. Besides, AsyncQVI requires only $\mathcal{O}(|\mathcal{S}|)$ memory, which is minimal possible (without using dimension reduction) to store π.

With a near-optimal sample complexity, the minimal memory requirement, and asynchronous-parallel implementation, AsyncQVI is a competitive RL algorithm.

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 ith component, respectively.

2 Related Works

AsyncQVI is not the first attempt to solve DMDP problems with asynchronous parallel. As early as in Bertsekas and Tsitsiklis (1989), the authors proposed async-parallel dynamic programming methods. They established and analyzed fundamental asynchronous models, which are characterized by coordinate update and asynchronous delay. This seminal work inspires the later study of async-parallel algorithms for DMDPs that are not fully known beforehand and can only be accessed by samples.

Tsitsiklis (1994) adapted Q-learning to async-parallel setting and provided the convergence guarantee. However, although several works have established sample complexity results for single-threaded cases (Kearns et al., 2002; Even-Dar and Mansour, 2003; Azar et al., 2011, 2013; Kalathil et al., 2014; Sidford et al., 2018a,b; Agarwal et al., 2019), there have been no such results for async-parallel algorithms. Moreover, considering the latent huge cost of taking samples, an explicit complexity result is more and more concerned and is an important algorithm comparison reference.

One may notice that to achieve promising complexity results, several works adopt the *generative model*, e.g., Kearns et al. (2002); Azar et al. (2011, 2013); Kalathil et al. (2014); Sidford et al. (2018a,b). This model is proposed by Kearns et al. (2002). It is indeed a sample oracle which takes any state-action pair (i, a) as input and returns a next state j with probability p^a_{ij} and the corresponding instant reward r^a_{ij}. Our algorithm is also built under the generative model and we develop the first async-parallel algorithm that has an explicit sample complexity.

We list related async-parallel methods for DMDPs in Table 1 and the generative model methods in Table 2. Note that some papers (Even-Dar and Mansour, 2003; Azar et al., 2011; Kalathil et al., 2014) use the word "asynchronous" for single-threaded coordinate update methods. In constrast, our algorithm is not only multi-

Table 1: Related Async-parallel Methods For DMDPs.

Algorithms	**Assumption**	**Async Delay**	**Sample Complexity**	**Memory**	**References**
Totally Async QVI	Fully known DMDP	Unbounded[2]	N/A	$\mathcal{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	Bertsekas and Tsitsiklis (1989)
Partially Async QVI	Fully known DMDP	Uniformly Bounded	N/A	$\mathcal{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	Bertsekas and Tsitsiklis (1989)
Async Q-learning	Trajectory samples	Unbounded[2]	–	$\mathcal{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	Tsitsiklis (1994)
AsyncQVI	Generative model	Uniformly Bounded	$\surd$	$\mathcal{O}(\lvert\mathcal{S}\rvert)$	This Paper

Table 2: Related Algorithms For DMDP With A Generative Model.

Algorithms	**Async**	**Sample Complexity**	**Memory**	**References**
Variance-Reduced VI	$\times$	$\tilde{O}\big(\frac{\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert}{(1-\gamma)^4\epsilon^2}\log(\frac{1}{\delta})\big)$	$\mathcal{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	Sidford et al. (2018b)
Variance-Reduced QVI	$\times$	$\tilde{O}\big(\frac{\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert}{(1-\gamma)^3\epsilon^2}\log(\frac{1}{\delta})\big)$	$\mathcal{O}(\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert)$	Sidford et al. (2018a)
AsyncQVI	$\surd$	$\tilde{O}\big(\frac{\lvert\mathcal{S}\rvert\lvert\mathcal{A}\rvert}{(1-\gamma)^5\epsilon^2}\log(\frac{1}{\delta})\big)$	$\mathcal{O}(\lvert\mathcal{S}\rvert)$	This Paper

threaded, but also allows stale information and async delay. Further, the lower sample complexities achieved by Sidford et al. (2018a,b) rely on the *variance reduction* technique, which requires periodic synchronization and $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ memory footprint to update and store a basis, say $\mathbf{p}_i^{a\top}\mathbf{v}_0$, $\forall\ i \in \mathcal{S}, a \in \mathcal{A}$. In order to take advantage of fully async-parallel structure and achieve the minimal memory complexity $\mathcal{O}(|\mathcal{S}|)$, we do not implement variance reduction and therefore, obtain a slightly higher sample complexity.

The last thing to mention is that there are some other nice async-parallel works about fixed point problems in a Hilbert space, e.g. Peng et al. (2016); Hannah and Yin (2018), while our algorithm is based on a contraction with respect to the ℓ_∞ norm.

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 $(\mathcal{S}, \mathcal{A}, \mathrm{P}, \mathrm{r}, \gamma)$ and a policy π, we define the action-value vector $\mathbf{Q}^\pi$ with entries

$$Q^\pi_{i,a} = \mathbb{E}^\pi[\sum_{t=0}^{\infty}\gamma^t r^{a_t}_{s_t s_{t+1}} \big|\ s_0 = i, a_0 = a].$$

For an optimal policy π^*, we let $\mathbf{Q}^*$ denote the corresponding optimal action-value vector. From $\mathbf{Q}^*$, we can obtain $\forall\ i \in \mathcal{S}$, $\pi^*_i = \arg\max_a Q^*_{i,a}$, $v^*_i = \max_a Q^*_{i,a}$. Hence, to derive an optimal policy π^*, it suffices to compute the optimal action-value vector $\mathbf{Q}^*$. To reach this end, we first define an operator $T : \mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ as

$$[T\mathbf{Q}]_{i,a} = \underbrace{\sum_{j\in\mathcal{S}} p^a_{ij} r^a_{ij}}_{\text{expected instant reward}} + \underbrace{\gamma\sum_{j\in\mathcal{S}} p^a_{ij}\max_{a'} Q_{j,a'}}_{\text{expected discounted future reward}}, \tag{1}$$

where $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ and $[T\mathbf{Q}]_{i,a}$ is the $((i-1)\times|\mathcal{A}|+a)$th component of $T\mathbf{Q}$ with $1 \le i \le |\mathcal{S}|, 1 \le a \le |\mathcal{A}|$. Actually, T is the well-known Bellman operator. It is an $\gamma-$contraction under ℓ_∞ norm and $\mathbf{Q}^*$ is the unique fixed point (see e.g. (Puterman, 2014)). Therefore, one can apply fixed-point iterations of T to recover $\mathbf{Q}^*$. Next, we introduce the async-parallel coordinate update fashion of fixed-point iterations.

3.2 Asynchronous-Parallel Coordinate Updates

Given an ℓ_∞ $\gamma-$ contraction $G : \mathbb{R}^n \to \mathbb{R}^n$, the fixed-point iteration $\mathbf{x}(t+1) = G(\mathbf{x}(t))$, $t \ge 0$ converges linearly. Rewriting $G\mathbf{x}$ as $(G_1\mathbf{x}, \ldots, G_n\mathbf{x})$, we call

$$x_i(t+1) = \begin{cases} G_i(\mathbf{x}(t)), & t \in \mathscr{T}^i; \\ x_i(t), & t \notin \mathscr{T}^i, \end{cases} \tag{2}$$

the coordinate update of $G\mathbf{x}$, where $x_i(t)$ is the ith coordinate of $\mathbf{x}$ at iteration t and

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

is the set of iterations at which x_i is updated.

[2]Under the assumption: $\forall i, j$, $\lim_{t\to\infty}\tau^i_j(t) = \infty$ holds with probability 1.

Algorithm 1: Asynchronous-Parallel Coordinate Updates

```
1 Shared variables: $\mathbf{x}^0$, $L > 0$, $t \leftarrow 0$;
2 Private variable: $\hat{\mathbf{x}}$;
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 $\hat{\mathbf{x}} \leftarrow \mathbf{x}$;
6     perform an update $x_i \leftarrow G_i(\hat{\mathbf{x}})$;
7     increment the global counter $t \leftarrow t + 1$;
```

We use a set of computing agents to perform coordinate update (2) 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 Hannah and Yin (2017).

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

By Line 6 in Algorithm 1, the tth update can be written as

$$x_i(t+1) = \begin{cases} G_i(\hat{\mathbf{x}}(t)), & t \in \mathscr{T}^i; \\ x_i(t), & t \notin \mathscr{T}^i. \end{cases} \tag{3}$$

Here, $\hat{\mathbf{x}}(t) := [x_1(\tau_1(t)), \ldots, x_n(\tau_n(t))]^\top$ represents the possibly stale information, 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 $0 \le \tau_j(t) \le t$. The difference $t - \tau_j(t)$ is called the *delay*. In this paper, we assume *partial asynchronism* (Bertsekas and Tsitsiklis, 1989):

Assumption 3.1 (Partial Asynchronism[3]). *For the async-parallel algorithm, there exists two positive integers B_1, B_2 (asynchronism measure) such that:*

(a) For every i and for every $t \ge 0$, at least one of the elements of the set $\{t, t+1, \ldots, t+B_1-1\}$ belongs to $\mathscr{T}^i$;

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

[3]Assumption 1.1 in Bertsekas and Tsitsiklis (1989, 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.

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

```
Input: $\varepsilon \in (0, (1-\gamma)^{-1})$, $\delta \in (0,1)$, $L$, $K$;
1 Shared variables: $\mathbf{v} \leftarrow \mathbf{0}$, $\pi \leftarrow \mathbf{0}$, $t \leftarrow 0$;
2 Private variables: $\hat{\mathbf{v}}, r, S, q$;
3 while $t < L$, every agent asynchronously do
4     select state $i_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$;
5     copy shared variable to local memory $\hat{\mathbf{v}} \leftarrow \mathbf{v}$;
6     call GM$(s_t, a_t)$ $K$ times and collect samples $s'_1, \ldots, s'_K$ and $r_1, \ldots, r_K$;
7     $q \leftarrow \frac{1}{K}\sum_{k=1}^K r_k + \gamma \frac{1}{K}\sum_{k=1}^K \hat{v}_{s'_k} - \frac{(1-\gamma)\varepsilon}{4}$;
8     if $q > v_{i_t}$ then
9         mutex lock;
10        $v_{i_t} \leftarrow q$, $\pi_{i_t} \leftarrow a_t$;
11        mutex unlock;
12    increment the global counter $t \leftarrow t + 1$;
13 return $\pi$
```

Assumption 3.1 (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. Convergence under this assumption was established in Feyzmahdavian and Johansson (2014).

Proposition 3.2. (Feyzmahdavian and Johansson, 2014, Theorem 2) *Consider the iterations Eq. (3) under Assumption 3.1. Suppose that G is γ-contractive under ℓ_∞ norm and $\mathbf{x}^*$ is the fixed point of G. Then $\|\mathbf{x}(t) - \mathbf{x}^*\|_\infty \le \|\mathbf{x}(0) - \mathbf{x}^*\|_\infty \rho^{t-2B_1}$ for all $t \ge B_1$, where $\rho = \gamma^{\frac{1}{B_1+B_2-1}}$.*

In many DMDP and RL problems, the transition probabilities P are sparse. So for any state-action pair (i, a), the possible next states form a tiny subset of $\mathcal{S}$. Hence, to apply async-parallel coordinate updates to Eq. (1), very few components are required and we only need to bound async dealy over a smaller subset. Therefore, we usually have $B_2 \ll B_1$, where $B_1 \ge |\mathcal{S}||\mathcal{A}|$. Hence, the convergence rate $\gamma^{\frac{1}{B_1+B_2-1}}$ we obtain is significantly better than $\gamma^{\frac{1}{2(B_1 \vee B_2)-1}}$ from (Feyzmahdavian and Johansson, 2014, Theorem. 2); the proof is deferred to Appendix A.

Remark 3.3 (Total Asynchronism). *Here we do not adopt the total asynchronism notion (Bertsekas and Tsitsiklis, 1989, Section 6.1). To start with, one cannot derive convergence rate results under total asynchronism since it allows arbitrarily long delays and no improvement can be said for finite iterations. On the contrary, partial asynchronism can avoid this case and be practically enforced (Bertsekas and Tsitsiklis, 1989, Section 7.1).*

4 AsyncQVI: Asynchronous-Parallel Q-value Iteration

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

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

$$Q_{i,a}(t+1) = \begin{cases} \sum_j p_{ij}^a r_{ij}^a + \gamma \sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}(t), & t \in \mathscr{T}^{i,a}; \\ Q_{i,a}(t), & t \notin \mathscr{T}^{i,a}. \end{cases} \tag{4}$$

Since there is no knowledge of the transition probability, we approximate the expectations $\sum_j p_{ij}^a \cdot$ by random sampling (Lines 6 and 7, Algorithm 2). This is done by accessing a generative model `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 (4), we substitute $\sum_j p_{ij}^a r_{ij}^a$ and $\sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}(t)$ by their empirical means, i.e., $r(t) := \frac{1}{K}\sum_k r_{i_t s_k'}^{a_t}$ and $S(\hat{\mathbf{Q}}(t)) := \frac{1}{K}\sum_k \max_{a'} \hat{Q}_{s_k',a'}(t)$, respectively. For the purpose of analysis, we also tune the update slightly by substracting a small constant $(1-\gamma)\varepsilon/4$. Consequently, AsyncQVI is equivalent to

$$Q_{i,a}(t+1) = \begin{cases} r(t) + \gamma S(\hat{\mathbf{Q}}(t)) - (1-\gamma)\varepsilon/4 & t \in \mathscr{T}^{i,a}; \\ Q_{i,a}(t), & t \notin \mathscr{T}^{i,a}. \end{cases} \tag{5}$$

For memory efficiency, we do not form $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. Instead, since only the values $\max_{a'} Q_{i,a'}$ are used for update, we maintain two vectors $\mathbf{v}, \pi \in \mathbb{R}^{|\mathcal{S}|}$; 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 $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ to $\mathcal{O}(|\mathcal{S}|)$, which is of a great advantage in real applications.

Remark 4.1 (Coordinate Selection)**.** *To guarantee convergence, the coordinate should be selected to satisfy Assumption 3.1. In practice, however, if all agents have similar powers, one can simply apply either uniformly random or globally cyclic selections.*

Remark 4.2 (Parallel Overhead)**.** *In AsyncQVI, overhead can only occur during copying variable from global memory to the local memory (Line 5) and where a memory lock is implemented (Lines 8-11). For the former case, the time complexity is $\mathcal{O}(|\mathcal{S}|)$, which is negligible when $\mathcal{O}(|\mathcal{S}|)$ is small or the process of querying samples is much slower. Otherwise, one can consider copying in a less frequent fashion, i.e., updating $\hat{\mathbf{v}}$ every l_0 iterations. Although it will increase B_2 by l_0, the sample complexity is still near-optimal as long as $B_1 + B_2 = \mathcal{O}(|\mathcal{S}||\mathcal{A}|)$. For the latter case, a memory lock (e.g. mutex) ensures that v_i and π_i are indeed the maximum value and a maximizer of the vector $\mathbf{Q}_i$, respectively. Since only two scalars are accessed and altered, the collision is rare.*

4.1 Convergence Analysis

Next, we establish convergence for AsyncQVI; all proofs in this section are deferred to Appendix B. To distinguish different sequences, we let $(\mathbf{Q}^{\mathbb{E}}(t))$ denote the asynchronous coordinate update sequence generated through Eq. (4), where the superscript represents the updates with real expectations. Specifically, if AsyncQVI produces a sequence according to Eq. (5) with $\hat{\mathbf{Q}}(t) = [Q_{1,1}(\tau_{1,1}(t)), \ldots, Q_{|\mathcal{S}|,|\mathcal{A}|}(\tau_{|\mathcal{S}|,|\mathcal{A}|}(t))]^\top$, then

$$Q_{i,a}^{\mathbb{E}}(t+1) = \begin{cases} \bar{r}_i^a + \gamma \sum_j p_{ij}^a \max_{a'} \hat{Q}_{j,a'}^{\mathbb{E}}(t), & t \in \mathscr{T}^{i,a}; \\ Q_{i,a}^{\mathbb{E}}(t), & t \notin \mathscr{T}^{i,a}, \end{cases} \tag{6}$$

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

(i) $(\mathbf{Q}^{\mathbb{E}}(t))_{t=0}^L$ and $(\mathbf{Q}(t))_{t=0}^L$ have the same initial point;

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

These properties are important to our analysis. Recall that we assume partial asynchronism (Assumption 3.1) for AsyncQVI. Then Eq. (6) also meets Assumption 3.1. Hence, Eq. (6) converges following the fact that T is a $\gamma-$contraction and Proposition 3.2. Since Eq. (5) is an approximation of Eq. (6), we can leverage the convergence of Eq. (6) to establish the convergence of AsyncQVI. To this end, we first use Hoeffeding's Inequality (Hoeffding, 1963) to analyze the sampling error. Specifically, if we take enough samples per iteration, then the error can be controlled with high probability.

Proposition 4.3 (Sample Concentration)**.** *With $K = \left\lceil \frac{8}{(1-\gamma)^4\varepsilon^2} \log\left(\frac{4L}{\delta}\right) \right\rceil$, AsyncQVI generates a sequence $(r(t), S(\hat{\mathbf{Q}}(t)))_{t=0}^{L-1}$ that satisfies $\left| r(t) + \gamma S(\hat{\mathbf{Q}}(t)) - \bar{r}_{i_t}^{a_t} - \gamma \mathbf{p}_{i_t}^{a_t\top} \hat{\mathbf{v}}(t) \right| \le \frac{(1-\gamma)\varepsilon}{4}$, $\forall\ 0 \le t \le L-1$, with probability at least $1-\delta$.*

Proposition 4.3 indeed establishes a control over a one-step approximation error between Eq. (5) and Eq. (6) provided that $\hat{\mathbf{Q}} = \hat{\mathbf{Q}}^{\mathbb{E}}$. However, for the two sequences $(\mathbf{Q}(t))$ and $(\mathbf{Q}^{\mathbb{E}}(t))$ that only share the same initial point, the error can accumulate. To tackle this issue, we further utilize the γ-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 can control the difference between $(\mathbf{Q}(t))$ and $(\mathbf{Q}^{\mathbb{E}}(t))$ by induction.

Proposition 4.4. *Given the total iteration number L, accuracy parameters ε and δ, with $K=\lceil\frac{8}{(1-\gamma)^4\varepsilon^2}\log\left(\frac{4L}{\delta}\right)\rceil$, AsyncQVI can generate a sequence $(\mathbf{Q}(t))_{t=1}^{L}$ satisfying $\|\mathbf{Q}(t)-\mathbf{Q}^{\mathbb{E}}(t)\|_\infty\le\varepsilon/2,\ \forall\ 1\le t\le L$ with probability at least $1-\delta$.*

Since $(\mathbf{Q}^{\mathbb{E}}(t))$ converges to $\mathbf{Q}^*$ linearly, combining Propositions 3.2 and 4.4 gives the desired result.

Theorem 4.5 (Linear Convergence)**.** *Under Assumption 3.1, given accuracy parameters ε and δ, with $L=\lceil 2B_1+\frac{B_1+B_2-1}{1-\gamma}\log\left(\frac{2}{(1-\gamma)\varepsilon}\right)\rceil$ and $K=\lceil\frac{8}{(1-\gamma)^4\varepsilon^2}\log\left(\frac{4L}{\delta}\right)\rceil$, AsyncQVI can produce $\mathbf{Q}(L)\in\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ and $\mathbf{v}(L)\in\mathbb{R}^{|\mathcal{S}|}$ satisfying $\|\mathbf{Q}^*-\mathbf{Q}(L)\|_\infty\le\varepsilon$ and $\|\mathbf{v}^*-\mathbf{v}(L)\|_\infty\le\varepsilon$ with probability at least $1-\delta$.*

4.2 ε-optimal Policy

In the following theorem, we show that the vector π maintained through the iterations is an ε-optimal policy; the proof is deferred to Appendix C. Using this theorem, we shall present the sample complexity of AsyncQVI in Corollary 4.7.

Theorem 4.6. *Under Assumption 3.1, given accuracy parameters ε and δ, with $L=\lceil 2B_1+\frac{B_1+B_2-1}{1-\gamma}\log\left(\frac{2}{(1-\gamma)\varepsilon}\right)\rceil$ and $K=\lceil\frac{8}{(1-\gamma)^4\varepsilon^2}\log\left(\frac{4L}{\delta}\right)\rceil$, AsyncQVI returns an ε-optimal policy π with probability at least $1-\delta$.*

Corollary 4.7. *Under Assumption 3.1, AsyncQVI returns an ε-optimal policy π with probability at least $1-\delta$ at the sample complexity*

$$\tilde{\mathcal{O}}\Big(\frac{B_1+B_2}{(1-\gamma)^5\varepsilon^2}\log(\frac{1}{\delta})\Big).$$

Hence, if $B_1+B_2=\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$, then AsyncQVI has a near-optimal sample complexity.

Moreover, given the complete knowledge of transition P and reward r, we can also solve it asynchronous parallelly. To utilize AsyncQVI, one can build a *generative model* in $\tilde{O}(|\mathcal{S}|^2|\mathcal{A}|)$ prepossessing time (Wang, 2017), and the `GM` produces a sample in $\tilde{O}(1)$ arithmetic operations. In this sense, AsyncQVI also has the following computational complexity results.

Corollary 4.8 (Computational Complexity)**.** *Given a DMDP $(\mathcal{S},\mathcal{A},\mathrm{P},\mathrm{r},\gamma)$, under Assumption 3.1 AsyncQVI returns an ε-optimal policy with probability at least $1-\delta$ at the computational complexity*

$$\tilde{\mathcal{O}}\big(|\mathcal{S}|^2|\mathcal{A}|+\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^5\varepsilon^2}\log(\frac{1}{\delta})\big),$$

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

5 Numerical Experiments

5.1 Sailing Problem

To investigate the performance of AsyncQVI, we solve the sailing problem from Vanderbei (1996) on a 100×100 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 (δ_x,δ_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 (w.p.) 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 $(x+\delta_x+\mathcal{N}(0,\sigma_1^2),\ y+\delta_y+\mathcal{N}(0,\sigma_1^2))$ w.p. $1-p$ and $(x+\delta_x+\mathcal{N}(0,\sigma_1^2+\sigma_2^2),\ y+\delta_y+\mathcal{N}(0,\sigma_1^2+\sigma_2^2))$ w.p. p. The wind direction at next time maintains its current direction w.p. 0.3, changes 45 degrees to either direction w.p. 0.2 each direction, changes 90 degrees to either direction w.p. 0.1 each, changes 135 degrees to either direction w.p. 0.04 each, and reverses direction w.p. 0.02. We set the instant reward as $d\times|\frac{\text{angle between wind and action directions}}{45}|$, 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 ($\mathcal{SO}$): AsyncQVI, Asynchronous-Parallel Q-learning with constant stepsize (AQLC), Asynchronous-Parallel Q-learning with diminishing stepsize (AQLD)(Tsitsiklis, 1994), Variance-reduced Value Iteration (VRVI)(Sidford et al., 2018b), and Variance-reduced Q-value Iteration (VRQVI)(Sidford et al., 2018a). All algorithms and the $\mathcal{SO}$ 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 AQLD was set as $1/t^{0.51}$ according to its theoretical analysis, where t is the iteration number. Our code is available in `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 more flags is preferred.

We test with different randomness and rewards which represent various MDP settings (see Figure 1). 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. 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. 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. This phenomenon occurs in VRVI and VRQVI as well. We skip the detailed parameters here.

In these four tests, AsyncQVI and AQLC are almost equivalently outstanding in terms of time and achieve an at least 10× speedup compared to VRQVI and VRVI with 20 threads running parallel. Further, 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 AQLC and provide heuristics on how to set sample number and learning rate.

5.4 Performance Analysis and Heuristics

Recall that AsyncQVI derives from the Q-value operator T (see Eq. (1)). Let $T_\alpha := (1-\alpha)I + \alpha T$, where α is the learning rate. One can get AQLC through the same approach. What's special is, AQLC 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_α, sample range is also discounted by α. For fixed δ and ϵ, the requested sample number m decreases quadratically with respect to α, since $m \geq C\frac{\alpha^2}{\epsilon^2}\log\left(\frac{1}{\delta}\right)$ by Hoeffedings Inequality (Hoeffding, 1963). Hence, when α is smaller, AQLC converges more stably. On the other hand, a tiny learning rate also leads to slow progress, since T_α'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 AQLC, where t is the iteration number. The results are depicted in Figure 2.

The above interpretation also shows that AQLC is a special case of AsyncQVI (with T_α and $K = 1$), which explains the similarity in their optimal performances. However, since AsyncQVI takes $\frac{1}{|\mathcal{A}|}\times$ memory of AQLC, 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 3). The result shows an ideal linear speedup.

5.6 Summary

AsyncQVI and AQLC have similar numerical performance, and they are faster than VRQVI, VRVI and AQLD. 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 $\mathcal{O}(|\mathcal{S}|)$ memory, which is much less than the $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ memory of the other three; recall Table 1. Therefore, AsyncQVI can solve much larger problem instances.

6 Conclusions and Future Work

In this paper, we propose an async-parallel 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 algorithm for DMDPs with a generative model that has an explicit sample complexity.

For future work, we plan to integrate function approximation and policy exploration. Recently, a line of work established sample complexity results in this direction, e.g., Yang and Wang (2019b,a); Chen et al. (2018). We will also consider extending to continuous cases.

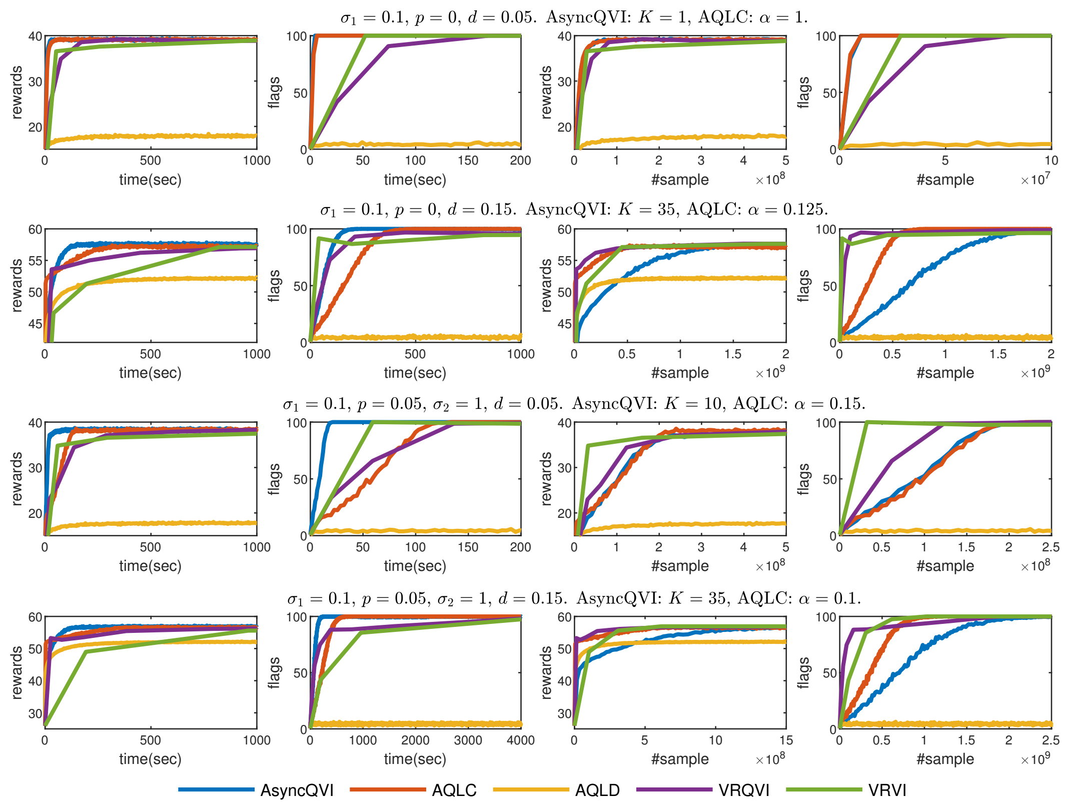

Figure 1: Performance Comparison Under Various Settings.

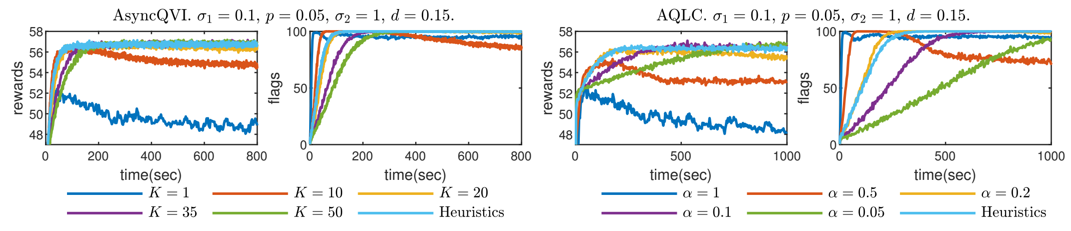

Figure 2: Performance Comparison With Different Sample Numbers Or Learning Rates.

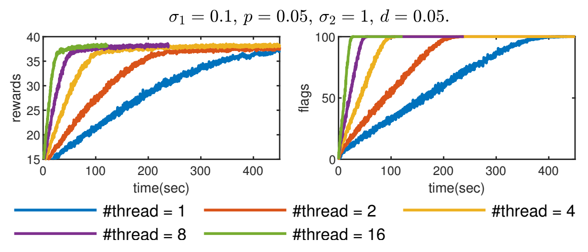

Figure 3: Parallel Performance Of AsyncQVI.

Acknowledgements

This work of Yibo Zeng is partially supported by Fudan University Exchange Program and UCEAP Reciprocal Program for his short-term visit to UCLA. The work of Fei Feng and Wotao Yin is partially supported by National Key R&D Program of China 2017YFB02029 and NSF DMS-1720237.

References

Agarwal, A., Kakade, S., and Yang, L. F. (2019). On the optimality of sparse model-based planning for markov decision processes. *arXiv preprint arXiv:1906.03804*.

Azar, M. G., Munos, R., Ghavamzadeh, M., and Kappen, H. (2011). Speedy q-learning. In *Advances in neural information processing systems*.

Azar, M. G., Munos, R., and Kappen, H. J. (2013). Minimax pac bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91(3):325–349.

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., and Kautz, J. (2016). Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1989). *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ.

Bertsekas, D. P. and Tsitsiklis, J. N. (1991). Some aspects of parallel and distributed iterative algorithms: a survey. *Automatica*, 27(1):3–21.

Chen, Y., Li, L., and Wang, M. (2018). Scalable bilinear π learning using state and action features. *arXiv preprint arXiv:1804.10328*.

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664.

Even-Dar, E. and Mansour, Y. (2003). Learning rates for q-learning. *Journal of Machine Learning Research*, 5(Dec):1–25.

Feyzmahdavian, H. R. and Johansson, M. (2014). On the convergence rates of asynchronous iterations. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 153–159. IEEE.

Grounds, M. and Kudenko, D. (2008). Parallel reinforcement learning with linear function approximation. In *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pages 60–74. Springer.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

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

Hannah, R. and Yin, W. (2018). On unbounded delays in asynchronous parallel fixed-point algorithms. *Journal of Scientific Computing*, 76(1):299–326.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30.

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

Kearns, M., Mansour, Y., and Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine learning*, 49(2-3):193–208.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Kosorok, M. R. and Moodie, E. E. (2015). *Adaptive TreatmentStrategies in Practice: Planning Trials and Analyzing Data for Personalized Medicine*, volume 21. SIAM.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., and Ostrovski, G. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*.

Peng, Z., Xu, Y., Yan, M., and Yin, W. (2016). Arock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5):A2851–A2879.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Sidford, A., Wang, M., Wu, X., Yang, L., and Ye, Y. (2018a). Near-optimal time and sample complexities for solving markov decision processes with a generative model. In *Advances in Neural Information Processing Systems*, pages 5192–5202.

Sidford, A., Wang, M., Wu, X., and Ye, Y. (2018b). 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*, pages 770–787. Society for Industrial and Applied Mathematics.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Stooke, A. and Abbeel, P. (2018). Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202.

Vanderbei, R. (1996). Sailing strategies, an application involving stochastics, optimization, and statistics.

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

Yang, L. F. and Wang, M. (2019a). Reinforcement leaning in feature space: Matrix bandit, kernels, and regret bound. *arXiv preprint arXiv:1905.10389*.

Yang, L. F. and Wang, M. (2019b). Sample-optimal parametric q-learning with linear transition models. *arXiv preprint arXiv:1902.04779*.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75.

Zhang, Y., Clavera, I., Tsai, B., and Abbeel, P. (2019). Asynchronous methods for model-based reinforcement learning. *arXiv preprint arXiv:1910.12453*.