Les Cahiers du GERAD ISSN: 0711–2440

Approximated multi-agent fitted Q iteration

A. Lesage-Landry, D.S. Callaway

G–2022–02

January 2022
Revised: May 2022

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Approximated multi-agent fitted Q iteration

Antoine Lesage-Landry [a, b]

Duncan S. Callaway [c]

[a] *Department of Electrical Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 1J4*

[b] *GERAD, Montréal (Qc), Canada, H3T 1J4*

[c] *Energy & Resources Group, University of California, Berkeley, Berkeley, USA 94720*

antoine.lesage-landry@polymtl.ca
dcal@berkeley.edu

January 2022
Revised: May 2022
Les Cahiers du GERAD
G–2022–02

Abstract : We formulate an efficient approximation for multi-agent batch reinforcement learning, the approximated multi-agent fitted Q iteration (`AMAFQI`). We present a detailed derivation of our approach. We propose an iterative policy search and show that it yields a greedy policy with respect to multiple approximations of the centralized, learned Q-function. In each iteration and policy evaluation, `AMAFQI` requires a number of computations that scales linearly with the number of agents whereas the analogous number of computations increase exponentially for the fitted Q iteration (`FQI`), a commonly used approaches in batch reinforcement learning. This property of `AMAFQI` is fundamental for the design of a tractable multi-agent approach. We evaluate the performance of `AMAFQI` and compare it to `FQI` in numerical simulations. The simulations illustrate the significant computation time reduction when using `AMAFQI` instead of `FQI` in multi-agent problems and corroborate the similar performance of both approaches.

Keywords : approximate dynamic programming, batch reinforcement learning, Markov decision process, multi-agent reinforcement learning

---

Acknowledgements: This work was funded in part by the Institute for Data Valorization (IVADO), in part by the Natural Sciences and Engineering Research Council of Canada, in part by the National Science Foundation, award 1351900, and in part by the Advanced Research Projects Agency-Energy, award DE–AR0001061.

This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at the University of California, Berkeley (supported by the UC Berkeley Chancellor, Vice Chancellor for Research, and Chief Information Officer).

1 Introduction

Reinforcement learning is a framework which considers stochastic, sequential decision-making problems with unknown dynamics [29]. These problems are modelled as Markov decision processes (MDPs). In each decision round of an MDP, a decision maker observes the current state of the system and must provide a decision or equivalently, a control. A scalar reward is subsequently revealed, and the current state shifts to a new state according to a transition function defined by the dynamics of the problem. In reinforcement learning, the transition function is unknown. Only the reward, the initial and resulting states, and the control are used to improve future controls. Batch reinforcement learning [7, 14, 21] is a subfield of reinforcement learning in which information about the system in the form of a set of historical transitions is known a priori to the decision maker. This is in contrast to typical reinforcement learning algorithms, e.g., the Q-learning algorithm [32], in which information is gathered in an online fashion. Batch reinforcement learning improves over its online counterpart (i) by reusing the gathered information multiple times (experience replay [17]) to increase the approach's convergence speed, (ii) by fitting an approximated function (e.g., Q or value functions) in between updates to mitigate instabilities, and (iii) by averaging similar transitions from the batch information to better estimate the MDP's stochastic model [14]. In batch reinforcement learning, a prevalent approach [14] is the fitted Q iteration (`FQI`) [7].

In multi-agent reinforcement learning, agents make sequential decisions to maximize their joint or individual rewards [4, 34]. The agents can be fully cooperative, i.e., maximizing a joint reward function, fully competitive, i.e., the agents' objectives are opposed, or a combination of both [4, 34]. The main challenge when considering the multi-agent reinforcement learning problem comes from the cardinality of the joint control set as it increases exponentially with the number of agents. This adds to the difficulty that the curse of dimensionality already poses to (approximate) dynamic programming-based methods [4, 27, 34]. The design of an approach that relies only on local control sets is, therefore, highly desirable to enable the implementation of batch reinforcement learning methods in real-world multi-agent systems, e.g., electric power systems [5]. For example, the approach we present in this work could extend current methods for demand response or distributed energy resource management like [19, 25, 30] to multi-agent implementations and increase the benefits for the electric grid without significantly impacting the computational cost of the approach. Other applications for multi-agent reinforcement learning include the control of a robot team [28] or of an autonomous vehicle fleet [23], autonomous driving [26], and stock trading [16]. In this work, we consider the batch reinforcement learning framework and design the approximated multi-agent fitted Q iteration (`AMAFQI`), an efficient approximation of `FQI` [7] tailored to fully cooperative, multi-agent problems.

Related work

Multi-agent reinforcement learning has been studied by many authors and the main recent advancements to this body of work are reviewed in [4, 12, 22, 33]. Multi-agent extensions to the Q-learning algorithm [32] are reviewed in [4]. Reference [33] focuses on theory-backed approaches. An overview of multi-agent deep reinforcement learning is presented in [12, 22]. In our work, we are interested in multi-agent extensions of batch reinforcement learning [14], and more specifically, of the kernel-based [21] `FQI` [7] framework. Multi-agent problems have also been studied under other reinforcement learning frameworks, e.g., classical Q-learning [2, 15] or actor-critic approaches [11, 34]. We review the literature relevant to multi-agent `FQI` next.

To the best of the authors' knowledge, the only extension of `FQI` to the multi-agent setting are presented in [8, 9, 35]. References [8, 9] only consider deterministic problems. The extension relies on the neural fitted Q (`NFQ`) algorithm [24]. The `NFQ` is a modified `FQI` approach that uses a neural network instead of a regression tree as the fitting method used to generalize the Q-value to all state-control pairs (see Section 2.1). Similarly to our approach, their work is based on the ideas of [15] in which an efficient multi-agent Q-learning algorithm [32], the Distributed Q-learning algorithm, for online, deterministic settings is presented, to obtain an approach that does not require computations over the joint control

set. The work of [8, 9] differs from ours because it uses an opportunistic approach enabled by the deterministic setting. Furthermore, [8, 9] only provide an empirical analysis of their algorithm because the properties of the neural network are hard to analyze. In our work, we (i) consider general stochastic problems, (ii) present a detailed derivation for `AMAFQI`, and (iii) provide a convergence analysis of the approximated local Q-functions used by our approach. Moreover, we characterize the performance of the greedy policy for `AMAFQI`. Lastly, [35] proposes a general, fully decentralized, multi-agent fitted Q algorithm that accounts for competitive agents and where any function approximator can be used to approximated the local Q-function. The authors further derive a finite-sample performance guarantee for their approach. However, [35]'s algorithm requires optimizing local Q-function over the joint control space which grows exponentially with the number of agents. Our main contribution is to provide an approach which uses only local control spaces.

Our specific contributions are:

- We formulate the approximated multi-agent fitted Q iteration (`AMAFQI`). `AMAFQI` is an efficient approximation of the `FQI` algorithm for multi-agent settings. In each iteration, `AMAFQI`'s computation scales linearly in the number of agents instead of exponentially as in `FQI`.
- We propose a policy search for `AMAFQI` and show that it is a greedy policy with respect to the approximation of the centralized, learned Q-functions from each agent.
- We derive a very efficient extension of `AMAFQI`, `AMAFQI-L`, that further reduces the computation requirement of the approach.
- We show the convergence of the local Q-function approximations computed by `AMAFQI` to unique and finite functions.
- We numerically evaluate the performance of `AMAFQI`. We show the similar performance and significant decrease in computation times when `AMAFQI` and `AMAFQI-L` are used instead of `FQI`.

2 Preliminaries

We consider an MDP $(\mathcal{X}, \mathcal{U}, f, r)$ where multiple agents must implement a control to maximize their expected joint cumulative reward. Let $m \in \mathbb{N}$ be the number of agents. We assume $m > 1$. Let $\mathcal{X} \subseteq \mathbb{R}^{n \times m}$, $\mathcal{U} \subseteq \mathbb{R}^{p \times m}$, and $\mathcal{W} \subseteq \mathbb{R}^{s \times m}$ where $n, p, s \in \mathbb{N}$ be the joint state, control, and disturbance space, respectively. Let $\mathbf{x} \in \mathcal{X}$ be a joint state, $\mathbf{u} \in \mathcal{U}$ be a joint control, and $\mathbf{w} \in \mathcal{W}$ be a random disturbance. Let $f : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \mapsto \mathcal{X}$ express the state transition function of the problem. The function f maps an initial state, a control and a disturbance to a resulting state. Lastly, let $r : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \mapsto \mathbb{R}$ be the function that returns the reward associated with an initial state, control, final state, and disturbance tuple. We make the following assumption regarding the reward function.

Assumption 1. *The reward function r is bounded below and above such that $0 \leq r(\mathbf{x}, \mathbf{u}, \mathbf{w}) \leq R < +\infty$ for all $(\mathbf{x}, \mathbf{u}, \mathbf{w}) \in \mathcal{X} \times \mathcal{U} \times \mathcal{W}$.*

The assumption on the upper bound of the reward function is a standard assumption for MDPs in reinforcement learning [7]. The lower bound assumption is mild because if not met, a constant can be added to the reward function so that it is non-negative. This translation does not change the optima [15].

To easily differentiate local and joint controls, we define local control variables and spaces. We let $\mathcal{A}^j \subset \mathbb{R}^p$ be the local control space of agent j where $\mathcal{U} = \times_{j=1}^{m} \mathcal{A}^j$. We denote a local control by $a \in \mathcal{A}^j$ and add the superscript j to refer to the j^{th} agent if needed.

Formally, the m agents want to cooperatively solve the following problem:

$$\max_{\{\mathbf{u}_T \in \mathcal{U}\}_{T=1}^{+\infty}} \mathbb{E}\left[\sum_{T=1}^{+\infty} \beta^T r(\mathbf{x}_T, \mathbf{u}_T, \mathbf{w}_T)\right] \tag{1}$$

where $\beta \in [0, 1)$ is the discount factor. The variables $\mathbf{u}_T$ and $\mathbf{x}_T$ represent the joint control and state at the decision round T, respectively. The random disturbance at T is represented by $\mathbf{w}_T$. Successive states are obtained from $\mathbf{x}_{T+1} = f(\mathbf{x}_T, \mathbf{u}_T, \mathbf{w}_T)$, where $\mathbf{w}_T \in \mathcal{W}$. The expectation in (1) is taken with respect to the probability of $\mathbf{w}_T$ given the state and control at round T.

We consider the batch reinforcement learning framework [7, 14, 21]. In this setting, f is unknown and only examples of past transitions can be used to solve (1). The decision makers or agents have access to batch data representing historical transitions [7]. The batch data is used to first compute an approximation of the Q-function and, second, to evaluate a policy. Let $L \in \mathbb{N}$ be the number of available samples in the batch data. The batch data set $\mathcal{S}_L$ is defined as:

$$\mathcal{S}_L = \left\{ \left(\mathbf{x}^l, \mathbf{u}^l, \mathbf{x}^l_+, r^l \right) \in \mathcal{X} \times \mathcal{U} \times \mathcal{X} \times \mathbb{R}_+, l = 1, 2, \ldots, L \right\},$$

where $\mathbf{x}^l_+$ refers to the state observed after control $\mathbf{u}^l$ was implemented in state $\mathbf{x}^l$. These samples do not need to be generated from continuous experiments. Specifically, we focus on regression tree-based `FQI` approaches [7]. `FQI` is introduced in detail in the next subsection.

2.1 Fitted Q iteration

We recall the motivation for `FQI` as presented in [7]. The state-action value or Q-function $Q : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ is the unique solution to the Bellman equation:

$$Q(\mathbf{x}, \mathbf{u}) = \mathbb{E}\left[r(\mathbf{x}, \mathbf{u}, \mathbf{w}) + \beta \max_{\mathbf{u}' \in \mathcal{U}} Q\left(f(\mathbf{x}, \mathbf{u}, \mathbf{w}), \mathbf{u}' \right) \right],$$

where $\beta \in [0, 1)$. The expectation is taken with respect to the probability of $\mathbf{w}$ given the state $\mathbf{x}$ and control $\mathbf{u}$. By the contraction mapping theorem [18], the Q-function can be obtained by successively solving

$$Q_N(\mathbf{x}, \mathbf{u}) = \mathbb{E}\left[r(\mathbf{x}, \mathbf{u}, \mathbf{w}) + \beta \max_{\mathbf{u}' \in \mathcal{U}} Q_{N-1}\left(f(\mathbf{x}, \mathbf{u}, \mathbf{w}), \mathbf{u}' \right) \right], \tag{2}$$

for all $N \geq 1$ with the boundary condition $Q_0(\mathbf{x}, \mathbf{u}) = 0$ for all $(\mathbf{x}, \mathbf{u}) \in \mathcal{X} \times \mathcal{U}$. In the deterministic case, (2) can be expressed as:

$$Q_N(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}) + \beta \max_{\mathbf{u}' \in \mathcal{U}} Q_{N-1}\left(\delta(\mathbf{x}, \mathbf{u}), \mathbf{u}' \right),$$

where $\delta : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$ is the deterministic function that returns the resulting state given a pair state-control. Given $\mathcal{S}_L$ and supposing Q_{N-1} is available, then for all data points $l = 1, 2, \ldots, L$, we can compute

$$Q_N\left(\mathbf{x}^l, \mathbf{u}^l \right) = r^l + \beta \max_{\mathbf{u}' \in \mathcal{U}} Q_{N-1}\left(\mathbf{x}^l_+, \mathbf{u}' \right), \tag{3}$$

because $r(\mathbf{x}^l, \mathbf{u}^l) = r^l$ and $\delta(\mathbf{x}^l, \mathbf{u}^l) = \mathbf{x}^l_+$. The `FQI` then works in the following way. Pairs of $\left(\mathbf{x}^l, \mathbf{u}^l \right)$ and their respective $Q_N\left(\mathbf{x}^l, \mathbf{u}^l \right)$-value can be generated using (3) for all l in the batch data. Then, an approximation $\hat{Q}_N^{\text{FQI}}(\mathbf{x}, \mathbf{u})$ of $Q_N(\mathbf{x}, \mathbf{u})$ is obtained by fitting a function over the pairs $\left(\left(\mathbf{x}^l, \mathbf{u}^l \right), Q_N\left(\mathbf{x}^l, \mathbf{u}^l \right) \right)$ for $l = 1, 2, \ldots, L$. This is done to estimate the state-action values for all state-control pairs based on the batch data. Using $\hat{Q}_{N-1}^{\text{FQI}}$ in (3) instead of Q_{N-1}, we can compute the state-action values at N, fit a function again based on the new pairs and obtain $\hat{Q}_N^{\text{FQI}}$. This process is then repeated until convergence. Finally, the authors of [7] argue that the process described above provides an adequate approximation $\hat{Q}_N^{\text{FQI}}(\mathbf{x}, \mathbf{u})$ for the stochastic case as well. In the stochastic case, the conditional expectation of (3)'s right-hand side given the current state and control is required for the update. Least squares regression [7] or the averaging at leaf nodes of regression tree methods [14] estimates the conditional expectation of the dependent variables given the independent variables, respectively the $\hat{Q}_N^{\text{FQI}}(\mathbf{x}^l, \mathbf{u}^l)$ and $\left(\mathbf{x}^l, \mathbf{u}^l, \mathbf{x}^l_+, r^l \right)$ in this setting. Least squares and tree regression methods hence approximate the right-hand side of (3) in the stochastic case [7, 14].

2.2 Regression tree methods

In this work, we use a regression tree to generalize the local Q-function and state-control pairs. Regression trees are chosen as the regression methods because (i) their properties allow us to establish the `AMAFQI`'s convergence (see Section 4) and (ii) they are computationally efficient, scalable and robust to noisy data [7]. We now introduce regression tree methods. Let $\mathcal{I} \subseteq \mathbb{R}^{n+p}$ and $\mathcal{O} \in \mathbb{R}$ be, respectively, the input and output sets of the data set $\mathcal{D} = \left\{\left(i^l, o^l\right) \in \mathcal{I} \times \mathcal{O}, l = 1, 2, \ldots, L\right\}$. Regression tree methods subdivide the input set into partitions of input points i^l using binary splits. Each partition is then given a unique output value. In regression trees, this is typically the average of all output points o^l belonging to the partition. Multiple techniques exist to generate regression trees, e.g., `KD-Tree` [1], `CART` [3], `Totally Randomized Trees` [10], or `Extra-Trees` [10]. The reader is referred to [13] for a detailed description of regression trees. We now state relevant properties and assumptions which we use in the next sections.

Using a regression tree method, a function $\hat{h} : \mathcal{I} \mapsto \mathcal{O}$ fitted to the data set $\mathcal{D}$ can be expressed as [7]: $\hat{h}(i) = \sum_{l=1}^{L} \text{kernel}\left(i^l; i\right) o^l$, for $i \in \mathcal{I}$. The kernels are defined by: $\text{kernel}\left(i^l; i\right) = \frac{\mathbb{I}_{i^l \in \mathcal{P}(i)}}{\sum_{(\hat{\imath},\hat{o}) \in \mathcal{D}} \mathbb{I}_{i^l \in \mathcal{P}(i)}}$, where $\mathbb{I}_x$, the indicator function, returns 1 if x is true and 0 otherwise, and $\mathcal{P}(i)$ returns the tree partition input i is part of. For ensemble methods, the kernels are: $\text{kernel}\left(i^l; i\right) = \frac{1}{e} \sum_{k=1}^{e} \frac{\mathbb{I}_{i^l \in \mathcal{P}_k(i)}}{\sum_{(\hat{\imath},\hat{o}) \in \mathcal{D}} \mathbb{I}_{i^l \in \mathcal{P}_k(i)}}$, where the subscript k refers to the k^{th} regression tree of the ensemble which consists of e trees.

In this work, two assumptions about the regression method we use are made. These assumptions are similar to [7].

Assumption 2. *The kernels and batch data used to fit them are the same in all iterations N of `AMAFQI`.*

Assumption 3. *The kernels are normalized, i.e., $\sum_{l=1}^{L} \text{kernel}\left(i^l; i\right) = 1 \ \forall i \in \mathcal{I}$.*

Moreover, the aforementioned definition of the kernel implies that the sum of the kernel's absolute value is also one when Assumption 3 is satisfied because kernels are nonnegative.

As noted by [7], Assumption 2 is satisfied naturally by a tree method like the `KD-Tree`. If the partitions generated by the tree method are random or depend on the output, this assumption can be met by computing the partitions and thus the kernels only once, i.e., when the first `AMAFQI` iteration is performed. This is the case, for example, for `Totally Randomized Trees` [10] which we use in Section 5. Regression tree approaches satisfy Assumption 3 by construction [7, 20, 21].

3 Approximated multi-agent fitted Q iteration

We now present our multi-agent approximation of `FQI`, `AMAFQI`. The fitting iterations and policy evaluation of `AMAFQI` only depend on the local control space of the agents and do not necessitate computations over the joint control space as would require `FQI`. This allows `AMAFQI` to be a tractable multi-agent approach for batch reinforcement learning problems because optimizing a fitted Q-function, e.g., in (3), must be done by enumeration due to the use of regression trees. The cardinality of the joint control space increases exponentially with the number of agents and the cardinality of the local control space. For `FQI`, this thus leads to a prohibitively large number of calculations when computing approximated Q-functions and when evaluating the policy in multi-agent settings. In the next subsections, we derive the `AMAFQI` algorithm and propose a greedy policy search for our approach.

3.1 Derivation

First, recall the standard Q-learning [32] update for deterministic settings [15]:

$$Q_N(\mathbf{x}, \mathbf{u}) = \begin{cases} Q_{N-1}(\mathbf{x}, \mathbf{u}), & \text{if } \mathbf{x} \neq \mathbf{x}_N \text{ or } \mathbf{u} \neq \mathbf{u}_N \\ r(\mathbf{x}, \mathbf{u}) + \beta \max_{\mathbf{u}' \in \mathcal{U}} Q_{N-1}(\delta(\mathbf{x}, \mathbf{u}), \mathbf{u}'), & \text{if } \mathbf{x} = \mathbf{x}_N \text{ and } \mathbf{u} = \mathbf{u}_N, \end{cases} \tag{4}$$

with $Q_0(\mathbf{x},\mathbf{u}) = 0$ for all $(\mathbf{x},\mathbf{u}) \in \mathcal{X}\times\mathcal{U}$. We remark that in the deterministic setting, the reward r is not a function of the disturbance $\mathbf{w}$. Second, consider for all agent $j = 1,2,\ldots,m$, the distributed Q-learning update for deterministic settings [15]:

$$q_N^j(\mathbf{x},a) = \begin{cases} q_{N-1}^j(\mathbf{x},a), & \text{if } \mathbf{x}\neq\mathbf{x}_N \text{ or } a\neq\mathbf{u}_N(j) \\ \max\Big\{q_{N-1}^j(\mathbf{x},a), r(\mathbf{x},\mathbf{u}) & \\ \qquad +\beta\max_{a'\in\mathcal{A}^j} q_{N-1}^j(\delta(\mathbf{x},\mathbf{u}),a')\Big\}, & \\ \qquad\qquad \text{if } \mathbf{x}=\mathbf{x}_N, \mathbf{u}=\mathbf{u}_N, \text{ and } a=\mathbf{u}_N(j), \end{cases} \tag{5}$$

with $q_0^j(\mathbf{x},a) = 0$ for all $(\mathbf{x},a)\in\mathcal{X}\times\mathcal{A}$. We refer to q_N^j as local q-functions. The proposition below establishes a relation between the centralized and distributed updates.

Proposition 1. *[15, Proposition 1] Let $(\mathbf{x},a)\in\mathcal{X}\times\mathcal{A}$ and suppose that $r(\mathbf{x},\mathbf{u})\geq 0$ for all $(\mathbf{x},\mathbf{u})\in\mathcal{X}\times\mathcal{U}$. Then, for a deterministic, fully cooperative problem, we have*

$$q_N^j(\mathbf{x},a) = \max_{\substack{\mathbf{u}\in\mathcal{U}\\ \mathbf{u}(j)=a}} Q_N(\mathbf{x},\mathbf{u}),$$

for all $j = 1,2,\ldots,m$ and $N\in\mathbb{N}$, where Q_N and q_N^j are computed using (4) and (5), respectively.

Let $N\in\mathbb{N}$ and $j\in\{1,2,\ldots,m\}$. Consider the sample point $(\mathbf{x}^l,\mathbf{u}^l,\mathbf{x}_+^l,r^l)\in\mathcal{S}_L$. For now, let's assume that the function $q_{N-1}^j(\mathbf{x},a)$ is known. We define

$$\begin{aligned} o_N^{l,j} &= q_N^j\left(\mathbf{x}^l,\mathbf{u}^l(j)\right) \\ &= \max\left\{q_{N-1}^j\left(\mathbf{x}^l,\mathbf{u}^l(j)\right), r^l+\beta\max_{a'\in\mathcal{A}^j} q_{N-1}^j\left(\mathbf{x}_+^l,a'\right)\right\}, \end{aligned}$$

where $\mathbf{u}^l(j)$ is the j^{th} component of the joint control $\mathbf{u}^l$, i.e., the control implemented by agent j. Proposition 1 leads to

$$o_N^{l,j} = q_N^j\left(\mathbf{x}^l,\mathbf{u}^l(j)\right) = \max_{\substack{\mathbf{u}\in\mathcal{U}\\ \mathbf{u}(j)=a}} Q_N\left(\mathbf{x}^l,\mathbf{u}\right),$$

where Q_N is computed via (4).

We now depart from prior multi-agent reinforcement learning approaches to derive `AMAFQI`. We apply the reasoning behind `FQI` [7] to compute an approximation $\hat{q}^j$ of the local q^j-function. This is done iteratively. First, we compute the q^j-function values at each batch data point using (5). Second, we fit the approximation function $\hat{q}_N^j(\mathbf{x},a)$ to the set $\left\{\left(\left(\mathbf{x}^l,\mathbf{u}^l(j)\right),\hat{q}_N^j\left(\mathbf{x}^l,\mathbf{u}^l(j)\right)\right), l=1,2,\ldots,L\right\}$ using a regression tree method. Specifically, at iteration $N\in\mathbb{N}$ and for all samples $l=1,2,\ldots,L$, let,

$$\begin{aligned} i^{l,j} &= \left(\mathbf{x}^l,\mathbf{u}^l(j)\right) \\ o_N^{l,j} &= \max\left\{\hat{q}_{N-1}^j\left(\mathbf{x}^l,\mathbf{u}^l(j)\right), r^l+\beta\max_{a'\in\mathcal{A}^j}\hat{q}_{N-1}^j\left(\mathbf{x}_+^l,a'\right)\right\}, \end{aligned}$$

where $\hat{q}_0^j(\mathbf{x},a) = 0$ for all $(\mathbf{x},a)\in\mathcal{X}\times\mathcal{A}$. Then, we compute

$$\hat{q}_N^j(\mathbf{x},a) = \text{RegressionTree}\left(\left\{\left(i^{l,j},o_N^{l,j}\right), l=1,2,\ldots,L\right\};(\mathbf{x},a)\right) \tag{6}$$

Equivalently, we can express (6) as

$$\hat{q}_N^j(\mathbf{x},a) = \sum_{l=1}^{L}\text{kernel}\left(\left(\mathbf{x}^l,\mathbf{u}^l(j)\right);(\mathbf{x},a)\right)o_N^{l,j}, \tag{7}$$

for all $j=1,2,\ldots,m$. The `FQI`-based approach is used to generalize the information obtained from the batch data to all state-control pairs [7]. The regression step estimates values of the local $\hat{q}^j$-function

and thus approximates the maximum of the Q-function for pairs not found in the batch data. From the above discussion, we have that

$$\hat{q}_N^j(\mathbf{x}, a) \approx \max_{\substack{\mathbf{u}\in\mathcal{U} \\ \mathbf{u}(j)=a}} Q_N(\mathbf{x}, \mathbf{u}), \tag{8}$$

In other words, $\hat{q}_N^j(\mathbf{x}, a)$ can be interpreted as the maximum of the learned, centralized Q-function as approximated by agent j when they implement control a. Let $\hat{Q}_N^j$ be the approximation of the Q-function for agent j after N iterations given the available batch data. We can redefine $\hat{q}_N^j(\mathbf{x}, a)$ in terms of the centralized Q-function approximation, $\hat{Q}_N^j$, as:

$$\hat{q}_N^j(\mathbf{x}, a) = \max_{\substack{\mathbf{u}\in\mathcal{U} \\ \mathbf{u}(j)=a}} \hat{Q}_N^j(\mathbf{x}, \mathbf{u}). \tag{9}$$

The right-hand side of (9) is similar to (8)'s and, hence, approximates the maximum of the centralized Q-function by the `FQI` approach [7]. We assume that $\hat{Q}_N^j$ be a monotonically increasing. This assumption is justified by the fact that the Q-function, the $\hat{q}^j$-function, and the `FQI` approximation of the Q-function are all monotonic. The monotonicity follows in all three cases from the structure of the updates when $r(\mathbf{x}, \mathbf{u}) \geq 0$ for all $(\mathbf{x}, \mathbf{u}) \in \mathcal{X} \times \mathcal{U}$ (see Lemma 1). Thus, we assume that an approximation $\hat{Q}_N^j$ of the centralized Q-function from each agent should share this property.

Next, we extend the aforementioned approach to the stochastic setting. Let $j \in \{1, 2, \ldots, m\}$ and $N \in \mathbb{N}$. The stochastic analog of (5) [15] is:

$$q_N^j(\mathbf{x}, a) = \begin{cases} q_{N-1}^j(\mathbf{x}, a), & \text{if } \mathbf{x} \neq \mathbf{x}_N \text{ or } a \neq \mathbf{u}_N(j) \\ \max\Big\{q_{N-1}^j(\mathbf{x}, a), \mathbb{E}\left[r(\mathbf{x}, \mathbf{u}, \mathbf{w})\right. & \\ \left. +\beta \max_{a'\in\mathcal{A}^j} q_{N-1}^j(f(\mathbf{x}, \mathbf{u}, \mathbf{w}), a')\right]\Big\}, & \\ \qquad\qquad \text{if } \mathbf{x} = \mathbf{x}_N, \mathbf{u} = \mathbf{u}_N, \text{ and } a = \mathbf{u}_N(j). \end{cases} \tag{10}$$

The approximation of the local q_N^j-functions for stochastic problems are evaluated as follows. For all $N \in \mathbb{N}$ and $l = 1, 2, \ldots, L$, let

$$\begin{aligned} i^{l,j} &= \left(\mathbf{x}^l, \mathbf{u}^l\right) \\ o_N^{l,j} &= r^l + \beta \max_{a'\in\mathcal{A}^j} \hat{q}_{N-1}^j\left(\mathbf{x}_+^l, a'\right), \end{aligned}$$

where $\hat{q}_0^j(\mathbf{x}, a) = 0$ for all $(\mathbf{x}, a) \in \mathcal{X} \times \mathcal{A}$. We remark that (10), in comparison to the deterministic update given in (5), requires the evaluation of an expectation when the local q^j-function is updated. Hence, the pairs $i^{l,j}$ and $o_N^{l,j}$ cannot be fitted directly as it was done for the deterministic setting. Similarly to [7], a regression tree can be used to estimate an expectation. In our case, we apply a regression tree method over the set of joint states and actions to approximate the expectation from (10)'s second line. We refer to this expectation approximation as the local auxiliary q_N^j-functions, $\tilde{q}_N^j$, which we express as:

$$\tilde{q}_N^j(\mathbf{x}, \mathbf{u}) = \sum_{l=1}^{L} \overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right) o_N^{l,j}, \tag{11}$$

where $\overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right)$, $l = 1, 2, \ldots, L$ are computed using a regression tree over the *joint* control set $\mathcal{U}$. Similarly to [7], a regression tree is used to estimate a conditional expectation. In our case, we approximate the argument of (10)'s maximum, which we denote $\tilde{q}_N^j(\mathbf{x}, \mathbf{u})$. This is motivated by the fact that a regression tree averages the value of the outputs, viz., $r(\mathbf{x}, \mathbf{u}, \mathbf{w}) + \beta \max_{a'\in\mathcal{A}^j} q_{N-1}^j(f(\mathbf{x}, \mathbf{u}, \mathbf{w}), a')$, corresponding to the inputs in a given leaf node or partition, viz., the state-control pairs from the batch data. Alternatively, a linear regression or the Robbins-Monro approximation can be used to estimate the conditional expectation [7, 15].

Finally, the approximation of the local q_N^j-function at $(\mathbf{x}, a) \in \mathcal{X} \times \mathcal{A}$ is given by:

$$\hat{q}_N^j(\mathbf{x}, a) = \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right) \max\left\{\hat{q}_{N-1}^j\left(\mathbf{x}^l, \mathbf{u}^l(j)\right), \tilde{q}_N^j\left(\mathbf{x}^l, \mathbf{u}^l\right)\right\}, \tag{12}$$

where this time, $\text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right)$, $l = 1, 2, \ldots, L$ are computed using a regression tree over the *local* control space $\mathcal{A}^j$. We remark that while $\tilde{q}_N^j(\mathbf{x}, \mathbf{u})$ is a function of the joint control space, we do need to evaluate its maximum over the joint control space and, therefore, $\tilde{q}_N^j(\mathbf{x}, \mathbf{u})$ leads to no scalability issue. Finally, we compute $\hat{q}_N^j$, $N = 1, 2, \ldots$, iteratively until a $\left\|\hat{q}_N^j - \hat{q}_{N-1}^j\right\|_\infty < \epsilon$, for some set tolerance $\epsilon > 0$. A detailed representation of `AMAFQI` is provided in Algorithm 1. Lastly, establishing an exact relation between the $\hat{q}^j$-functions and the centralized Q-function as computed by `FQI` from Section 2.1 is a topic for future work.

Algorithm 1 Approximated multi-agent fitted Q iteration (`AMAFQI`)

Parameters: L, $\mathcal{S}_L$, $\beta \in [0, 1)$, $\epsilon > 0$
Initialization: $N = 0$, $\hat{q}_0^j(\mathbf{x}, a) = 0$ for all $j, \mathbf{x}, a$.

1: Compute kernel $\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, \mathbf{u}(j))\right)$ and $\overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right)$ for all l and j using a regression tree algorithm.
2: **while** $\left\|\hat{q}_N^j - \hat{q}_{N-1}^j\right\|_\infty \geq \epsilon$ **do**
3: $N = N + 1$
4: **for** $j = 1, 2, \ldots, m$ **do**
5: **for** $l = 1, 2, \ldots, L$ **do**
6: Generate the fitting pairs:

$$i^{l,j} = \left(\mathbf{x}^l, \mathbf{u}^l(j)\right)$$
$$o_N^{l,j} = r^l + \beta \max_{a' \in \mathcal{A}} \hat{q}_{N-1}^j\left(\mathbf{x}_+^l, a'\right).$$

7: **end for**
8: **end for**
9: **for** $j = 1, 2, \ldots, m$ **do**
10: Compute the auxiliary $\tilde{q}_N^j$-function:

$$\tilde{q}_N^j(\mathbf{x}, \mathbf{u}) = \sum_{l=1}^{L} \overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right) o_N^{l,j}.$$

11: Update the $\hat{q}_N^j$-function:

$$\hat{q}_N^j(\mathbf{x}, a) = \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right) \max\left\{\hat{q}_{N-1}^j\left(\mathbf{x}^l, \mathbf{u}^l(j)\right), \tilde{q}_N^j\left(\mathbf{x}^l, \mathbf{u}^l\right)\right\}.$$

12: **end for**
13: **end while**

3.2 Greedy policy search

Next, we propose a policy search for `AMAFQI`. We note that there are no guarantees that locally maximizing $\hat{q}^j$-functions leads to a joint optimal policy, e.g., if there are many joint optimal controls for a given state, maximizing $\hat{q}^j$ across the agents $j = 1, 2, \ldots, m$ can lead to local controls belonging to different joint optima, thus, resulting in a suboptimal joint control [15]. For this reason, our policy search sequentially identifies controls that yield an increase in $\hat{q}^j$'s maximum. The policy search is presented in Algorithm 2 and is shown to be a greedy policy in Theorem 1. The search can be extended to decentralized settings using a coordination mechanism [2, 4, 31]. Specifically, the decision set is ordered and tie breaking within the regression (Algorithm 1, Line 1) or classification (Algorithm 2, Line 9) trees is done according to this ordering. The batch data is also ordered and made available to all agents. Using this convention, each agent computes the $\hat{q}^j$-function for all j and uses Algorithm 2 to compute a unique greedy policy. Because the policy is unique, the local control implemented by agent j leads to a joint greedy control.

Algorithm 2 Policy search for `AMAFQI`

Parameters: L, $\mathcal{S}_L$, $\beta \in [0,1)$, $0 < \epsilon \leq \gamma$, $\mathcal{L}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, and $p \in \mathbb{R}$.
Initialization: $N = 0$, $\boldsymbol{\pi}_0(\mathbf{x}^l) = p\mathbf{1}$ for all l.

1: **for** all iteration N **do**
2: **for** $\mathbf{x} \in \mathcal{X}$ **do**
3: **for** l in $\mathcal{L}(\mathbf{x})$ **do**
4: Update policy $\boldsymbol{\pi}_N(\mathbf{x})$ according to (13).
5: **end for**
6: **end for**
7: **end for**
8: **if** $\boldsymbol{\pi}(\mathbf{x}) = p\mathbf{1}$ for $\mathbf{x} \in \mathcal{X}$ **then**
9: Generalize the greedy policy:

$$\hat{\boldsymbol{\pi}}_N(\mathbf{x}) = \text{ClassificationTree}\left(\left\{\left(\mathbf{x}^l, \boldsymbol{\pi}(\mathbf{x}^l)\right), l = 1,2,\ldots,L \,|\, \boldsymbol{\pi}_N(\mathbf{x}^l) \neq p\mathbf{1}\right\}, \mathbf{x}\right)$$

10: **end if**

Let $j \in \{1,2\ldots,m\}$, $l \in \{1,2,\ldots,L\}$. Let $0 < \epsilon \leq \gamma < +\infty$. The parameter γ governs how the policy is iteratively updated, and is related to ϵ, the maximum difference between two consecutive $\hat{q}_N^j$ values at convergence (see Algorithm 1, line 2). Parameter γ can be equal to but no smaller than ϵ. Choosing γ larger than ϵ may enable the identification of suboptimal actions in cases where smaller values of γ lead to an inconclusive policy search. In this sense, by enabling a relaxation of the stringency of the policy, this parameter provides practical value, however we leave the exploration of its theoretical properties to future work. Let $\mathcal{L}(\mathbf{x}) = \left\{l = 1,2,\ldots,L \,|\, \mathbf{x} = \mathbf{x}^l, \left(\mathbf{x}^l, \mathbf{u}^l, \mathbf{x}_+^l, r^l\right) \in \mathcal{S}_L\right\}$ for all $\mathbf{x} \in \mathcal{X}$. The set $\mathcal{L}(\mathbf{x})$ identifies sample points l such data $\mathbf{x}^l = \mathbf{x}$. Let $N \in \mathbb{N}$ where $N \geq 1$. Consider the policy $\boldsymbol{\pi}_N : \mathcal{X} \mapsto \mathcal{U}$ evaluated at a point from the batch data provided in (13) with $\boldsymbol{\pi}_0(\mathbf{x}) = p\mathbf{1}$ for all $\mathbf{x} \in \mathcal{X}$.

$$\boldsymbol{\pi}_N(\mathbf{x}) = \begin{cases} \mathbf{u}^l, & \text{if } \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a) - \max_{a \in \mathcal{A}^j} \hat{q}_{N-1}^j(\mathbf{x}, a) \geq \gamma \ \forall j \in \{1,2,\ldots,m\} \\ & \quad \text{and } \hat{q}_N^j\left(\mathbf{x}, \mathbf{u}^l(j)\right) = \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a) \ \forall j \in \{1,2,\ldots,m\}, \text{ s.t. } l \in \mathcal{L}(\mathbf{x}) \\ p\mathbf{1}, & \text{if } \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a) - \max_{a \in \mathcal{A}^j} \hat{q}_{N-1}^j(\mathbf{x}, a) \geq \gamma \ \forall j \in \{1,2,\ldots,m\} \\ & \quad \text{and } \hat{q}_N^j\left(\mathbf{x}, \mathbf{u}^l(j)\right) \neq \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a) \text{ for } j \in \{1,2,\ldots,m\}, \text{ s.t. } l \in \mathcal{L}(\mathbf{x}) \\ \boldsymbol{\pi}_{N-1}(\mathbf{x}), & \text{otherwise.} \end{cases} \tag{13}$$

In (13), $\mathbf{1}$ is an m-dimensional vector consisting only of ones and p is an auxiliary parameter used to indicate that no control within the data set corresponds to the greedy maximum for state $\mathbf{x}$ after the N^{th} `AMAFQI` iteration. It is used to restart the search. If $\boldsymbol{\pi}_N(\mathbf{x}) = p\mathbf{1}$ when the search ends, then the policy for state $\mathbf{x}$ must be approximated from similar states $\mathbf{x}'$ for which a greedy decision has been identified, i.e., $\boldsymbol{\pi}_N(\mathbf{x}') \neq p\mathbf{1}$. This will be discussed at the end of this section. We now have the following results about the policy (13).

Theorem 1. *Let $l \in \{1,2,\ldots,L\}$ such that $\boldsymbol{\pi}_N\left(\mathbf{x}^l\right) \neq p\mathbf{1}$ and $\overline{\mathbf{u}} \in \boldsymbol{\pi}_N\left(\mathbf{x}^l\right)$. Then, for all $j = \{1,2,\ldots,m\}$, we have:*

$$\max_{\mathbf{u} \in \mathcal{U}} \hat{Q}_N^j\left(\mathbf{x}^l, \mathbf{u}\right) - \hat{Q}_N^j\left(\mathbf{x}^l, \overline{\mathbf{u}}\right) < 2\gamma,$$

and $\boldsymbol{\pi}_N\left(\mathbf{x}^l\right)$ is a 2γ-greedy policy at $\mathbf{x}^l$ with respect to all $\hat{Q}_N^j$, the monotonic approximations of the centralized Q-function from each agent.

The proof of Theorem 1 is presented in A. The above policy search identifies controls using $\hat{q}_N^j$-values that are within 2γ of the $\hat{Q}_N^j$'s maximum for states $\mathbf{x}$ that belongs to the batch data. The search is inconclusive if the optimal control with respect to $\hat{Q}_N^j$ at state $\mathbf{x} \in \mathcal{X}$ for some agent j is not in the batch data or if the optimal control performed poorly when sampled to generate the batch data because of stochasticity.

If $\boldsymbol{\pi}(\mathbf{x}) \neq p\mathbf{1}$ for all $\mathbf{x} \in \mathcal{X}$, then the policy can be used directly. If $\boldsymbol{\pi}(\mathbf{x}) = p\mathbf{1}$ for some $\mathbf{x} \in \mathcal{X}$, then we use an approximation to generalize the policy to all states similarly to the approach used to

generalize the $\hat{q}$-value to all state-control pairs. Let $\hat{\boldsymbol{\pi}}_N : \mathcal{X} \mapsto \mathcal{U}$ be the approximation of the greedy policy with respect to all $\hat{Q}_N^j$, $j = 1, 2, \ldots, m$:

$$\hat{\boldsymbol{\pi}}_N(\mathbf{x}) = \text{ClassificationTree}\left(\left\{\left(\mathbf{x}^l, \boldsymbol{\pi}(\mathbf{x}^l)\right), l = 1, 2, \ldots, L \middle| \boldsymbol{\pi}_N(\mathbf{x}^l) \neq p\mathbf{1}\right\}, \mathbf{x}\right) \tag{14}$$

Finally, if $\boldsymbol{\pi}(\mathbf{x}) = p\mathbf{1}$ for all $\mathbf{x} \in \mathcal{X}$, the batch data does not permit to identify a 2γ-greedy policy with respect to all $\hat{Q}_N^j$-functions. We remark that $\hat{\boldsymbol{\pi}}_N$ only needs to be computed once when the `AMAFQI` has converged to the $\hat{q}^j$ functions. Thus, a significant advantage of `AMAFQI`'s policy is that once the `AMAFQI` algorithm has converged, little to no computations are required to determine the controls when the policy is used. In comparison, the maximum over the joint control space $\mathcal{U}$ of the approximated Q-function needs to be computed when `FQI` is implemented. This must be done by enumeration because the maximization problem is neither analytically nor numerically solvable. In a multi-agent setting, the cardinality of the joint control space increases exponentially with the number of agents. Thus, removing the need to compute this maximum further reduces the computational burden of `FQI` when `AMAFQI` is used.

3.3 `AMAFQI-L` update

In the previous subsection, we presented a 2γ-greedy policy search with respect to the approximations of the centralized Q-function of all agents j. This policy search can be modified to only use the $\hat{q}_N^j$-function of a single agent j. We refer to this alternate policy as `AMAFQI-L`. Because of (8), the maximum of a single $\hat{q}_N^j$ still approximates the centralized Q-function's maximum. The difference is that `AMAFQI-L` is now a 2γ-greedy policy with respect to agent j's approximation of the centralized Q-function rather than with respect to the approximation of all agents. Thus, this approximation is looser than the previous one. The main gain is, however, computational efficiency because only a single $\hat{q}^j$-function must be iteratively computed. The computational requirement is thus constant with respect to the number of agents whereas it scales linearly and exponentially with the number of agents for `AMAFQI` and `FQI`, respectively.

`AMAFQI-L`'s algorithm is similar to `AMAFQI`, except that j is set to a constant value within $\{1, 2, \ldots, m\}$ throughout the iterations N and the policy search. The algorithm is presented in Algorithm 3 of B. The `AMAFQI-L` policy search is identical to `AMAFQI`'s, but uses (15) instead of (13):

$$\boldsymbol{\pi}_N^{\mathrm{L}}(\mathbf{x}) = \begin{cases} \mathbf{u}^l, & \text{if } \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a) - \max_{a \in \mathcal{A}^j} \hat{q}_{N-1}^j(\mathbf{x}, a) \geq \gamma \\ & \text{and } \hat{q}_N^j\left(\mathbf{x}, \mathbf{u}^l(j)\right) = \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a), \text{ s.t. } l \in \mathcal{L}(\mathbf{x}) \\ p\mathbf{1}, & \text{if } \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a) - \max_{a \in \mathcal{A}^j} \hat{q}_{N-1}^j(\mathbf{x}, a) \geq \gamma \\ & \text{and } \hat{q}_N^j\left(\mathbf{x}, \mathbf{u}^l(j)\right) \neq \max_{a \in \mathcal{A}^j} \hat{q}_N^j(\mathbf{x}, a), \text{ s.t. } l \in \mathcal{L}(\mathbf{x}) \\ \boldsymbol{\pi}_{N-1}(\mathbf{x}), & \text{otherwise.} \end{cases} \tag{15}$$

The above discussion is formalized by the following result.

Corollary 1. *Consider Theorem 1's assumptions, and suppose $j \in \{1, 2, \ldots, m\}$. If $\overline{\mathbf{u}} \in \boldsymbol{\pi}_N^L(\mathbf{x}^l)$, then we obtain: $\max_{\mathbf{u} \in \mathcal{U}} \hat{Q}_N^j(\mathbf{x}^l, \mathbf{u}) - \hat{Q}_N^j(\mathbf{x}^l, \overline{\mathbf{u}}) < 2\gamma$, and $\boldsymbol{\pi}_N^L(\mathbf{x}^l)$ is a 2γ-greedy policy at $\mathbf{x}^l$ with respect to $\hat{Q}_N^j$.*

The proof follows from Theorem 1 for a single j.

4 Convergence

We show that each local $\hat{q}_N^j$-function defined in (12) converges to a unique and finite function with respect to the infinity norm. We first establish the monotonicity of $\hat{q}_N^j$ for all j.

Lemma 1. *Suppose $r(\mathbf{x}, \mathbf{u}, \mathbf{w}) \geq 0$ and $\hat{q}_0^j(\mathbf{x}, a) = 0$ for all $(\mathbf{x}, a, \mathbf{w}) \in \mathcal{X} \times \mathcal{A} \times \mathcal{W}$, then $\hat{q}_N^j(\mathbf{x}, a) \leq \hat{q}_{N+1}^j(\mathbf{x}, a)$ for all $(\mathbf{x}, a) \in \mathcal{X} \times \mathcal{A}$ and $N \in \mathbb{N}$.*

We now state the convergence result.

Theorem 2. *Suppose Assumptions 1–3 hold and* $\hat{q}_0^j(\mathbf{x}, a) = 0$ *for all* $(\mathbf{x}, a, \mathbf{w}) \in \mathcal{X} \times \mathcal{A} \times \mathcal{W}$ *and* $j = 1, 2, \ldots, m$. *Then* $\hat{q}_N^j(\mathbf{x}, a)$ *converges to the unique limit* $\hat{q}_{\mathcal{S}_L}^j(\mathbf{x}, a)$, *i.e., the unique maximum of the* $\hat{Q}^j$*-function for* $\mathbf{x}$ *and* $\mathbf{u}(j) = a$ *when estimated using the data set* $\mathcal{S}_L$. *Moreover, for all* $\epsilon > 0$, *there exists* $n(j) \in \mathbb{N}$ *such that for all* $N \geq n(j)$,

$$\left\| \hat{q}_N^j - \hat{q}_{\mathcal{S}_L}^j \right\|_\infty < \epsilon.$$

The proofs of Lemma 1 and Theorem 2 are in C and D, respectively. Theorem 2 ensures that there exist unique, finite-valued $\hat{q}^j$-functions for the data set $\mathcal{S}_L$ which can be used for the policy search. Thus, $\hat{q}_{\mathcal{S}_L}^j$-functions for the data set $\mathcal{S}_L$ can always be computed under the aforementioned assumptions. We remark that Theorem 2 applies to `AMAFQI` and `AMAFQI-L` because it holds for any j.

Similarly to [7], the error due to the regression-tree method (or any other supervised learning approach) is not modeled explicitly in this work. For `AMAFQI`, this error would translate in $\hat{Q}^j$-functions suffering itself from a larger approximation error. We remark that using a regression tree method allows us to establish the `AMAFQI`'s convergence. The regression error is a topic for future investigation.

5 Numerical examples

In this section, we compare the performance of `AMAFQI` and `FQI` in numerical simulations. Our comparison uses `FQI` because it provides a learned Q-function that is the unique solution to Bellman's equation given the batch data [7]. It can, therefore, be considered as an adequate benchmark for the batch reinforcement learning setting. We test our approach on a multi-agent, multi-state random problem similar to the example presented in [6, 34].

Let $\hat{Q}_N^{\texttt{FQI}} : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}$ be the approximated Q-function after N iterations evaluated via `FQI` [7]. Single problem instance simulations are run on a 2.4 GHz Intel Core i5 laptop computer and multiple instance simulations are run on the Savio computational cluster resource from the Berkeley Research Computing program. The computations of $\hat{q}_N^j$ and $\hat{Q}_N^{\texttt{FQI}}$ for all samples l are parallelized to reduce the full computation time.

5.1 Setting

The multi-agent, multi-state random problem is as follows. We consider m agent having to coordinate their individual binary decision to reach one of the X joint states and maximize their reward over τ rounds. The joint binary decision determines the probability of moving from one state to another. Let $P(\mathbf{x}) : \mathcal{U} \times \mathcal{X} \mapsto \mathcal{X}$ be the transition matrix for state $\mathbf{x} \in \mathcal{X}$. All transition matrices are randomly generated according to uniform distributions and then normalized to obtain row-stochastic matrices. The reward is determined by the joint state at the end of a round. Let the mean reward for a state $\mathbf{x} \in \mathcal{X}$ be $R(\mathbf{x}) \sim$ Uniform[0,5]. The reward for reaching state $\mathbf{x} \in \mathcal{X}$ is then $r(\mathbf{x}) \sim$ Uniform$[R(\mathbf{x}) - \frac{1}{2}, R(\mathbf{x}) + \frac{1}{2}]$.

5.2 Experiments

We use `Totally Randomized Trees` [10] for the regression tree. We consider ensembles of 5 trees with each at a minimum of 10 data points in a leaf node. We let $\beta = 0.5$.

5.2.1 5 agents

We let $m = 5$ and card $\mathcal{X} = 5$. We uniformly sample $L = 2000$ $(\mathbf{x}^l, \mathbf{u}^l, \mathbf{x}_+^l, r^l)$-tuples. The convergence of both `AMAFQI` and `FQI` implementations for this numerical experiment is shown in Figure 1. Figure 1

shows that $\left\|\hat{q}_N^j - \hat{q}_{N-1}^j\right\|_\infty$ and $\left\|\hat{Q}_N^{\texttt{FQI}} - \hat{Q}_{N-1}^{\texttt{FQI}}\right\|_\infty$ go to zero as N increases. Thus, both values converge to their respective unique and finite limits.

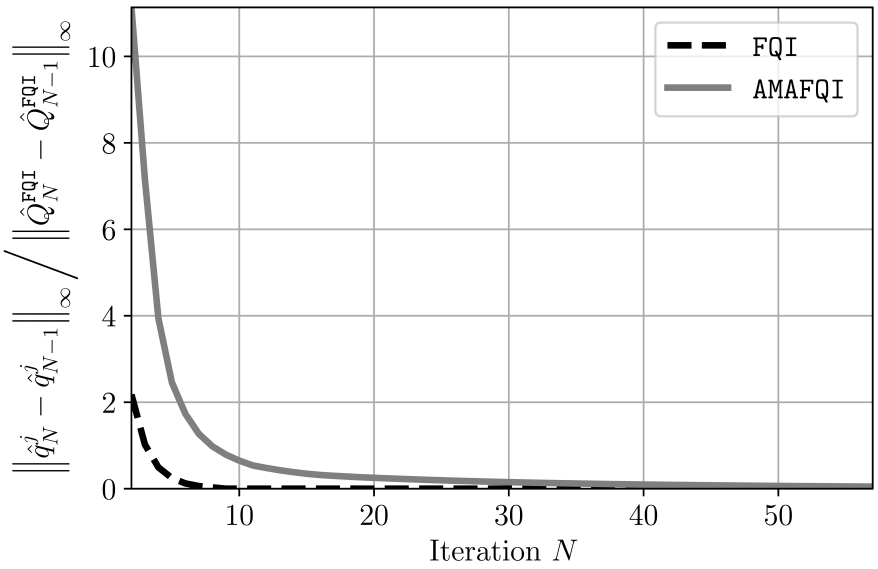

Figure 1: Convergence of `AMAFQI` and `FQI` in the 5-player, 5-state problem

We compare the approximated value function at $\mathbf{x}$ for `AMAFQI` and `FQI` using the relative absolute difference between both maxima, defined as $\Delta(j,\mathbf{x}) = \left|\frac{\max_{a\in\mathcal{A}} \hat{q}_N^j(\mathbf{x},a) - \max_{\mathbf{u}\in\mathcal{U}} \hat{Q}_N^{\texttt{FQI}}(\mathbf{x},\mathbf{u})}{\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}_N^{\texttt{FQI}}(\mathbf{x},\mathbf{u})}\right|$, for $j = 1, 2, \ldots, m$ and $\mathbf{x} \in \mathcal{X}$.

We sequentially compute the $\hat{q}^j$- and $\hat{Q}^{\texttt{FQI}}$-functions for 150 different problem instances, each time sampling a new data set $\mathcal{S}_L$. The average $\Delta(j,\mathbf{x})$ for all the problem instances are reported in Figure 2. The average over all problem instances of the relative difference $\Delta(j,\mathbf{x})$ is 2.92%.

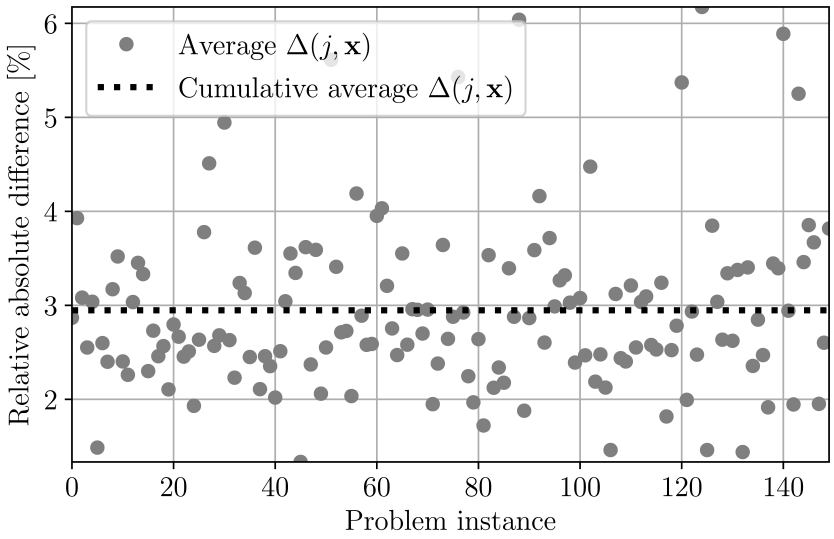

Figure 2: Average $\Delta(j,\mathbf{x})$ over all $j,\mathbf{x}$ for 150 random instances of the 5-agent, 5-state problem

For each problem instance, we compute the reward obtained by the greedy policies over 100 trials each with a time horizon $\tau = 100$ rounds. For each trial, the initial state is randomly sampled. The average reward of `FQI`'s, `AMAFQI`'s, and `AMAFQI-L`'s greedy policies are shown in Figure 3. The relative difference in average cumulative reward between `AMAFQI` and `FQI` is small and only 7.17%. The performance of `AMAFQI-L` is lower than `AMAFQI`'s and leads to a 16.79% cumulative reward decrease in comparison to `FQI`.

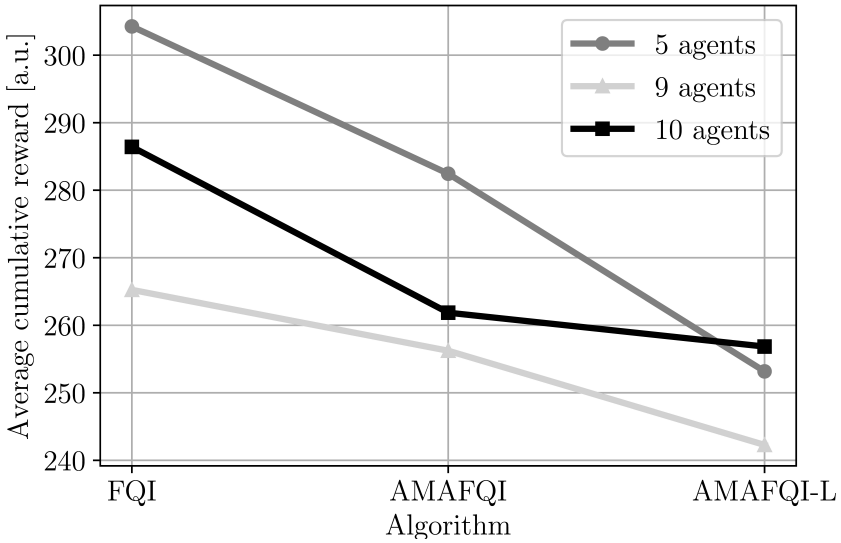

Figure 3: Average cumulative reward for the 5, 9, and 10-agent, 5-state problem over 150, 10, and 5 problem instances, respectively

We conclude by discussing the computation time of `AMAFQI`. The average computation time for a single iteration N and until convergence for `FQI`, `AMAFQI` and `AMAFQI-L` are reported in Table 1 for the 150 problem instances. The numbers from Table 1 given in parentheses and the subsequent similar tables represent the total computation times which includes the policy search. An iteration of `AMAFQI` and `AMAFQI-L` with and without the policy search has a shorter duration than an `FQI` iteration. Because the approximation requires more N iterations, `AMAFQI` still takes more time to converge. The amount of time to convergence for `AMAFQI-L` and `FQI` are similar. The problem size is still small given its binary controls and only 5 agents. Hence, an approach tailored to multi-agent settings is not necessarily needed yet. We provide this example of a small problem instance so that both `AMAFQI` and `FQI` can be simulated repetitively in an acceptable time frame. The comparison's bottleneck is `FQI` which is computationally very time consuming. Thus, given our computing infrastructure, we restrict our analysis to 10 agents or less as our objective is to compare `AMAFQI`'s performances to `FQI`'s on several instances.

Table 1: Average computation times for the 5-agent, 5-state problem (150 problem instances, 100 trials)

Average time	Iteration [s]	Convergence (policy) [s]
`FQI`	23.39	155.00
`AMAFQI`	12.09	577.20 (658.45)
`AMAFQI-L`	2.41	115.44 (135.11)

5.2.2 9 and 10 agents

When the number of agents increases, the computational advantage of `AMAFQI` is clear. Tables 2 and 3 present the computation times for $m = 9$ with $L = 5000$ and $m = 10$ with $L = 7000$, respectively. The average $\Delta(j, \mathbf{x})$ is 8.17% when $m = 9$ and 7.90% when $m = 10$. We note that $\Delta(j, \mathbf{x})$ can be

further reduced by increasing L at the expense of a longer computation time. The averaged cumulative reward for the 100 trials of each problem instance is provided in Figure 3 for both the 9- and 10-agent problem.

As shown in Tables 2 and 3, `AMAFQI` requires much less computation time than `FQI` to converge when m increases and only leads to a limited decrease in cumulative reward. In the present case, we register a 3.40% ($m = 9$) and 8.57% ($m = 10$) reduction of the average reward when using `AMAFQI`. Moreover, for `AMAFQI`, the total computation time until convergence includes most of the calculations required for the evaluation step. `AMAFQI-L` further reduces the total computation time. For $m = 9$, `AMAFQI-L` requires less than 8 minutes to convergence and to compute the policy instead of 84 minutes for `AMAFQI` and 3 hours (177 minutes) for `FQI`. When considering $m = 10$, `AMAFQI-L` needs 14 minutes whereas `AMAFQI` and `FQI` takes, respectively, 3 hours (181 minutes) and 12 hours (723 minutes). The performance of the `AMAFQI-L` policy is slightly lower and leads to a decrease in the cumulative reward of 8.65% ($m = 9$) and 10.32% ($m = 10$) with respect to `FQI`.

Table 2: Average computation times for the 9-agent, 5-state problem (10 problem instances)

Average time	Iteration [s]	Convergence (policy) [s]
`FQI`	1660.07	10615.95
`AMAFQI`	77.31	3766.03 (4998.52)
`AMAFQI-L`	8.59	418.44 (454.23)

Table 3: Average computation times for the 10-agent, 5-state problem (5 problem instances)

Average time	Iteration [s]	Convergence (policy) [s]
`FQI`	6579.77	43421.90
`AMAFQI`	156.58	7859.76 (10840.89)
`AMAFQI-L`	15.67	785.98 (785.98)

6 Conclusion

In this work, we propose the `AMAFQI` algorithm, a tractable multi-agent approximation of `FQI` for batch reinforcement learning problems. We design an iterative policy search for `AMAFQI` and demonstrate that it is a greedy policy with respect to an approximation of the learned Q-function of all agents. Our approach performs computations only over local control sets contrarily to `FQI` that works over the joint control space. The number of calculations required in each iteration of the algorithm grows linearly and exponentially with the number of agents, respectively, for `AMAFQI` and for `FQI`. Consequently, `FQI` is impractical and quickly intractable in presence of multiple agents. Our approach offers an efficient alternative for multi-agent batch reinforcement learning problems. We present a derivative of our approach, `AMAFQI-L`, which further reduces the computational burden of `AMAFQI`.

We consider a multi-agent batch reinforcement learning problem and compare the performance of `AMAFQI` with `FQI`. Numerical simulations show that the value functions computed by our approximation and by `FQI` are similar, e.g., with a discrepancy of 2.92% when $m = 5$, and that the performance level is also alike, e.g., with a difference of 7.12%. Lastly, computation times are compared and `AMAFQI` and `AMAFQI-L` outperform significantly `FQI` when the number of agent increases. For example, `AMAFQI` and `AMAFQI-L` require, respectively, only 181 minutes and 13 minutes against a total computation time of 723 minutes, on average, for `FQI` when $m = 10$. In future work, we wish wish to investigate approaches to reduce the number of N iterations performed in `AMAFQI` before convergence, e.g., by considering the growing batch learning paradigm [14] in which an exploration policy is used, and new observed transitions are periodically incorporated in the batch data. Lastly, we would like to compare `AMAFQI` to `FQI` in a more sophisticated setting and use `AMAFQI` to dispatch flexible loads for network-safe demand response in unknown electric grids. This is a topic for future work.

A Proof of Theorem 1

We base our proof on [15, Proposition 2]. Consider the monotonic approximation of the centralized Q-functions from all agents, $\hat{Q}^j_N$, $j = 1, 2, \ldots, m$. Let $l \in \{1, 2, \ldots, L\}$. Let $0 \leq N' < N$ such that for all $j \in \{1, 2, \ldots, m\}$ we have:

$$\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N'+1}\left(\mathbf{x}^l, \mathbf{u}\right) - \max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N'}\left(\mathbf{x}^l, \mathbf{u}\right) \geq \gamma, \tag{16}$$

and,

$$\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{n}\left(\mathbf{x}^l, \mathbf{u}\right) - \max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N'+1}\left(\mathbf{x}^l, \mathbf{u}\right) < \gamma, \tag{17}$$

for $n = N' + 2, N' + 3, \ldots, N$. From the approximation definition (9), we equivalently have for all $j \in \{1, 2, \ldots, m\}$:

$$\max_{a\in\mathcal{A}^j} \hat{q}^j_{N'+1}\left(\mathbf{x}^l, a\right) - \max_{a\in\mathcal{A}^j} \hat{q}^j_{N'}\left(\mathbf{x}^l, a\right) \geq \gamma, \tag{18}$$

and

$$\max_{a\in\mathcal{A}^j} \hat{q}^j_{n}\left(\mathbf{x}^l, a\right) - \max_{a\in\mathcal{A}^j} \hat{q}^j_{N'+1}\left(\mathbf{x}^l, a\right) < \gamma, \tag{19}$$

for $n = N' + 2, N' + 3, \ldots, N$. By (18) and (19), the last update to the policy at $\mathbf{x}^l$ can only occur at $N' + 1$. Regarding the policy update, if $\hat{q}^j_{N+1}\left(\mathbf{x}, \mathbf{u}^l(j)\right) = \max_{a\in\mathcal{A}^j} \hat{q}^j_{N+1}(\mathbf{x}, a)$ such that $l \in \mathcal{L}(\mathbf{x})$ for all j, then this last update was performed when the control $\mathbf{u}^l$ was considered by the `AMAFQI` update. Otherwise, if there exists no $l \in \mathcal{L}(\mathbf{x})$ such that $\hat{q}^j_{N+1}\left(\mathbf{x}, \mathbf{u}^l(j)\right) = \max_{a\in\mathcal{A}^j} \hat{q}^j_{N+1}(\mathbf{x}, a)$ or the equality does not hold for all j, the search is inconclusive for the iteration N. By assumption, $\boldsymbol{\pi}_{N'+1}(\mathbf{x}^l) \neq p\mathbf{1}$ and at least one policy update was performed.

Finally, iteration $N' + 1$ coincides to the last time the maximum $\hat{Q}^j$-function changed by at least γ for all j because of (16) and (17). Thus, for all $\overline{\mathbf{u}}_{N'+1} \in \boldsymbol{\pi}_{N'+1}(\mathbf{x}^l)$ we have

$$\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N'+1}\left(\mathbf{x}^l, \mathbf{u}\right) - \hat{Q}^j_{N'+1}\left(\mathbf{x}^l, \overline{\mathbf{u}}_{N'+1}\right) < \gamma, \tag{20}$$

for all $j \in \{1, 2, \ldots, m\}$. The monotonicity of the $\hat{Q}^j_N$-function implies that (20) can be re-expressed as

$$\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N'+1}\left(\mathbf{x}^l, \mathbf{u}\right) - \hat{Q}^j_{N}\left(\mathbf{x}^l, \overline{\mathbf{u}}_{N'+1}\right) < \gamma. \tag{21}$$

From (17), we know that

$$\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N}\left(\mathbf{x}^l, \mathbf{u}\right) - \gamma < \max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_{N'+1}\left(\mathbf{x}^l, \mathbf{u}\right). \tag{22}$$

Using (22) in (21), we obtain $\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_N\left(\mathbf{x}^l, \mathbf{u}\right) - \hat{Q}^j_N\left(\mathbf{x}^l, \overline{\mathbf{u}}_{N'+1}\right) < 2\gamma$. Lastly, because the policy is not updated between $N' + 1$ and N, we have that $\boldsymbol{\pi}_{N'+1}\left(\mathbf{x}^l\right) = \boldsymbol{\pi}_N\left(\mathbf{x}^l\right)$ and thus, we have $\max_{\mathbf{u}\in\mathcal{U}} \hat{Q}^j_N\left(\mathbf{x}^l, \mathbf{u}\right) - \hat{Q}^j_N\left(\mathbf{x}^l, \overline{\mathbf{u}}_N\right) < 2\gamma$, where $\overline{\mathbf{u}}_N \in \boldsymbol{\pi}_N\left(\mathbf{x}^l\right)$. Hence, the policy $\boldsymbol{\pi}_N\left(\mathbf{x}^l\right) \neq p\mathbf{1}$ is a 2γ-greedy policy for the approximation of the centralized Q-function of all agents. ■

B AMAFQI-L algorithm

Algorithm 3 Approximated multi-agent fitted Q iteration – Light (AMAFQI-L)

Parameters: L, $\mathcal{S}_L$, $\beta \in [0,1)$, $\epsilon > 0$, $j \in \{1,2,\ldots,m\}$.
Initialization: $N = 0$, $\hat{q}_0^j(\mathbf{x},a) = 0$ for all $\mathbf{x}, a$.

Compute kernel $\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, \mathbf{u}(j))\right)$ and $\overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right)$ for all l using a regression tree algorithm.
while $\left\|\hat{q}_N^j - \hat{q}_{N-1}^j\right\|_\infty \geq \epsilon$ **do**
  $N = N + 1$
  for $l = 1, 2, \ldots, L$ **do**
    Generate the fitting pairs:

$$i^{l,j} = \left(\mathbf{x}^l, \mathbf{u}^l(j)\right)$$
$$o_N^{l,j} = r^l + \beta \max_{a' \in \mathcal{A}} \hat{q}_{N-1}^j\left(\mathbf{x}_+^l, a'\right).$$

  end for
  Compute the auxiliary $\tilde{q}_N^j$-function:

$$\tilde{q}_N^j(\mathbf{x}, \mathbf{u}) = \sum_{l=1}^{L} \overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right) o_N^{l,j}.$$

  Update the $\hat{q}_N^j$-function:

$$\hat{q}_N^j(\mathbf{x}, a) = \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right) \max\left\{\hat{q}_{N-1}^j\left(\mathbf{x}^l, \mathbf{u}^l(j)\right), \tilde{q}_N^j\left(\mathbf{x}^l, \mathbf{u}^l\right)\right\}.$$

end while

C Proof of Lemma 1

We prove this lemma by induction. Let $(\mathbf{x}, a) \in \mathcal{X} \times \mathcal{A}$ and $j \in \{1,2,\ldots,m\}$. For $N = 0$, we have $\hat{q}_0^j(\mathbf{x}, a) = 0$ for all $\mathbf{x}, a$ by assumption. For $N = 1$, we then have:

$$\begin{aligned}\hat{q}_1^j(\mathbf{x}, a) &= \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right) \max\left\{\hat{q}_0^j\left(\mathbf{x}^l, \mathbf{u}^l(j)\right), \tilde{q}_1^j\left(\mathbf{x}^l, \mathbf{u}^l\right)\right\} \\ &= \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right) \max\left\{0, r^l\right\}\end{aligned}$$

because $\sum_{l=1}^{L} \overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right) = 1$ for all $(\mathbf{x}, \mathbf{u}) \in \mathcal{X} \times \mathcal{U}$. By assumption, $r^l \geq 0$ and, therefore, $\hat{q}_0^j(\mathbf{x}, a) \leq \hat{q}_1^j(\mathbf{x}, a)$. We now show that, the induction hypothesis, $\hat{q}_N^j(\mathbf{x}, a) \leq \hat{q}_{N+1}^j(\mathbf{x}, a)$, holds for $N \to N + 1$. At $N + 1$, the $\hat{q}^j$-function is

$$\hat{q}_{N+1}^j(\mathbf{x}, a) = \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l, \mathbf{u}^l(j)\right); (\mathbf{x}, a)\right) \max\left\{\hat{q}_N^j\left(\mathbf{x}^l, \mathbf{u}^l(j)\right), \tilde{q}_{N+1}^j\left(\mathbf{x}^l, \mathbf{u}^l\right)\right\}, \tag{23}$$

where

$$\tilde{q}_{N+1}^j(\mathbf{x}, \mathbf{u}) = \sum_{l=1}^{L} \overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right) \left[r^l + \beta \max_{a' \in \mathcal{A}} \hat{q}_N^j\left(\mathbf{x}_+^l, a'\right)\right] \tag{24}$$

We first use the induction hypothesis in (24) and obtain

$$\begin{aligned}\tilde{q}_{N+1}^j(\mathbf{x}, \mathbf{u}) &\leq \sum_{l=1}^{L} \overline{\text{kernel}}\left(\left(\mathbf{x}^l, \mathbf{u}^l\right); (\mathbf{x}, \mathbf{u})\right) \left[r^l + \beta \max_{a' \in \mathcal{A}} \hat{q}_{N+1}^j\left(\mathbf{x}_+^l, a'\right)\right] \\ &\leq \tilde{q}_{N+2}^j(\mathbf{x}, \mathbf{u})\end{aligned} \tag{25}$$

Second, we use the induction hypothesis and (25) in (23). This leads to

$$\begin{aligned}\hat{q}^j_{N+1}(\mathbf{x},a) &\leq \sum_{l=1}^{L} \text{kernel}\left(\left(\mathbf{x}^l,\mathbf{u}^l(j)\right);(\mathbf{x},a)\right) \max\left\{\hat{q}^j_{N+1}\left(\mathbf{x}^l,\mathbf{u}^l(j)\right),\tilde{q}^j_{N+2}\left(\mathbf{x}^l,\mathbf{u}^l\right)\right\}\\ &= \hat{q}^j_{N+2}(\mathbf{x},a)\end{aligned}$$

where we last used the definition of $\hat{q}^j_{N+2}$. Thus, we have established that $\hat{q}^j_N(\mathbf{x},a)$ is monotonically increasing for all $(\mathbf{x},a)\in\mathcal{X}\times\mathcal{A}$ and all $N\in\mathbb{N}$. ■

D Proof of Theorem 2

We first show that $\hat{q}^j_N$ is bounded. By Assumption 1, we have $r(\mathbf{x},\mathbf{u},\mathbf{w})\leq R$. Let $j\in\{1,2,\ldots,m\}$. By definition, $\hat{q}^j_0(\mathbf{x},a)=0$ for all $(\mathbf{x},\mathbf{u})\in\mathcal{X}\times\mathcal{U}$. For $N=1$, we have

$$\begin{aligned}\left\|\hat{q}^j_1(\mathbf{x},a)\right\|_\infty &\leq \left\|\sum_{l=1}^{L}\text{kernel}\left(\left(\mathbf{x}^l,\mathbf{u}^l(j)\right);(\mathbf{x},a)\right)\max\left\{0,\sum_{l=1}^{L}\overline{\text{kernel}}\left(\left(\mathbf{x}^l,\mathbf{u}^l\right);\left(\mathbf{x}^l,\mathbf{u}^l\right)\right)R\right\}\right\|_\infty\\ &= \max\{0,R\}\end{aligned}$$

because kernels are non-negative and their sum is normalized. By the same process, we sequentially bound $\hat{q}^j_N(\mathbf{x},a)$ for all $N\in\mathbb{N}$:

$$\left\|\hat{q}^j_N(\mathbf{x},a)\right\|_\infty \leq \left\|\sum_{l=1}^{L}\text{kernel}\left(\left(\mathbf{x}^l,\mathbf{u}^l(j)\right);(\mathbf{x},a)\right)\max\left\{\sum_{n=1}^{N-1}\beta^{n-1}R,R+\beta\sum_{n=1}^{N-1}\beta^{n-1}R\right\}\right\|_\infty \tag{26}$$

We further bound (26) and obtain: $\left\|\hat{q}^j_N(\mathbf{x},a)\right\|_\infty \leq \frac{R}{1-\beta}$ for all $N\in\mathbb{N}$. Therefore, $\left\|\hat{q}^j_N(\mathbf{x},a)\right\|_\infty$ is bounded from above for all $j\in\{1,2,\ldots,m\}$, and $N\in\mathbb{N}$. We remark that this is an upper bound and not necessarily the supremum of $\hat{q}^j_N$.

By the monotone convergence theorem, $\hat{q}^j_N(\mathbf{x},a)\to\hat{q}^j(\mathbf{x},a)$, where $\hat{q}^j(\mathbf{x},a)\leq\frac{R}{1-\beta}$ is the supremum of the sequence given in (12) at $(\mathbf{x},a)$ because the sequence is monotonically increasing by Lemma 1 and is bounded from above. A limit is unique if it exists and therefore $\hat{q}^j(\mathbf{x},a)$ is the unique solution of (7) at $(\mathbf{x},a)\in\mathcal{X}\times\mathcal{A}$. It follows from (9) that the limit is the maximum of the centralized Q-function approximation at $\mathbf{x}$ and $\mathbf{u}(j)=a$.

Lastly, for all $\epsilon>0$, there exists $N(\mathbf{x},a)$ such that for all $N\geq N^j(\mathbf{x},a)$ and we can write $\left|\hat{q}^j_N(\mathbf{x},a)-\hat{q}^j(\mathbf{x},a)\right|<\epsilon$. Consequently, for $\epsilon>0$, we have $\left\|\hat{q}^j_N-\hat{q}^j\right\|_\infty<\epsilon$. for all $N\geq n(j)=\max_{\mathbf{x},a}N^j(\mathbf{x},a)$. ■

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9):509–517, 1975.

[2] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[3] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. Classification and regression trees. CRC press, 1984.

[4] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38(2):156–172, 2008.

[5] Duncan S Callaway and Ian A Hiskens. Achieving controllability of electric loads. Proceedings of the IEEE, 99(1):184–199, 2010.

[6] Christoph Dann, Gerhard Neumann, Jan Peters, et al. Policy evaluation with temporal differences: A survey and comparison. Journal of Machine Learning Research, 15:809–883, 2014.

[7] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. Journal of Machine Learning Research, 6(Apr):503–556, 2005.

[8] Thomas Gabel and Martin Riedmiller. Evaluation of batch-mode reinforcement learning methods for solving dec-mdps with changing action sets. In European Workshop on Reinforcement Learning, pages 82–95. Springer, 2008.

[9] Thomas Gabel and Martin A Riedmiller. Reinforcement learning for dec-mdps with changing action sets and partially ordered dependencies. In AAMAS (3), pages 1333–1336, 2008.

[10] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. Machine learning, 63(1):3–42, 2006.

[11] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In International Conference on Autonomous Agents and Multiagent Systems, pages 66–83. Springer, 2017.

[12] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. Autonomous Agents and Multi-Agent Systems, 33(6):750–797, 2019.

[13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning, volume 112. Springer, 2013.

[14] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In Reinforcement learning, pages 45–73. Springer, 2012.

[15] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In In Proceedings of the Seventeenth International Conference on Machine Learning. Citeseer, 2000.

[16] Jae Won Lee, Jonghun Park, O Jangmin, Jongwoo Lee, and Euyseok Hong. A multiagent approach to Q-learning for daily stock trading. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 37(6):864–877, 2007.

[17] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3-4):293–321, 1992.

[18] David G Luenberger. Optimization by vector space methods. John Wiley & Sons, New York, NY, 1997.

[19] Brida V Mbuwir, Frederik Ruelens, Fred Spiessens, and Geert Deconinck. Battery energy management in a microgrid using batch reinforcement learning. Energies, 10(11):1846, 2017.

[20] Dirk Ormoneit and Peter Glynn. Kernel-based reinforcement learning in average-cost problems. IEEE Transactions on Automatic Control, 47(10):1624–1636, 2002.

[21] Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. Machine learning, 49(2-3):161–178, 2002.

[22] Afshin OroojlooyJadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. arXiv preprint arXiv:1908.03963, 2019.

[23] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Aria Nefian. Cooperative and distributed reinforcement learning of drones for field coverage. arXiv preprint arXiv:1803.07250, 2018.

[24] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In European Conference on Machine Learning, pages 317–328. Springer, 2005.

[25] Frederik Ruelens, Bert J Claessens, Stijn Vandael, Bart De Schutter, Robert Babuška, and Ronnie Belmans. Residential demand response of thermostatically controlled loads using batch reinforcement learning. IEEE Transactions on Smart Grid, 8(5):2149–2159, 2016.

[26] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint arXiv:1610.03295, 2016.

[27] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. Technical Report, 2003.

[28] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. Autonomous Robots, 8(3):345–383, 2000.

[29] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.

[30] Stijn Vandael, Bert Claessens, Damien Ernst, Tom Holvoet, and Geert Deconinck. Reinforcement learning of heuristic ev fleet charging in a day-ahead electricity market. IEEE Transactions on Smart Grid, 6(4):1795–1805, 2015.

[31] Nikos Vlassis. A concise introduction to multiagent systems and distributed artificial intelligence. Synthesis Lectures on Artificial Intelligence and Machine Learning, 1(1):1–71, 2007.

[32] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.

[33] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. arXiv preprint arXiv:1911.10635, 2019.

[34] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Başar. Fully decentralized multi-agent reinforcement learning with networked agents. arXiv preprint arXiv:1802.08757, 2018.

[35] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Finite-sample analysis for decentralized batch multi-agent reinforcement learning with networked agents. IEEE Transactions on Automatic Control, 2021.