Q-function Decomposition with Intervention Semantics for Factored Action Spaces

Junkyu Lee[1], **Tian Gao**[1], **Elliot Nelson**[2], **Miao Liu**[1], **Debarun Bhattacharjya**[1], **Songtao Lu**[1]

[1]IBM Research, [2]Independent

{junkyu.lee,songtao}@ibm.com, elliot137@gmail.com, {tgao,debarunb}@us.ibm.com

Abstract

Many practical reinforcement learning environments have a discrete factored action space that induces a large combinatorial set of actions, thereby posing significant challenges. Existing approaches leverage the regular structure of the action space and resort to a linear decomposition of Q-functions, which avoids enumerating all combinations of factored actions. In this paper, we consider Q-functions defined over a lower dimensional projected subspace of the original action space, and study the condition for the unbiasedness of decomposed Q-functions using causal effect estimation from the observed confounder setting in causal statistics. This leads to a general scheme that uses the projected Q-functions to approximate the Q-function in standard model-free reinforcement learning algorithms. The proposed approach is shown to improve sample complexity in a model-based reinforcement learning setting. We demonstrate improvements in sample efficiency compared to state-of-the-art baselines in online continuous control environments and a real-world offline sepsis treatment environment.

Introduction

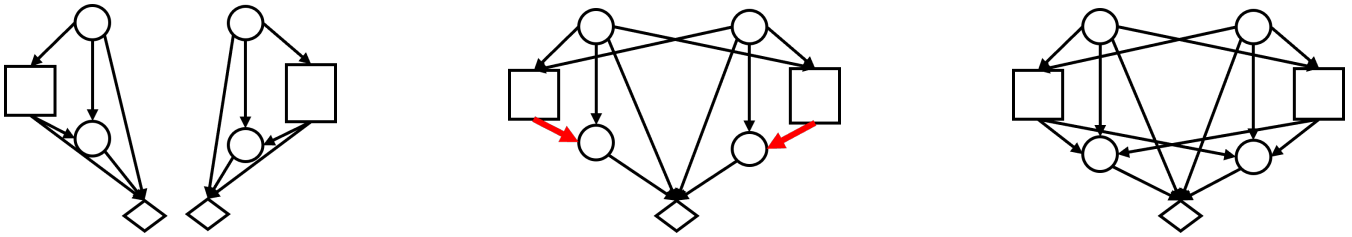
Reinforcement learning (RL) combined with deep learning has advanced to achieve superhuman levels of performance in many application domains (Mnih et al. 2015; Silver et al. 2017), but there is still significant room for improving sample complexity and computational tractability for wider acceptance and deployment in real-world applications (Pearl 2019; Schölkopf 2022). In many practical applications such as healthcare domains, collecting a batch of interaction data in an off-policy manner or even in an offline setting is preferable, although it limits collecting diverse and large amount of samples due to the costly or infeasible nature of interactions (Komorowski et al. 2018; Tang et al. 2022). Even in online environments, a structured combinatorial action space is well known to deteriorate sample efficiency significantly (Dulac-Arnold et al. 2015; Tavakoli, Pardo, and Kormushev 2018; Tavakoli, Fatemi, and Kormushev 2020).

The goal of this paper is to improve the sample efficiency of value-based RL algorithms for solving problems having a large factored action space. The challenges for handling large action spaces are well recognized and typical solutions involve either decomposing the action space (Tang et al. 2022; Rebello et al. 2023) or augmenting data (Pitis, Creager, and Garg 2020; Pitis et al. 2022; Tang and Wiens 2023). In multi-agent RL (Sunehag et al. 2018; Son et al. 2019; Wang et al. 2020; Rashid et al. 2020), there exists a clear decomposable structure that allows representing the global value function as a combination of local value function per each agent. However, in single-agent RL, existing approaches are motivated by the assumption that the given MDP can be separated into independent MDPs leading to a linear Q-function decomposition (Russell and Zimdars 2003; Tang et al. 2022; Seyde et al. 2022).

In factored action spaces, each action $\mathbf{a}$ is defined over multiple action variables $\mathbf{A} = [A_1, \ldots, A_K]$, where it is often the case that the effects of action subspaces defined over the disjoint action variables such as $\{A_1, A_2\}$ and $\{A_3, A_4\}$ may not interact with one another. For example, in control problems over a 2-dimensional plane with state variables of position and velocity for each dimension, the effect of an impulse impacting an object is separated per dimension due to the modularity of underlying causal mechanisms. Motivated by such a modular structure in factored action spaces, we investigate Q-function decomposition with the intervention semantics, leading to a general decomposition scheme that leverages the projected action spaces. In Section 3, we study theoretical properties around the soundness and sample complexity of Q-function decomposition in factored action spaces under the tabular model-based RL setting. In Section 4, we present a practical scheme called action decomposed RL that augments model-free algorithms (Mnih et al. 2015; Fujimoto, Meger, and Precup 2019) with an improved critic learning procedure based on Q-function decomposition. In Section 5, we implement this scheme with Deep Q-networks (DQN) (Mnih et al. 2015; Van Hasselt, Guez, and Silver 2016) and batch constrained Q-learning (BCQ) (Fujimoto, Meger, and Precup 2019) for experiments involving online 2D control environments and sepsis treatment offline environments derived from the real world MIMIC-III dataset, demonstrating improved sample efficiency with our proposed approach.

Preliminaries

We consider reinforcement learning (RL) environments as factored Markov decision processes (MDPs) described by



(a) Fully Separable MDPs (b) Separable Effects (c) Non-separable MDP

Figure 1: Decomposable Structures in Factored MDPs. The diagrams shows factored MDPs, where the circles, squares, and diamonds represent state, action, and reward variables. **Fully Separable Structure**: Fig 1a shows a factored MDP that can be fully separable into two independent MDPs, considered in the previous work (Tang et al. 2022; Seyde et al. 2022). **Separable Effects**: Fig 1b shows a factored MDP that has non-separable dynamics and rewards. However, the effects of factored actions are non-interacting, and we study this structure with intervention semantics. **Non-separable Structure**: Fig 1c shows a non-separable factored MDP.

a tuple $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P^0, P, R, \gamma \rangle$, where the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are factored into a set of variables $\mathbf{S} := \{S_1, \ldots, S_M\}$ and $\mathbf{A} := \{A_1, \ldots, A_N\}$, and we denote states and actions by vectors $\mathbf{s} = [s_1, \cdots, s_M]$ and $\mathbf{a} = [a_1, \cdots, a_N]$, respectively. $P^0 := \mathcal{S} \to [0, 1]$ is the initial state distribution, $P := \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, $R := \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1]$ is a discounting factor.

RL agents find an optimal policy $\pi^* \in \{\pi : \mathcal{S} \to \mathcal{A}\}$ that maximizes the value function, $V_\pi(\mathbf{s}) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}^t, \mathbf{a}^t, \mathbf{s}^{t+1}) \,|\, \mathbf{s}^0 = \mathbf{s}]$. Off-policy Q-learning algorithms sample state transition tuples $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ using a behavior policy $\pi^b : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and learns the Q-function $Q_\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}^t, \mathbf{a}^t, \mathbf{s}^{t+1}) \,|\, \mathbf{s}^0 = \mathbf{s}, \mathbf{a}^0 = \mathbf{a}]$ to find the optimal deterministic policy $\pi^*(\mathbf{s})$ by $\mathbf{a}^* = \arg\max_{\mathbf{a} \in \mathcal{A}} Q_{\pi^*}(\mathbf{s}, \mathbf{a})$.

In this paper, we study Q-functions with intervention semantics in the 'no unobserved confounder setting' (Pearl 2009; Schulte and Poupart 2024). Given states $\mathbf{s}$ and $\mathbf{s}'$ at the current and the next time steps, a causal model prescribes the dynamics by structural equations $S' \leftarrow F_{S'}\big(pa(S'), U_{S'}\big)$ defined over all variables $S' \in \mathbf{S}'$, where $U_{S'}$ is an exogenous noise variable and $pa(S') \subset \mathbf{S}$. In the absence of intervention, the structural equations factorize the state transition function as, $P(\mathbf{S}'|\mathbf{S}) = \prod_{m=1}^{M} P\big(S'_m | pa(S'_m)\big)$, which we call *no-op* dynamics, governing the transition of the state variables free from the direct effect of $do(\mathbf{a})$[1].

Action $do(\mathbf{A})$ fixes the value of the next state variables $\text{Eff}(\mathbf{A}) \subset \mathbf{S}'$ subject to the current state variables $\text{Pre}(\mathbf{A}) \subset \mathbf{S}$ [2]. Namely, for all $S' \in \text{Eff}(\mathbf{A})$, the structural equation $F_{S'}$ is replaced by a conditional intervention policy such that $\text{Eff}(\mathbf{A}) \leftarrow \sigma_{\mathbf{A}}\big(\text{Pre}(\mathbf{A})\big)$. Then, the state transition function $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big)$ under intervention follows a truncated factorization as,

$$P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big) = P(\mathbf{S}' \setminus \text{Eff}(\mathbf{A})|\mathbf{S}) \cdot \mathbb{I}\Big[\text{Eff}(\mathbf{A}) = \sigma_{\mathbf{A}}\big(\text{Pre}(\mathbf{A})\big)\Big], \tag{1}$$

where $\mathbb{I}$ is the indicator function.

The deterministic reward can be modeled as a structural equation $R \leftarrow F_R\big(pa(R)\big)$, where $pa(R) \subset \mathbf{S} \cup \mathbf{S}'$ and the domain of R, $dom(R)$ is the range of the reward function [3]. The potential outcome (Rubin 1974) of R subject to action $do(\mathbf{A})$ is prescribed by $R \leftarrow F_R\big(pa(R), do(\mathbf{A})\big)$ and the conditional expected reward $\mathbb{E}[R(\mathbf{s}, \mathbf{a}, \mathbf{s}')|\mathbf{s}, \mathbf{a}]$ can be written as a causal effect,

$$\sum_{\mathbf{S}'} P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big) \sum_{r \in dom(R)} r \cdot \mathbb{I}[r = F_R\big(pa(R), do(\mathbf{A})\big)]. \tag{2}$$

In terms of the Neyman-Rubin potential outcomes frameworks (Rubin 2005), the state $\mathbf{S}$ is the observed confounder since it influences both treatment on $\mathbf{S}'$ and the outcome R. The primary interest in causal statistics is to estimate causal effects in the presence of confounding bias, which stems from the fundamental problem of causal inference. If RL agents could revisit the same states and collect alternative action choices, then we have no issue with the confounding bias in principle. However, we will see that formulating RL as causal effect estimation with intervention semantics offers an opportunity to improve the sample efficiency. There are active research efforts to improve sample efficiency in bandits (Johansson, Shalit, and Sontag 2016; Saito, Ren, and Joachims 2023) and machine learning applications such as recommender systems (Gao et al. 2024).

The decomposition structures in factored MDPs can be categorized by considering what components are separable, as shown in Figure 1. (Tang et al. 2022) studied the case shown in Figure 1a, where the state space and the reward function are all separable per projected action spaces. In such a case, the Q-function can be exactly decomposed as $\mathcal{Q}_\pi(\mathbf{S}, \mathbf{A}) = \sum_{n=1}^{N} Q^n(\mathbf{S}, A_n)$, where each $Q^n(\mathbf{S}, A_n)$ is

[1] We use do operator to emphasize the intervention semantics of an action $\mathbf{a}$.

[2] We use terminology from planning where $\text{Eff}(\mathbf{A})$ and $\text{Pre}(\mathbf{A})$ refer to the action's effect and precondition. $\text{Eff}(\mathbf{A})$ are the state variables that are controllable by action, whereas $\mathbf{S}' \setminus \text{Eff}(\mathbf{A})$ are non-controllable state variables. The intervention policy can be depend on the state variables $\text{Pre}(\mathbf{A})$.

[3] We abused the notation. R denotes the random variable that defines the reward function with a structural equation F_R as well as the reward function.

defined over the whole state space, yet restricted to apply actions in the subspace $\mathcal{A}_n$. (Rebello et al. 2023) proposed an importance sampling estimator that leverages the fully separable MDP structure. (Seyde et al. 2022) demonstrated state-of-the-art performance for solving continuous control problems through a simple modification that replaces Q-functions with a linear decomposition in deep Q-learning (DQN) algorithms (Mnih et al. 2015).

Methods

Projected Action Space MDPs

Let's consider the decomposition case shown in Figure 1b, where we assume that the effect of factored actions are non-interacting.

Assumption 1. Given a partition of action variables $\mathbf{A} = [\mathbf{A}_1, \ldots, \mathbf{A}_K]$, there exists a partition over state variables $\mathbf{S}' = [\mathbf{S}'_1, \ldots, \mathbf{S}'_K, \mathbf{S}'_{K+1}]$ such that $\mathbf{S}'_k = \text{Eff}(\mathbf{A}_k)$ and $\mathbf{S}'_{K+1} \cap \text{Eff}(\mathbf{A}) = \emptyset$, where $\text{Eff}(\mathbf{A}) = \cup_{k=1}^{K} \text{Eff}(\mathbf{A}_k)$.

Then, we can factorize the interventional state transition function $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big)$ as

$$P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big) = P(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A})) \prod_{k=1}^{K} P\big(\mathbf{S}'_k|\mathbf{S}, do(\mathbf{A}_k)\big). \tag{3}$$

In terms of the interventional state transition function, $Q_\pi(\mathbf{s}, \mathbf{a})$ can be written as

$$Q_\pi(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}'} P\big(\mathbf{s}'|\mathbf{s}, do(\mathbf{a})\big)\big[R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q_\pi\big(\mathbf{s}', \pi(\mathbf{s}')\big)\big]. \tag{4}$$

For each of the projected action space $\mathcal{A}_k$ subject to action variables $\mathbf{A}_k$, we define a projected action space MDP as follows.

Definition 1 (Projected Action Space MDP)**.** Given a factored MDP $\mathcal{M} := \langle \mathcal{S}, \mathcal{A}, P^0, P, R, \gamma \rangle$, and a projected action space $\mathcal{A}_k := \times_{A_i \in \mathbf{A}_k} A_i$, a projected action space MDP $\mathcal{M}^k := \langle \mathcal{S}, \mathcal{A}_k, P^0, P, R, \gamma \rangle$ is defined by the state transition function in terms of the interventional distribution $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big)$ as,

$$P\big(\mathbf{S}'_k|\mathbf{S}, do(\mathbf{A}_k)\big) P\big(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A})\big) \prod_{i \in [1..K], i \neq k} P(\mathbf{S}'_i|\mathbf{S}), \tag{5}$$

where $P\big(\mathbf{S}'_k|\mathbf{S}, do(\mathbf{A}_k)\big)$ is the interventional distribution over $\mathbf{S}'_k$ under the effect of $do(\mathbf{A}_k)$.

Under Assumption 1, we can rewrite the state transition function $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big)$ with $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big)$ as,

$$P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big) = P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big) \prod_{i=1, i \neq k}^{K} \frac{P\big(\mathbf{S}'_i|\mathbf{S}, do(\mathbf{A}_i)\big)}{P(\mathbf{S}'_i|\mathbf{S})} \tag{6}$$

$$= P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big) \prod_{i=1, i \neq k}^{K} \frac{\mathbb{I}\big[\mathbf{S}'_i = \sigma_{\mathbf{A}_i}\big(\text{Pre}(\mathbf{A}_i)\big)\big]}{P(\mathbf{S}'_i|\mathbf{S})} \tag{7}$$

$$= P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big) \prod_{i=1, i \neq k}^{K} 1 \Big/ P\Big(\mathbf{S}'_i = \sigma_{\mathbf{A}_i}\big(\text{Pre}(\mathbf{A}_i)\big)|\mathbf{S}\Big), \tag{8}$$

where $\sigma_{\mathbf{A}_i}\big(\text{Pre}(\mathbf{A}_i)\big)$ is the conditional intervention policy that fixes the value of $\text{Eff}(\mathbf{A}_i)$.

Proposition 1 shows that the Q-function $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ in $\mathcal{M}^k$ can be written in terms of $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big)$.

Proposition 1 (Projected Q-function)**.** Given a projected MDP $\mathcal{M}^k$ over action variables $\mathbf{A}_k$, the Q-function $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ can be recursively written as,

$$Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k) = \sum_{\mathbf{s}'} P\big(\mathbf{s}'|\mathbf{s}, do(\mathbf{a}_k)\big)\big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{s}') + \gamma Q_{\pi_k}(\mathbf{s}', \pi_k(\mathbf{s}'))\big],$$

where $\pi_k : \mathcal{S} \to \mathcal{A}_k$ is a factored policy for $\mathbf{A}_k$.

Proof. The actions in $\mathcal{A}_k$ only intervene on $\mathbf{S}'_k$ and the rest of the state variables $\mathbf{S}' \setminus \text{Eff}(\mathbf{A}_k)$ follow the *no-op* dynamics. Namely, the state transition follows Eq. (5). By definition, $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ is the value of applying action $do(\mathbf{A}^0 = \mathbf{a}_k)$ in state $\mathbf{S}^0 = \mathbf{s}$ at time step $t = 0$, and applying actions $do\big(\mathbf{A}^t = \pi_k(\mathbf{S}^t)\big)$ for the remaining time steps $t = 1..\infty$. It is easy to rewrite $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k) = \sum_{\mathbf{S}^1} P(\mathbf{S}^1|\mathbf{s}, do(\mathbf{a}_k))R(\mathbf{s}, \mathbf{a}_k, \mathbf{S}^1) + \sum_{t=1}^{\infty} \sum_{\mathbf{S}^1..\mathbf{S}^t} \prod_{j=0}^{t} \gamma^t P(\mathbf{S}^{j+1}|\mathbf{S}^j, do(\pi_k(\mathbf{S}^j)))R(\mathbf{S}^t, \pi_k(\mathbf{S}^t), \mathbf{S}^{t+1})$ as desired. □

From Eq. (6), we see that the interventional state transition functions $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A})\big)$ and $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big)$ are different only in $P(\mathbf{S}'_i|\cdot)$ for $i \neq k$ such that we can weight the projected Q-functions $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ to represent $Q_\pi(\mathbf{s}, \mathbf{a})$.

Definition 2 (Weighted Projected Q-functions)**.** Given a policy π and its projected policy π_k, a weighted projected Q-function $\tilde{Q}_\pi(\mathbf{s}, \mathbf{a}_k)$ can be defined as,

$$\tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k) = \sum_{\mathbf{s}'} \frac{P\big(\mathbf{s}'|\mathbf{s}, do(\mathbf{a}_k)\big)}{\rho_{-k}(\mathbf{s}, \mathbf{s}')} \big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{s}') + \gamma \tilde{Q}_{\pi_k}\big(\mathbf{s}', \pi_k(\mathbf{s}')\big)\big] \mathbb{I}\big[\text{Eff}(\mathbf{A}) = \sigma_{\mathbf{A}}\big(\text{Pre}(\mathbf{A})\big)\big], \tag{9}$$

where $\rho_{-k}(\mathbf{s}, \mathbf{s}') = \prod_{i=1, i \neq k}^{K} P\big(\mathbf{s}'_i = \sigma_{\mathbf{a}_i}(\text{Pre}(\mathbf{a}_i))|\mathbf{s}\big)$ from Eq. (8), and the values of the next state variables $\text{Eff}(\mathbf{A})$ are consistent to action $do\big(\pi(\mathbf{a}')\big)$.

Note that there are two main differences between $\tilde{Q}_{\pi_k}(\mathbf{s}', \mathbf{a}'_k)$ and $Q_{\pi_k}(\mathbf{s}', \mathbf{a}'_k)$. First, the weighted projected Q-functions are defined relative to the action trajectories that follow the policy π in the action space $\mathcal{A}$. Second, the weights $\rho_{-k}(\mathbf{s}, \mathbf{s}')$ represent the propensity score (Rubin 2005) of the *no-op* dynamics.

Model-based Factored Policy Iteration

Next, we present a concrete model-based RL algorithm, called model-based factored policy iteration (MB-FPI) for studying the soundness of the Q-function decomposition, and the improvement in the sample complexity, under the following assumptions.

Assumption 2. The *no-op* dynamics $P(\mathbf{S}'|\mathbf{S}) = \prod_{k=1}^{K} P(\mathbf{S}'_k|\mathbf{S}) P\big(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A})\big)$ is positive.

Assumption 3. Given a behavior policy $\pi^b(\mathbf{a}|\mathbf{s})$, and its factored policy $\pi^b_k(\mathbf{a}_k|\mathbf{s})$, the supports of $P(\mathbf{s}'|\mathbf{s})$ induced by π^b and π^b_k are the same in every states $\mathbf{s}$, namely, $\big\{\mathbf{s}' | \sum_{\mathbf{a}} P(\mathbf{s}'|\mathbf{a}, \mathbf{s})\pi^b(\mathbf{a}|\mathbf{s}) > 0\big\} = \big\{\mathbf{s}' | \sum_{\mathbf{a}_k} P(\mathbf{s}'|\mathbf{a}_k, \mathbf{s})\pi^b_k(\mathbf{a}_k|\mathbf{s}) > 0\big\}$.

Algorithm 1 Model-based Factored Policy Iteration

1: Learn intervention policies $\sigma_{\mathbf{A}_k}$ for all $k = 1toK$, the factored *no-op* dynamics $P\big(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A})\big)$, and the reward function $R(\mathbf{S}, \mathbf{A}, \mathbf{S}')$.
2: Initialize an arbitrary factored policy $\pi = [\pi_1, \ldots, \pi_K]$
3: **repeat**
  //POLICY EVALUATION
4: **for** k = 1 to K **do**
5: $P(\mathbf{S}'|\mathbf{S}, \mathbf{A}_k) \leftarrow \mathbb{I}[\mathbf{S}'_k = \sigma_{\mathbf{A}_k}(\mathbf{S})] \prod_{i=1, i\neq k}^{K} \mathbb{I}[\mathbf{S}'_i = \sigma_{\pi_i(\mathbf{S})}(\mathbf{S})] P\big(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A})\big)$
6: **for** each $(\mathbf{s}, \mathbf{a}_k) \in (\mathbf{S}, \mathbf{A}_k)$ **do**
7: $\tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k) \leftarrow \sum_{\mathbf{S}'} P(\mathbf{S}'|\mathbf{s}, \mathbf{a}_k)[R(\mathbf{s}, \mathbf{a}_k, \mathbf{S}') + \gamma \tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k)]$
  //POLICY IMPROVEMENT
8: Select an arbitrary k for policy improvement
9: **for** each state $\mathbf{s} \in \mathcal{S}$ **do**
10: $\pi_k(\mathbf{s}) \leftarrow \arg\max_{\mathbf{a}_k} \tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$
11: **until** No change in $\pi(\mathbf{s})$

In words, the above assumptions ensure that the exploration with projected actions in $\mathcal{A}_k$ covers the same state space visited by the action space $\mathcal{A}$.

Algorithm 1 is a model-based algorithm that learns models of projected MDPs $\mathcal{M}^k$ (line 1), and performs policy iteration (Sutton and Barto 2018) to find a locally optimal policy $\pi^*(\mathbf{s}) = [\pi_1^*(\mathbf{s}), \ldots, \pi_K^*(\mathbf{s})]$. The policy evaluation procedure computes state transition function $P(\mathbf{S}'|\mathbf{S}, \mathbf{A}_k)$ (line 5) and updates Q-tables $\tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ (line 7). The policy improvement procedure selects an arbitrary π_k to improve (line 11). The algorithm terminates and returns the locally optimal policy π^* when there is no change during updating π.

Under the intervention semantics, $\sigma_{\mathbf{A}_k}$ fixes the value of $\mathbf{S}'_k$, and we see that $P(\mathbf{S}'|\mathbf{S}, \mathbf{A}_k)$ constrains the state trajectory to follow not only a single factored policy π_k, but also all other policies following the factorization shown in Eq. (3). As a result, $\tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ maintains the values of $\mathbf{a}_k$ in $\mathbf{s}$ while fixing all the rest of the actions $\mathbf{a}_i$ to follow the current policy π_i for all $i \in [1..K]$ except $i = k$. This implies the consistency between two value functions that are evaluated on a deterministic policy π and its projection π_k, namely, $\tilde{V}_{\pi}(\mathbf{s}) = \tilde{V}_{\pi_k}(\mathbf{s})$, where $\tilde{V}_{\pi}(\mathbf{s}) = \tilde{Q}_{\pi}\big(\mathbf{s}, \pi(\mathbf{s})\big)$ and $\tilde{V}_{\pi_k}(\mathbf{s}) = \tilde{Q}_{\pi_k}\big(\mathbf{s}, \pi_k(\mathbf{s})\big)$. However, it does not guarantee to find the globally optimal policy. The following Theorem 1 shows that MB-FPI converges to the globally optimal policy under the additional assumption that the underlying $Q_{\pi}(\mathbf{s}, \mathbf{a})$ is monotonic.

Theorem 1 (Convergence of MB-FPI)**.** MB-FPI converges to a locally optimal policy in a finite number of iterations. If the underlying $Q_{\pi}(\mathbf{s}, \mathbf{a})$ is monotonic, then MB-FPI converges to the optimal policy π^*. =

Theorem 2 (Sample Complexity of MB-FPI)**.** Given $\delta \in (0, 1)$ and $\epsilon > 0$, the sample complexity for learning the *no-op* dynamics $P(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A}))$ and the intervention policy $\sigma_{\mathbf{A}_k}(\text{Pre}(\mathbf{A}_k)$ within error bound ϵ with at least probability $1 - \delta$ are $N_P \geq \frac{|\mathbf{S}_{K+1}||\mathbf{S}||\mathbf{S}\setminus\mathbf{S}_{K+1}|}{\epsilon^2} \log \frac{2|\mathbf{S}||\mathbf{S}\setminus\mathbf{S}_{K+1}|}{\delta}$ and $N_\sigma \geq \frac{|\text{Eff}(\mathbf{A}_k)||\text{Pre}(\mathbf{A}_k)|}{\epsilon^2} \log \frac{2|\text{Pre}(\mathbf{A}_k)|}{\delta}$, respectively.

N_p and N_σ highlight the improved sample complexity due to the factored MDP structure as well as the non-interacting effects between projected action spaces, compared with the sample complexity of the full dynamics model of the non-separable MDPs, $N \geq |\mathbf{S}|^2|\mathbf{A}|/\epsilon^2 \cdot \log(2|\mathbf{S}||\mathbf{A}|/\delta)$.

Action Decomposed RL Framework

In the previous section, we investigated theoretical properties around Q-function decomposition in factored action spaces and its advantage in a tabular model-based RL setting, which implicitly assumes that the intervention policies can be learned in a well-structured discrete state space such that we know how the state variables change under the influence of actions. In practical environments, such assumptions don't hold if the state space is continuous, unstructured, or high-dimensional. Therefore, we take the essential ideas developed in the previous section, and present a practical and general scheme that augments two components that leverages the Q-function decomposition for the critic learning in model-free algorithms.

First, we use the projected Q-functions $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ as a basis for approximating the Q-function $Q_{\pi}(\mathbf{s}, \mathbf{a})$. Let's consider a function class $\mathcal{F}$ be comprised of the projected Q-functions $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$, $\mathcal{F} := \{\mathcal{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k) \mid k \in [1..K]\}$. A linear combination of the projected Q-functions can be written as $\tilde{Q}_{\pi}(\mathbf{s}, \mathbf{a}) = \sum_{k=1}^{K} w_k Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ for some coefficients $\{w_k \mid k \in [1..K]\}$, and more general combinations using neural networks could be written as

$$\tilde{Q}_{\pi}(\mathbf{s}, \mathbf{a}) = F(Q_{\pi_1}\big(\mathbf{s}, \mathbf{a}_1), \ldots, Q_{\pi_K}(\mathbf{s}, \mathbf{a}_K)\big). \tag{10}$$

The second component is a data augmentation procedure that generates samples of the projected MDPs by using the learned dynamics models and the reward model for training individual projected Q-functions $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$. Next, we present two concrete algorithms that modify the critic learning steps in DQN and BCQ.

Action Decomposed DQN

One of the main challenges in DQN-style algorithms is handling large discrete action spaces, especially when the size of the action space grows in a combinatorial manner. In general, value decomposition approaches introduce local value functions and perform action selection in a local manner to avoid enumerating all possible combinations of actions. Here, we leverage the projected Q-functions in factored action spaces to improve the sample efficiency of DQN.

Algorithm 2 presents our proposed modification to DQN, which we call action decomposed DQN (AD-DQN). Given a factored action space $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_K$, the Q-network $\tilde{Q}_{\pi}(\mathbf{s}, \mathbf{a})$ in AD-DQN combines K sub Q-networks, each corresponding to $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ with a mixer network F (Rashid et al. 2020) that implements Eq. (10). The mixer network could be a linear layer, or a non-linear MLP for combining the values of $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ to yield the final outcome of $\tilde{Q}_{\pi}(\mathbf{s}, \mathbf{a})$.

During training (line 11-15), we first sample a mini-batch B from a replay buffer D. For training sub Q-networks

Algorithm 2 Action-Decomposed DQN

1: **for** episode = 1 to num-episodes **do**
2: $\mathbf{s} \leftarrow$ initial state from environment
3: **for** step = 1 to episode-length **do**
//COLLECT SAMPLES
4: **if** random $< \epsilon$ **then**
5: draw a random action $\mathbf{a}$
with probability p, draw a projected action
6: **else**
7: select actions $[\mathbf{a}_1, \ldots, \mathbf{a}_K]$ from $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$
8: Sample a transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$ from environment
9: Store the sample to replay buffer D
if action was $\mathbf{a}_k$, store the sample to buffer D_k
10: $\mathbf{s} \leftarrow \mathbf{s}'$
//TRAIN Q-NETWORKS
11: Sample a batch B from buffer D
12: **for** k= 1 to K **do**
13: Modify B to B_k to follow $P(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k))$
using learned dynamics and reward model
14: Train $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ with B_k
15: Train $\tilde{Q}_\pi(\mathbf{s}, \mathbf{a})$ with B
16: Update target network
//TRAIN DYNAMICS AND REWARD MODELS
17: Sample a batch B from D, train reward model
18: **for** k = 1 to K **do**
19: Sample a batch B_k from D_k, train dynamics

$Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$, we modify the samples in B to follow the transition dynamics $P\big(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k)\big)$ of the projected MDP $\mathcal{M}^k$ with learned dynamics models (line 13). Namely, for a transition $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \in B$, we generate $(\mathbf{s}, \mathbf{a}_k, \tilde{\mathbf{s}}', \tilde{r})$ by projecting $\mathbf{a}$ to $\mathbf{a}_k$, sampling $\tilde{\mathbf{s}}' \sim P\big(\mathbf{S}'|\mathbf{s}, do(\mathbf{a}_k)\big)$, and evaluating the learned reward model, $\tilde{r} = R(\mathbf{s}, \mathbf{a}_k, \tilde{\mathbf{s}}')$. Next, we train the full network $\tilde{Q}_\pi(\mathbf{s}, \mathbf{a})$ including the mixer F that combines the values from $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ (line 15). While collecting samples or training the networks, we select actions $\mathbf{a}$ by concatenating the projected actions $[\mathbf{a}_1, \ldots, \mathbf{a}_K]$ using the global Q-network, $\mathbf{a}_k \leftarrow \arg\max_{\mathbf{A}_k} \tilde{Q}_\pi(\mathbf{s}, \mathbf{A}_k)$.

The steps for training dynamics and reward models make AD-DQN a model-based RL algorithm (line 17-19). When we have access to the intervention policies, the dynamics model can be further simplified to the single *no-op* dynamics model $P\big(\mathbf{S}'|\mathbf{S}, \text{Eff}(\mathbf{A})\big)$, which is sufficient to generate $\tilde{\mathbf{s}}'$. However, it may be difficult to learn such intervention policies in practical RL environments. Therefore, AD-DQN learns the dynamics model for the projected MDPs using the samples generated by projected actions (line 5), and it generates the synthetic data $(\mathbf{s}, \mathbf{a}_k, \tilde{\mathbf{s}}', \tilde{r})$ from the samples $(\mathbf{s}, \mathbf{a}, \mathbf{s}, r)$ obtained from the environment to train $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$.

Action Decomposed BCQ

In off-policy RL approaches (Lange, Gabel, and Riedmiller 2012; Levine et al. 2020; Uehara, Shi, and Kallus 2022), a behavior policy generates a batch of samples that are uncorrelated from the distributions under the evaluation policy. Fujimoto, Meger, and Precup (2019) showed that such a discrepancy induces extrapolation error that severely degrades performance in offline as well as online environments. The

Algorithm 3 Action-Decomposed BCQ

//TRAIN DYNAMICS AND REWARD MODELS
1: Train a reward model with offline data D
2: **for** k = 1 to K **do**
3: Collect D_k with projected actions $\mathbf{a}_k$ from D
4: Train dynamics model with D_k
5: **for** t = 1 to max-training **do**
//TRAIN Q-NETWORKS AND GENERATIVE MODELS
6: Sample a batch B from D
7: **for** k=1 to K **do**
8: Modify B to B_k to follow $P(\mathbf{S}'|\mathbf{S}, do(\mathbf{A}_k))$
9: Train $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ and $G_k(\mathbf{a}_k|\mathbf{s})$ for BCQ
10: Train $Q_\pi(\mathbf{s}, \mathbf{a})$ and $G(\mathbf{a}|\mathbf{s})$ for BCQ
11: Update target network

mismatch between the data distribution and the target distribution introduces a selection bias since the empirical mean is taken over the batch, and it also induces a large variance if the importance sampling method is employed in the region where the inverse propensity score is large. Therefore, critic learning in offline RL algorithms has an additional challenge that stems from limitations around sample collection, which calls for methods that improve sample efficiency. Algorithm 3 is another variation to critic learning in BCQ, which we call action decomposed BCQ (AD-BCQ). AD-BCQ also introduces small changes to the base algorithm BCQ, similar in the way AD-DQN modifies DQN. We augment the steps for learning the dynamics models for the projected MDPs, and train the deep Q-networks $\tilde{Q}_\pi(\mathbf{s}, \mathbf{a})$ that combines sub Q-networks $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$. Since BCQ is an offline RL algorithm, there is no step for collecting samples. Therefore we preprocess the whole collected data D and train the dynamics and reward models before training the Q-networks (line 1-4). For training the Q-networks, we sample a mini-batch B from the dataset D, and modify it as B_k to follow the transitions in the projected MDPs for training the projected Q-networks $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ and the generative models $G_k(\mathbf{a}_K|\mathbf{s})$ (line 7-9). After training projected Q-networks $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$, we train the global deep Q-network including the mixer (line 10).

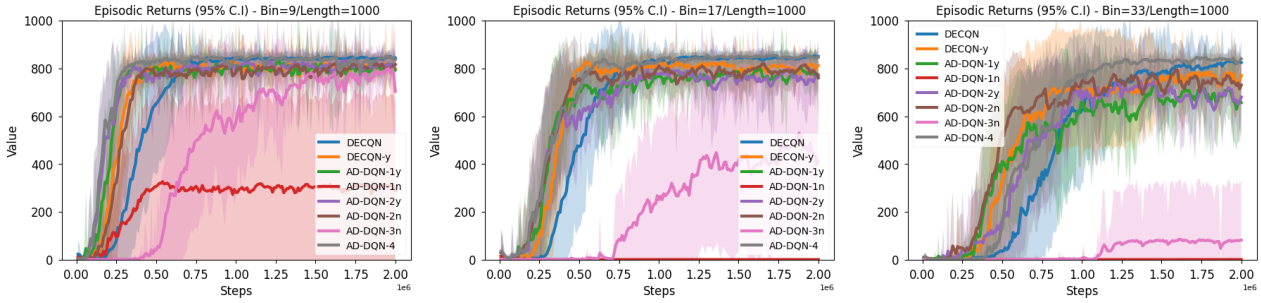
Experiments

We conduct experiments in 2D-point mass control environment from deep mind control suite (Tassa et al. 2018) and MIMIC-III sepsis treatment environment (Komorowski et al. 2018; Goldberger et al. 2000; Johnson et al. 2016; Killian et al. 2020) for evaluating AD-DQN and AD-BCQ, respectively. Both RL environments are close to the real-world problems and they also have well-defined factored action spaces. In addition, there are the state-of-the-art algorithms that utilizes the linear Q-function decomposition.

2D Point-Mass Control

We evaluate AD-DQN and the decoupled Q-networks (DECQN) (Seyde et al. 2022) [4]. For both algorithms, we

[4]We also evaluated variations of DQN that flattening the factored action space, and they all failed to learn Q-functions due to the large number of action labels.



(a) 9x9 Action Space (b) 17x17 Action Space (c) 33x33 Action Space

Figure 2: Comparing the values in the test environment on three action spaces. The X-axis shows 2 million steps of training (2000 episodes) and the Y-axis is the value evaluated from a test environment. The plot aggregates 10 trials with the random seeds from 1 to 10 for training and 1001 to 1010 for testing.

discretized the action space of the continuous actions to apply DQN-style algorithms. In 2D point-mass control task, the state space is 4 dimensional continuous space comprised of the position and velocity in the x and y axis on a plane , and the action space is 2 dimensional factored action space that defines Q_{π_x} and Q_{π_y}. For measuring the performance, we evaluated average episodic returns in a separate evaluation environment for 10 trials.

Algorithm Configurations In the experiment, we evaluated various algorithm configurations by varying the hyperparameters. Table 1 summarizes all configurations. All configurations except for the AD-DQN-3n shares the weights of Q_{π_x} and Q_{π_y}. The mixer combines the two decomposed value functions by a simple average, passing a 3 layer MLP with ReLu, or 2 linear layers. Data augmentation learns the dynamics for each projected MDP and synthesize the samples for training Q_{π_x} and Q_{π_y}. All algorithm configurations select the actions from each head of Q_{π_x} and Q_{π_y} and if a mixer is used, a greedy action is selected by evaluating the outcome of the mixer. AD-DQN-4 follows AD-DQN but it switches to the pure model-free setting when the evaluated value of the current greedy policy is greater than 500.

Table 1: Algorithm configurations in 2D-Point mass control experiment.

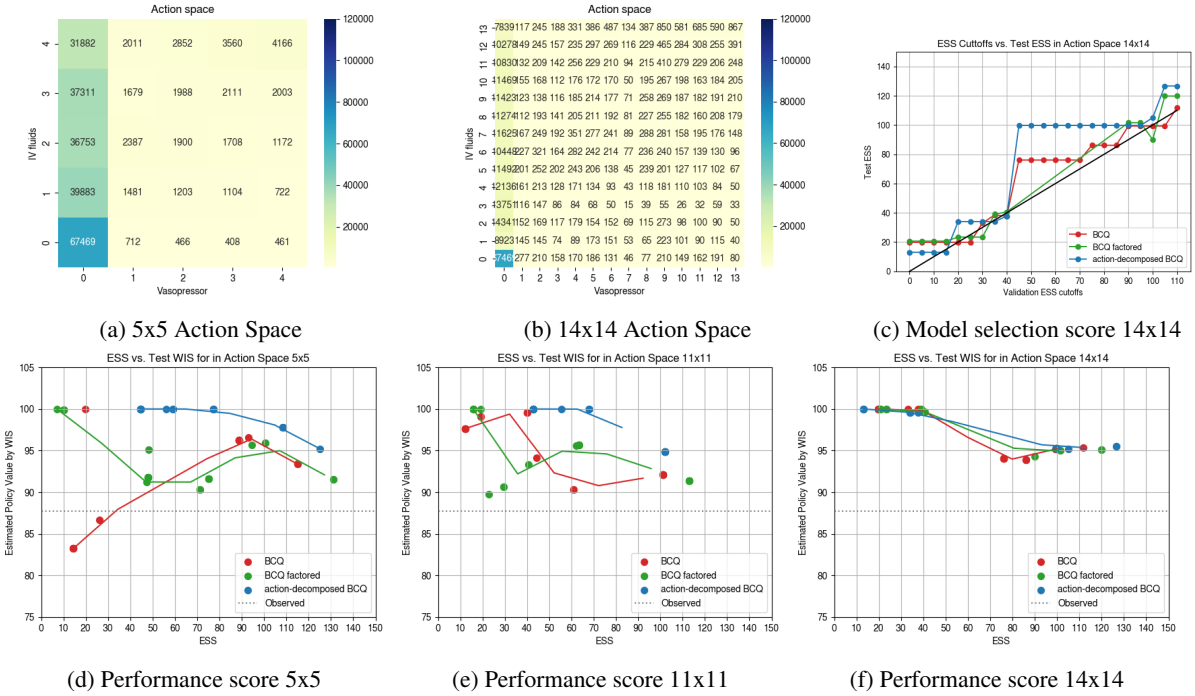
Algorithms	Weights	Mixer	Data aug
DECQN	shared	average	no
DEQCN-y	shared	average	yes
AD-DQN-1y	shared	ReLu	yes
AD-DQN-1n	shared	ReLu	no
AD-DQN-2y	shared	2 linear layers	yes
AD-DQN-2n	shared	2 linear layers	no
AD-DQN-3n	not shared	2 linear layers	no
AD-DQN-4	shared	2 linear layers	yes

Results Figure 2 shows the episodic returns in 3 action spaces by increasing the number of discrete bins from 9 to 33. First of all, DECQN can be seen as one of the algorithm configuration of AD-DQN with a mixer network that aggregates the value with a simple average without performing the data augmentation step. Overall, we see that the weight sharing between two projected Q-networks performs better, and the mixer network without the non-linear ReLu activation performs the best. The best algorithm configuration is AD-DQN-4, but other configurations also improve the speed of convergence compared with the baseline DECQN (blue).

MIMIC-III Sepsis Treatment

We evaluate AD-BCQ and factored BCQ (Tang et al. 2022) in the sepsis treatment environment derived from MIMIC-III database version 1.4 (Goldberger et al. 2000; Johnson et al. 2016; Komorowski et al. 2018). Processing the MIMIC-III dataset, we obtained a cohort of 177,877 patients and applied 70/15/15 split for training, validation and testing. Each patient has 5 non-time varying demographic features and 33 time varying features collected at 4-hour intervals over 72 hours that make up a discrete time-series. The reward is assigned 100 for the discharged state, -100 for the mortality, and 0 for otherwise. Among possible choices for the state representation learning (Killian et al. 2020), we used the approximate information state encoder (Subramanian et al. 2022) and obtained 64 dimensional state space. The action space is comprised of two continuous variables for the total volume of intravenous fluid and the amount of vassopressors treated. We experimented in the action spaces by varying the number of bins for the discretization from 5x5 to 14x14. of those two continuous variables by dividing the quantiles evenly, as shown in Figure 3a and 3b.

Performance Measure Let π be a policy under the evaluation and π_b be a observed behavioral policy in the offline data. Given an episode $(s_1, a_1, r_1, \ldots, s_L, a_L, r_L)$ of length L, the per-step importance ratio is $\rho_t = \pi(a_t|s_t)/\pi_b(a_t|s_t)$ and the cumulative importance ratio is $\rho_{1:t} = \prod_{t'=1}^{t} \rho_{t'}$. Given m episodes, the off-policy evaluation value from the weighted importance sampling (WIS) can be computed as follows. $\hat{V}_{\text{WIS}}(\pi) = 1/m \sum_{j=1}^{m} \rho^{(j)}_{1:L^{(j)}} / w_{L^{(j)}} \left(\sum_{t=1}^{L^{(j)}} \gamma^{t-1} r_t^{(j)} \right)$, where $w_t = 1/m \sum_{j=1}^{m} \rho^{(j)}_{1:t}$ is the average cumulative importance ratio at time step t. The superscript (j) indicates that the length $L^{(j)}$, rewards $r_t^{(j)}$, and the accumulated impor-



(a) 5x5 Action Space

(b) 14x14 Action Space

(c) Model selection score 14x14

(d) Performance score 5x5

(e) Performance score 11x11

(f) Performance score 14x14

Figure 3: Figure 3a and 3b visualize the number of samples for two discrete action spaces in the training set. The offline RL dataset is split into 70% training, 15% for validation, and 15% test sets. Figure 3c shows the test effective sample size (ESS) score of the policy functions selected by the validation ESS score. Figure 3d–3f show the performance score evaluated from the test set. The AD-BCQ clearly improves upon two baselines, BCQ and factored-BCQ.

tance ratio $\rho^{(j)}_{1:t}$ are computed relative to the j-th episode. The effective sample size (ESS) of the importance sampling can be computed as $\text{ESS} = (\sum_{t=1}^{N} w_t)^2 / \sum_{t=1}^{N} w_t^2$, where the weights are not normalized (Martino, Elvira, and Louzada 2016). While computing the weights, we clip the values by $1,000$ and softened π by mixing a random policy, namely $\hat{\pi}(a|s) = \mathbb{I}_{a=\pi(s)}(1-\epsilon) + \mathbb{I}_{a\neq\pi(s)}(\epsilon/|\mathcal{A}| - 1)$ with $\epsilon = 0.01$ following (Tang and Wiens 2021).

Evaluation We follow the evaluation protocol using the open source environment offered by (Tang and Wiens 2021; Tang et al. 2022). For all hyperparameters swept and 20 random trials, we select the best model with the highest WIS in the validation set given a minimum ESS cutoff value as shown in the black line in Figure 3c. Using the selected model, we evaluate its WIS and ESS in the test set for the final comparison. In terms of the model selection score, we see that AD-BCQ (blue) shows the higher ESS scores compared with the baselines. Figure 3d–3f show the test WIS scores on the Y-axis, and the ESS score on the X-axis. The observed value of the clinician's policy employed to the patients is 87.75 in the test set, shown in the black dotted lines. Since this data set has only a single reward at the end of each episode and the discounting factor is 1.0 for the evaluation, this value is the average of the rewards of episodes. Each algorithm was trained on 20 different random seeds, and we see that AD-BCQ forms a Pareto frontier compared to the other two algorithms, confirming that AD-BCQ learns policy functions that dominates the performance of the baselines.

Conclusion

In this paper, we study the decomposable action structure in factored MDPs by considering the intervention semantics. The primary advantage of the Q-function decomposition approach is known to improve the sample efficiency in the presence of large discrete action spaces, one of the major challenges in reinforcement learning. The theoretical investigation shows that we can extend the Q-function decomposition scheme from the fully separable MDPs to non-separable MDPs if the effects of the projected action spaces are non-interacting. In addition, if the underlying Q-function is monotonic, a series of local improvements along with the projected action spaces still returns the optimal policy. Transferring such findings in ideal settings to a more practical DQN-style algorithms, we present action decomposed reinforcement learning framework that improves the critic learning procedures and demonstrated the improved sample efficiency in realistic online and offline environments. In future work, there are several interesting research directions. First, we could jointly learn the causal state representations and the causal mechanisms to further improve the sample efficiency, as promised in the theoretical results. Second, a more thorough analysis would enhance the theoretical understanding of the Q-function decomposition. Last, the extension of problem settings to the 'unobserved confounder setting' will present considerable research challenges, while also enabling wider real world applications where the full knowledge of state spaces is lacking.

References

Agarwal, A.; Jiang, N.; Kakade, S. M.; and Sun, W. 2019. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 32: 96.

Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; and Coppin, B. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, 2052–2062. PMLR.

Gao, C.; Zheng, Y.; Wang, W.; Feng, F.; He, X.; and Li, Y. 2024. Causal Inference in Recommender Systems: A Survey and Future Directions. *ACM Transactions on Information Systems*, 42(4): 1–32.

Goldberger, A. L.; Amaral, L. A.; Glass, L.; Hausdorff, J. M.; Ivanov, P. C.; Mark, R. G.; Mietus, J. E.; Moody, G. B.; Peng, C.-K.; and Stanley, H. E. 2000. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *circulation*, 101(23): e215–e220.

Johansson, F. D.; Shalit, U.; and Sontag, D. 2016. Learning representations for counterfactual inference. In *33rd International Conference on Machine Learning, ICML 2016*, 4407–4418. International Machine Learning Society (IMLS).

Johnson, A. E.; Pollard, T. J.; Shen, L.; Lehman, L.-w. H.; Feng, M.; Ghassemi, M.; Moody, B.; Szolovits, P.; Anthony Celi, L.; and Mark, R. G. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3(1): 1–9.

Killian, T. W.; Zhang, H.; Subramanian, J.; Fatemi, M.; and Ghassemi, M. 2020. An empirical study of representation learning for reinforcement learning in healthcare. *arXiv preprint arXiv:2011.11235*.

Komorowski, M.; Celi, L. A.; Badawi, O.; Gordon, A. C.; and Faisal, A. A. 2018. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature medicine*, 24(11): 1716–1720.

Lange, S.; Gabel, T.; and Riedmiller, M. 2012. Batch reinforcement learning. In *Reinforcement learning: State-of-the-art*, 45–73. Springer.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Martino, L.; Elvira, V.; and Louzada, F. 2016. Alternative effective sample size measures for importance sampling. In *2016 IEEE Statistical Signal Processing Workshop (SSP)*, 1–5. IEEE.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Pearl, J. 2009. *Causality*. Cambridge University Press.

Pearl, J. 2019. The Seven Tools of Causal Inference, with Reflections on Machine Learning. *Communications of the ACM*, 62(3): 54–60.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Pitis, S.; Creager, E.; and Garg, A. 2020. Counterfactual Data Augmentation Using Locally Factored Dynamics. In *Advances in Neural Information Processing Systems*.

Pitis, S.; Creager, E.; Mandlekar, A.; and Garg, A. 2022. MoCoDA: Model-based Counterfactual Data Augmentation. In *Advances in Neural Information Processing Systems*.

Rashid, T.; Samvelyan, M.; De Witt, C. S.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2020. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1): 7234–7284.

Rebello, A. P.; Tang, S.; Wiens, J.; and Parbhoo, S. 2023. Leveraging Factored Action Spaces for Off-Policy Evaluation. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*.

Rubin, D. B. 1974. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5): 688.

Rubin, D. B. 2005. Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469): 322–331.

Russell, S. J.; and Zimdars, A. 2003. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 656–663.

Saito, Y.; Ren, Q.; and Joachims, T. 2023. Off-policy evaluation for large action spaces via conjunct effect modeling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Schölkopf, B. 2022. Causality for Machine Learning. In *Probabilistic and Causal Inference: The Works of Judea Pearl*, 765–804.

Schulte, O.; and Poupart, P. 2024. Why Online Reinforcement Learning is Causal. *arXiv preprint arXiv:2403.04221*.

Seyde, T.; Werner, P.; Schwarting, W.; Gilitschenski, I.; Riedmiller, M.; Rus, D.; and Wulfmeier, M. 2022. Solving Continuous Control via Q-learning. In *Proceedings of the International Conference on Learning Representations*.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676): 354–359.

Son, K.; Kim, D.; Kang, W. J.; Hostallero, D. E.; and Yi, Y. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, 5887–5896. PMLR.

Subramanian, J.; Sinha, A.; Seraj, R.; and Mahajan, A. 2022. Approximate information state for approximate planning and reinforcement learning in partially observed systems. *Journal of Machine Learning Research*, 23(12): 1–83.

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2018. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *International Conference on Autonomous Agents and MultiAgent Systems*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT press.

Tang, S.; Makar, M.; Sjoding, M.; Doshi-Velez, F.; and Wiens, J. 2022. Leveraging factored action spaces for efficient offline reinforcement learning in healthcare. *Advances in Neural Information Processing Systems*, 35: 34272–34286.

Tang, S.; and Wiens, J. 2021. Model selection for offline reinforcement learning: Practical considerations for healthcare settings. In *Machine Learning for Healthcare Conference*, 2–35. PMLR.

Tang, S.; and Wiens, J. 2023. Counterfactual-Augmented Importance Sampling for Semi-Offline Policy Evaluation. *Advances in Neural Information Processing Systems*, 36: 11394–11429.

Tassa, Y.; Doron, Y.; Muldal, A.; Erez, T.; Li, Y.; Casas, D. d. L.; Budden, D.; Abdolmaleki, A.; Merel, J.; Lefrancq, A.; et al. 2018. Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Tavakoli, A.; Fatemi, M.; and Kormushev, P. 2020. Learning to Represent Action Values as a Hypergraph on the Action Vertices. In *Proceedings of the International Conference on Learning Representations*.

Tavakoli, A.; Pardo, F.; and Kormushev, P. 2018. Action branching architectures for deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 32.

Uehara, M.; Shi, C.; and Kallus, N. 2022. A review of off-policy evaluation in reinforcement learning. *arXiv preprint arXiv:2212.06355*.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Wang, J.; Ren, Z.; Liu, T.; Yu, Y.; and Zhang, C. 2020. QPLEX: Duplex Dueling Multi-Agent Q-Learning. In *International Conference on Learning Representations*.

A Missing Proofs

Proposition 1 (Projected Q-function)**.** Given a projected MDP $\mathcal{M}^k$ over action variables $\mathbf{A}_k$, the Q-function $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ can be recursively written as,

$$Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k) = \sum_{\mathbf{s}'} P\big(\mathbf{s}' | \mathbf{s}, do(\mathbf{a}_k)\big) \big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{s}') + \gamma Q_{\pi_k}(\mathbf{s}', \pi_k(\mathbf{s}'))\big],$$

where $\pi_k : \mathcal{S} \to \mathcal{A}_k$ is a factored policy for $\mathbf{A}_k$.

Proof. The actions in $\mathcal{A}_k$ only intervene on $\mathbf{S}'_k$ and the rest of the state variables $\mathbf{S}' \setminus \text{Eff}(\mathbf{A}_k)$ follow the *no-op* dynamics. Namely, the state transition follows

$$P\big(\mathbf{S}'_k | \mathbf{S}, do(\mathbf{A}_k)\big) P\big(\mathbf{S}'_{K+1} | \mathbf{S}, \text{Eff}(\mathbf{A})\big) \prod_{i \in [1..K], i \neq k} P(\mathbf{S}'_i | \mathbf{S}).$$

By definition, $Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ is the value of applying action $do(\mathbf{A}^0 = \mathbf{a}_k)$ in state $\mathbf{S}^0 = \mathbf{s}$ at time step $t = 0$, and applying actions $do\big(\mathbf{A}^t = \pi_k(\mathbf{S}^t)\big)$ for the remaining time steps $t = 1..\infty$. It is easy to rewrite

$$Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k) = \sum_{\mathbf{S}^1} P(\mathbf{S}^1 | \mathbf{s}, do(\mathbf{a}_k)) R(\mathbf{s}, \mathbf{a}_k, \mathbf{S}^1) + \sum_{t=1}^{\infty} \sum_{\mathbf{S}^1 .. \mathbf{S}^t} \prod_{j=0}^{t} \gamma^t P(\mathbf{S}^{j+1} | \mathbf{S}^j, do(\pi_k(\mathbf{S}^j))) R(\mathbf{S}^t, \pi_k(\mathbf{S}^t), \mathbf{S}^{t+1}) \tag{11}$$

$$= \sum_{\mathbf{S}^1} P(\mathbf{S}^1 | \mathbf{s}, do(\mathbf{a}_k)) \big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{S}^1) + \gamma \sum_{t=0}^{\infty} \gamma^t \sum_{\mathbf{S}^1 .. \mathbf{S}^{t+1}} \prod_{j=0}^{t} P(\mathbf{S}^{j+1} | \mathbf{S}^j, do(\pi_k(\mathbf{S}^j))) R(\mathbf{S}^t, \pi_k(\mathbf{S}^t), \mathbf{S}^{t+1})\big] \tag{12}$$

$$= \sum_{\mathbf{S}^1} P(\mathbf{S}^1 | \mathbf{s}, do(\mathbf{a}_k)) \big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{S}^1) + \sum_{t=1}^{\infty} \gamma^t \sum_{\mathbf{S}^2 .. \mathbf{S}^{t+1}} \prod_{j=1}^{t} P(\mathbf{S}^{j+1} | \mathbf{S}^j, do(\pi_k(\mathbf{S}^j))) R(\mathbf{S}^t, \pi_k(\mathbf{S}^t), \mathbf{S}^{t+1})\big] \tag{13}$$

$$= \sum_{\mathbf{S}^1} P(\mathbf{S}^1 | \mathbf{s}, do(\mathbf{a}_k)) \big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{S}^1) + \gamma Q_{\pi_k}(\mathbf{S}^1, \pi_k(\mathbf{S}^1))\big] \tag{14}$$

$$= \sum_{\mathbf{s}'} P(\mathbf{s}' | \mathbf{s}, do(\mathbf{a}_k)) \big[R(\mathbf{s}, \mathbf{a}_k, \mathbf{s}') + \gamma Q_{\pi_k}(\mathbf{s}', \pi_k(\mathbf{s}'))\big], \tag{15}$$

as desired. □

Theorem 1 (Convergence of MB-FPI)**.** MB-FPI converges to a locally optimal policy in a finite number of iterations. If the underlying $Q_\pi(\mathbf{s}, \mathbf{a})$ is monotonic, then MB-FPI converges to the optimal policy π^*. =

Proof. The proof follows the convergence of the policy iteration algorithm.

convergence of policy evaluation We show that the policy evaluation steps in line 4-7 converges to $Q_\pi(\mathbf{s}, \mathbf{a}_k)$. Note that $Q_\pi(\mathbf{s}, \mathbf{a}_k)$ is the value of applying action $[\pi_1 \mathbf{s}), \ldots, \pi_{k-1}(\mathbf{s}), \mathbf{a}_k, \pi_{k+1}(\mathbf{s}), \ldots, \pi_K(\mathbf{s})]$ in $\mathbf{s}$ followed by the actions following π in the subsequent states. Let π be the policy under the evaluation. We can see that $P(\mathbf{S}' | \mathbf{S}, \mathbf{A}_k)$ in line 5 is $P\big(\mathbf{S}' | \mathbf{S}, do(\mathbf{A})\big)$ since

$$\begin{aligned} P(\mathbf{S}' | \mathbf{S}, \mathbf{A}_k) &= \mathbb{I}[\mathbf{S}'_k = \sigma_{\mathbf{A}_k}(\mathbf{S})] \prod_{i=1, i \neq k}^{K} \mathbb{I}[\mathbf{S}'_i = \sigma_{\pi_i(\mathbf{S})}(\mathbf{S})] P\big(\mathbf{S}'_{K+1} | \mathbf{S}, \text{Eff}(\mathbf{A})\big) \\ &= \mathbb{I}[\text{Eff}(\mathbf{A}) = \sigma_{\mathbf{A}}\big(\text{Pre}(\mathbf{A})\big)] P\big(\mathbf{S}'_{K+1} | \mathbf{S}, \text{Eff}(\mathbf{A})\big) \\ &= P\big(\mathbf{S}' | \mathbf{S}, do(\mathbf{A})\big). \end{aligned}$$

Then, we can rewrite $\tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k)$ in Definition 2 as

$$\begin{aligned} \tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a}_k) &= \sum_{\mathbf{s}'} P\big(\mathbf{s}' | \mathbf{s}, do(\mathbf{a})\big) [R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \tilde{Q}_{\pi_k}(\mathbf{s}, \mathbf{a})] \\ &= \sum_{\mathbf{s}'} P\big(\mathbf{s}' | \mathbf{s}, do(\mathbf{a})\big) [R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q_\pi(\mathbf{s}, \mathbf{a})] \\ &= Q_\pi(\mathbf{s}, \mathbf{a}), \end{aligned}$$

where $\mathbf{a}_k$ is the projected action in $\mathcal{M}^k$ and $\mathbf{a} = [\pi_1(\mathbf{s}), \ldots, \pi_{k-1}(\mathbf{s}), \mathbf{a}_k, \pi_{k+1}(\mathbf{s}), \ldots, \pi_{\mathbf{s}})]$ is the action in $\mathcal{M}$ that follows π, yet it replaces $\pi_k(\mathbf{s})$ with $\mathbf{a}_k$. The policy evaluation in the factored policy iteration algorithm is equivalent to the policy evaluation in the synchronous policy iteration algorithm. The only difference is that we restrict the Q-function to vary only at the projected action variables $\mathbf{A}_k$ instead of all action variables $\mathbf{A}$. Therefore, we can conclude that the policy evaluation in the factored policy iteration algorithm converges.

local convergence of policy improvement The convergence of the policy improvement step (line 8-10) follows from the fact that the policy improvement step changes the factored policy $\pi = [\pi_1, \ldots, \pi_k]$ per block-wise update of the action variables in a finite space MDP (line 8).

global convergence of policy improvement If the underlying $Q_\pi(\mathbf{s}, \mathbf{a})$ is monotonic (Rashid et al. 2020), then the block-wise improvement is equivalent to the joint improvement, $Q_{\pi_k}\big(\mathbf{s}, \pi_k(\mathbf{a}_k)\big) > Q_{\pi_k}(\mathbf{s}, \mathbf{a}_k) \iff Q_\pi\big(\mathbf{s}, \pi_k(\mathbf{a})\big) > Q_\pi(\mathbf{s}, \mathbf{a})$. For any iteration of the policy improvement in the factored policy iteration algorithm, there exists an equivalent update in the policy iteration algorithm due to the monotonicity of the Q-function. Therefore, the policy improvement step reaches the same fixed point that would have reached by the non-factored policy iteration. □

Theorem 2 (Sample Complexity of MB-FPI)**.** Given $\delta \in (0, 1)$ and $\epsilon > 0$, the sample complexity for learning the *no-op* dynamics $P(\mathbf{S}'_{K+1}|\mathbf{S}, \text{Eff}(\mathbf{A}))$ and the intervention policy $\sigma_{\mathbf{A}_k}(\text{Pre}(\mathbf{A}_k)$ within error bound ϵ with at least probability $1 - \delta$ are $N_P \geq \frac{|\mathbf{S}_{K+1}||\mathbf{S}||\mathbf{S}\setminus\mathbf{S}_{K+1}|}{\epsilon^2} \log \frac{2|\mathbf{S}||\mathbf{S}\setminus\mathbf{S}_{K+1}|}{\delta}$ and $N_\sigma \geq \frac{|\text{Eff}(\mathbf{A}_k)||\text{Pre}(\mathbf{A}_k)|}{\epsilon^2} \log \frac{2|\text{Pre}(\mathbf{A}_k)|}{\delta}$, respectively.

Proof. Given K random variables $X^1, \ldots, X^K$ with the domain $X^j \in [0, 1]$, and N independent samples $X_1^j, X_2^j, \ldots, X_N^j$, let $S_N^j = \sum_{i=1}^N X_i^j$ and $\epsilon > 0$. Then,

$$P(|S_N^j - \mathbb{E}[S_N^j]/N| \geq \epsilon) \leq 2 \exp(-2N\epsilon^2),$$

by Hoeffding's inequality, and

$$P(\cup_{j=1}^K |S_N^j/N - \mathbb{E}[S_N^j]/N| \geq \epsilon) \leq \sum_{j=1}^K P(|S_N^j/N - \mathbb{E}[S_N^j]/N| \geq \epsilon) \leq 2K \exp(-2N\epsilon^2),$$

by union bound.

We can restate the above inequality as, with probability $1 - \delta$, $|S_N^j/N - \mathbb{E}[S_N^j]/N| \leq \epsilon$ for all $j = 1..K$ if $\epsilon = \sqrt{1/(2N) \log(2K/\delta)}$ given N and δ or $N \geq 1/(2\epsilon^2) \log(2K/\delta)$ given $\epsilon > 0$ and δ.

Given a discrete probability distribution $P(\mathbf{X}|\mathbf{Y})$, we can show the sample complexity of learning P in the tabular model-based RL setting (Agarwal et al. 2019), $N \geq |\mathbf{X}||\mathbf{Y}|/\epsilon^2 \cdot \log(2|\mathbf{Y}|/\delta)$. For all vectors $\mathbf{Y}$, we bound the error of the probability parameters by

$$P(\cup_{\mathbf{y}} |\hat{P}(X_i|\mathbf{y}) - P(X_i|\mathbf{y})| \geq \epsilon) \leq |\mathbf{Y}| P(|\hat{P}(X_i|\mathbf{y}) - P(X_i|\mathbf{y})| \geq \epsilon) \leq 2|\mathbf{Y}| \exp(-2N\epsilon^2).$$

For each X_i, the number of samples is at least $1/(2\epsilon^2) \log(2|\mathbf{Y}|/\delta)$, and we need $\mathbf{X}N$ samples for estimating all parameters is $N \geq |\mathbf{X}||\mathbf{Y}|/\epsilon^2 \log(2|\mathbf{Y}|)/\delta)$. The desired results can be obtained by adjusting the sets $\mathbf{X}$ and $\mathbf{Y}$ appropriately, i.e., $\mathbf{X} = \mathbf{S}'_{K+1}$ and $\mathbf{Y} = \mathbf{S} \cup \text{Eff}(\mathbf{A})$. □

B Experiment Details

B.1 Computing Resources

We used a cluster environment with 2 to 4 CPUs per each run.

B.2 2D Point-Mass Control

Algorithm Hyperparameters

Hyperparameters The following hyperparamters are chosen by grid search.

- Adam optimizer learning rate: $1e^{-4}$
- discount: 0.99
- target update frequency: 100 steps
- target network update rate (τ): 1.0
- no-op fraction probability: 0.1
- start ϵ: 1.0
- end ϵ: 0.1
- batch size: 128
- learning starts: 20 episodes
- model train data size: 200 episodes

Q-networks

- AD-DQN:

```
class ADDQN(nn.Module):
    def __init__(self, state_size, action_size, num_bins):
        super().__init__()
        self.action_size = action_size
        self.num_bins = num_bins
        self.q = nn.Sequential(
            nn.Linear(state_size, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, num_bins + num_bins),
        )
        self.mixer = nn.Sequential(
            nn.Linear(self.action_size, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )
        self.apply(self._init_weights)

    def forward(self, x, actions):
        actions = actions.reshape(-1, 2).to(torch.int64)
        x_mask = torch.nn.functional.one_hot(actions[:, 0], num_classes=self.num_bins)
        y_mask = torch.nn.functional.one_hot(actions[:, 1], num_classes=self.num_bins)
        action_mask = torch.cat([x_mask, y_mask], dim=1)
        z = self.q(x)
        z = z * action_mask
        z = self.mixer(z)
        return z
```

- DECQN:

```
class DecQNetwork(nn.Module):
    def __init__(self, state_size, action_size, num_bins):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(state_size, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, action_size),
            Reshape(-1, action_size//num_bins, num_bins)
        )

    def forward(self, x):
        return self.network(x)
```

Models

- Dynamics:

```
class DynamicsModelDelta2l(nn.Module):
    def __init__(self, state_size=2, output_size=1):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(state_size, 64),
            nn.Linear(64, 64),
            nn.Linear(64, output_size)
        )
        self.output_size = output_size
```

```
    def forward(self, x):
        return self.network(x)

    def evaluate(self, x, prev, noise_variance=0.0001):
        with torch.no_grad():
            output = self(x)
            noise = np.random.normal(0, np.sqrt(noise_variance))
            noise = torch.tensor([noise], dtype=torch.float32).to(output.device)
            return prev + output*(1 + noise)
```

- Rewards:

```
class RewardModel(nn.Module):
    def __init__(self, state_size):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(state_size*2 + 2, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, x):
        return self.network(x)
```

B.3 MIMIC-III Sepsis Treatment

Formulation of RL Environment

- The sepsis treatment environment (Komorowski et al. 2018) was derived from MIMIC-III database version 1.4 (Goldberger et al. 2000; Johnson et al. 2016).
- The sepsis treatment reinforcement learning environment from MIMIC-III database was first formulated by Komorowski et al. (2018). Killian et al. (2020) enhanced the formulation by improving the representation learning. Tang and Wiens (2021) studied an approach for offline RL evaluation for healthcare settings. Tang et al. (2022) showed overall evaluation protocol that we follow using the open source environment offered by Tang et al. (2022) [5].

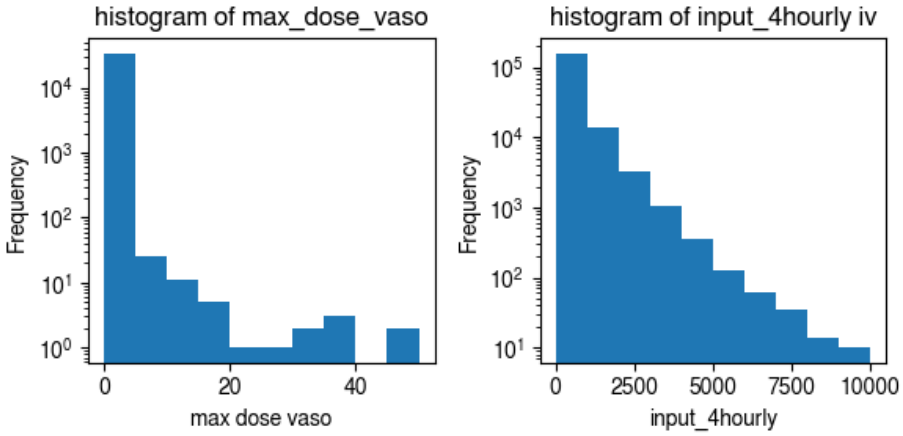
RL Environment

- We used the code offered by Killian et al. (2020) with the open sourced projects to process the MIMIC-III database [6].
- The original code was applied to MIMIC-III version 1.3 and we see that applying the same code to the version 1.4 results in slightly different statistics.
- Tang et al. (2022) modified action space of (Komorowski et al. 2018) and also modified the reward that the mortality gets 0 reward. We are reverting this change such that the reward is 100 for discharged patients, -100 for the mortality and 0 for all other patients.
- We obtained a cohort of 177,877 patients and applied 70/15/15 split for training, validation, and testing. This split is exactly the same as (Tang et al. 2022) since our baseline algorithms are BCQ algorithms in (Tang et al. 2022)
- Each patient has 5 non-time varying demographic features and 33 time varying features collected at 4-hour intervals over 72 hours that make up a discrete time-series of the maximum length 20. Some episodes are of length much less than 20 if the episode was terminated either by recovery or mortality. The index 0 of the episode is the starting state and the index 19 is the largest step number for the final state.
- The state encoding is done by the approximate information state encoder (AIS) (Subramanian et al. 2022), which gives 64 continuous state variables.
- The action space is comprised of two continuous variables for the total volume of intravenous fluid and the amount of vassopressors treated.
- We experimented on action spaces having different discretizations, ranging from 5x5, 10x10, 11x11, 13x13, and 14x14.

[5] https://github.com/MLD3/OfflineRL_FactoredActions

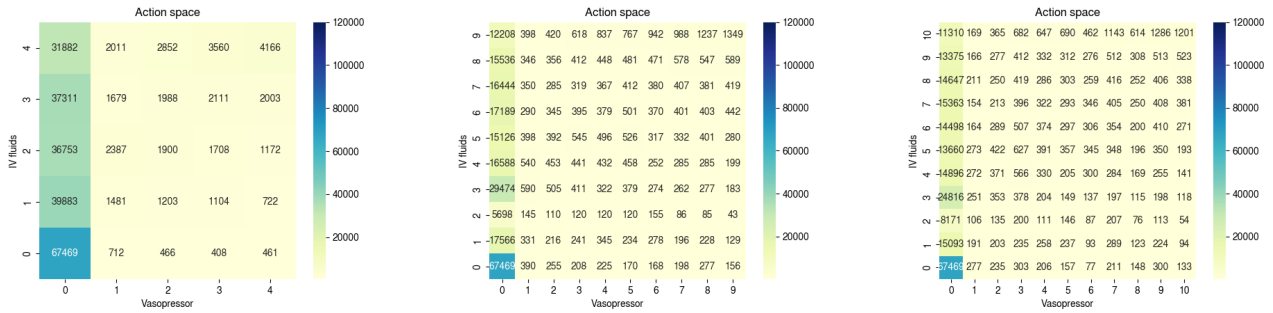
[6] https://github.com/MLforHealth/rl_representations/, https://github.com/microsoft/mimic_sepsis

Action Spaces Tang et al. (2022) modified the action space from the original space by Komorowski et al. (2018). We follow the discretization by Komorowski et al. (2018), by dividing the histogram below with the equally spaced quantiles.

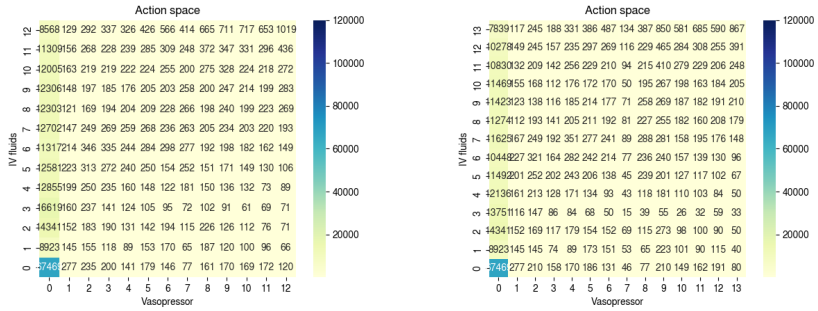


For example, the bins for the action space of size 5 is [0, 0.07, 0.2, 0.38, 50] for the vassopressor dose, and [0.02, 48.0, 150.0, 492.5, 9992.67] for the IV fluid dose. The binds for the action space of size 14 is [0.00, 0.03, 0.04, 0.06, 0.09, 0.12, 0.17, 0.20, 0.23, 0.30, 0.45, 0.56, 0.90, 50] for the vassopressor dose and 0.02, 20, 40, 40, 55.92, 80.08, 120, 186.25, 270, 380, 512.50, 728.98, 1110, 9992.67 for the IV fluids dose.

Applying finer grained discretization leads to larger action spaces as follows. The 0-th actions are the no-op action.



(a) Action space of 5x5 discretization (b) Action space of 10x10 discretization (c) Action space of 11x11 discretization



(d) Action space of 13x13 discretization (e) Action space of 14x14 discretization

Figure 4: Discretized action spaces ranging from 5x5 to 14x14

Evaluation Methods

WIS and ESS Let π be a policy under evaluation and π_b be a observed behavioral policy in the offline data. Given an episode $(s_1, a_1, r_1, \ldots, s_L, a_L, r_L)$ of length L, the per-step importance ratio is $\rho_t = \frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t}$ and the cumulative importance ratio is

$\rho_{1:t} = \prod_{t'=1}^{t} \rho_{t'}$. Given m episodes, the off-policy evaluation value from the weighted importance sampling can be computed as follows.

$$\hat{V}_{\text{WIS}}(\pi) = \frac{1}{m} \sum_{j=1}^{m} \frac{\rho^{(j)}_{1:L^{(j)}}}{w_{L^{(j)}}} \Big(\sum_{t=1}^{L^{(j)}} \gamma^{t-1} r_t^{(j)} \Big), \tag{16}$$

where $w_t = \frac{1}{m} \sum_{j=1}^{m} \rho^{(j)}_{1:t}$ is the average cumulative importance ratio at time step t. The superscript (j) indicates that the length $L^{(j)}$, rewards $r_t^{(j)}$, and the accumulated importance ratio $\rho^{(j)}_{1:t}$ are computed relative to the j-th episode. The effective sample size of the importance sampling can be computed as follows (Martino, Elvira, and Louzada 2016).

$$\text{ESS} = \frac{(\sum_{t=1}^{N} w_t)^2}{\sum_{t=1}^{N} w_t^2}, \tag{17}$$

where the weights are not normalized. While computing the weights, we clip the values by $1,000$ and softened π by mixing a random policy, namely $\hat{\pi}(a|s) = \mathbb{I}_{a=\pi(s)}(1-\epsilon) + \mathbb{I}_{a\neq\pi(s)}(\frac{\epsilon}{|\mathcal{A}|-1})$ with $\epsilon = 0.01$ following (Tang and Wiens 2021).

- The offline dataset was split into the train, validation, and test sets with 70, 15, and 15 proportions (12989, 2779, and 2791 patients).
- Given the dataset, we extract the observed policy (behavior policy) using the k nearest neighbor classifier implemented in (Pedregosa et al. 2011), and we used the default parameter except for $K = 100$. The stochastic observed policy can be found by predicting the probability of selecting each action in each state encoded by AIS.
- The observed values (the average discounted accumulated rewards) are 87.74, 87.98, and 87,75 for the train, validation, and test set. Since this data set has only a single reward at the end of each episode and the discounting factor is 1.0 for the evaluation, the value is the average of the rewards of episodes.
- For all hyperparameters swept and random trials, we select the best model with the highest validation WIS given ESS cutoff value. Using the selected model, we evaluate its test WIS and test ESS for the final comparison.

Algorithm Hyperparameters

BCQ hyperparameters

- Adam optimizer learning rate: $3e^{-4}$
- weight decay: $1e^{-3}$
- discount: 0.99
- target update frequency: 1
- Q-learning learning rate (τ): 0.005
- Q-learning target update: Polyak update
- BCQ thresholds: $[0.0, 0.01, 0.05, 0.1, 0.3, 0.5, 0.75, 0.9999]$

Q-networks

- BCQ: state-dim=64, hidden-dim=128, action-dim=$[5^2, 10^2, 11^2, 13^2, 14^2]$

```
class BCQ_Net(nn.Module):
    def __init__(self, state_dim, action_dim, hidden_dim):
        super().__init__()
        self.q = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim),
        )
        self.pi_b = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim),
        )
```

```
    def forward(self, x):
        q_values = self.q(x)
        p_logits = self.pi_b(x)
        return q_values, F.log_softmax(p_logits, dim=1), p_logits
```

- BCQ factored: state-dim=64, hidden-dim=128, action-dim=$[5 \cdot 2, 10 \cdot 2, 11 \cdot 2, 13 \cdot 2, 14 \cdot 2]$

```
class BCQf_Net(nn.Module):
    def __init__(self, state_dim, action_dim, hidden_dim):
        super().__init__()
        self.q = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim), # vaso + iv
        )
        self.pi_b = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim), # vaso + iv
        )

    def forward(self, x):
        q_values = self.q(x)
        p_logits = self.pi_b(x)
        return q_values, F.log_softmax(p_logits, dim=-1), p_logits
```

- Action Decomposed BCQ: state-dim=64, hidden-dim=128, action-dim=$[5, 10, 11, 13, 14]$

```
class BCQad_Net(nn.Module):
    def __init__(self, state_dim, action_dim, hidden_dim=64):
        super().__init__()

        self.q_embedding = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            # nn.ReLU(),
            # nn.Linear(hidden_dim, hidden_dim),
        )
        self.Q1 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim)
        )
        self.Q2 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim)
        )
        self.Q = nn.Sequential(
            nn.Linear(action_dim + action_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
```

```
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim+action_dim)
        )
        self.pi_embedding = nn.Sequential(
            nn.Linear(state_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            # nn.ReLU(),
            # nn.Linear(hidden_dim, hidden_dim),
        )
        self.pi1 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim)
        )
        self.pi2 = nn.Sequential(
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim)
        )
        self.pi = nn.Sequential(
            nn.Linear(action_dim + action_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, action_dim+action_dim)
        )

    def forward(self, x, mode):
        if mode == "vaso":
            q = self.q_embedding(x)
            q_values = self.Q1(q)
            p = self.pi_embedding(x)
            p_logits = self.pi1(p)
            return q_values, F.log_softmax(p_logits, dim=-1), p_logits

        if mode == "iv":
            q = self.q_embedding(x)
            q_values = self.Q2(q)
            p = self.pi_embedding(x)
            p_logits = self.pi2(p)
            return q_values, F.log_softmax(p_logits, dim=-1), p_logits

        if mode == "mix":
            with torch.no_grad():
                q = self.q_embedding(x)
                q_values_1 = self.Q1(q)
                q_values_2 = self.Q2(q)
            q_values = self.Q(torch.cat([q_values_1, q_values_2], dim=1))

            with torch.no_grad():
                p = self.pi_embedding(x)
                p_logits1 = self.pi1(p)
                p_logits2 = self.pi2(p)
            p_logits = self.pi(torch.cat([p_logits1, p_logits2], dim=1))
```

```
        return q_values, F.log_softmax(p_logits, dim=-1), p_logits

    if mode == "eval":
        with torch.no_grad():
            q = self.q_embedding(x)
            q_values_1 = self.Q1(q)
            q_values_2 = self.Q2(q)
            q_values = self.Q(torch.cat([q_values_1, q_values_2], dim=1))

            p = self.pi_embedding(x)
            p_logits1 = self.pi1(p)
            p_logits2 = self.pi2(p)
            p_logits = self.pi(torch.cat([p_logits1, p_logits2], dim=1))
        return q_values, F.log_softmax(p_logits, dim=-1), p_logits
```