
BraVE: Offline Reinforcement Learning for Discrete Combinatorial Action Spaces

Matthew Landers^{1*}, Taylor W. Killian², Hugo Barnes¹, Thomas Hartvigsen¹, Afsaneh Doryab¹

¹University of Virginia, ²MBZUAI

Abstract

Offline reinforcement learning in high-dimensional, discrete action spaces is challenging due to the exponential scaling of the joint action space with the number of sub-actions and the complexity of modeling sub-action dependencies. Existing methods either exhaustively evaluate the action space, making them computationally infeasible, or factorize Q-values, failing to represent joint sub-action effects. We propose **Branch Value Estimation (BraVE)**, a value-based method that uses tree-structured action traversal to evaluate a linear number of joint actions while preserving dependency structure. BraVE outperforms prior offline RL methods by up to $20\times$ in environments with over four million actions.²

1 Introduction

Offline reinforcement learning (RL) enables agents to learn decision-making policies from fixed datasets, avoiding the risks and costs inherent in online exploration [21, 22]. Existing methods have shown strong performance in low-dimensional discrete settings and continuous control tasks [1, 12, 13, 15, 19, 20]. However, many real-world decisions require selecting actions from high-dimensional, discrete spaces. In healthcare, for example, practitioners must choose among thousands of possible combinations of procedures, medications, and tests at each decision point [33].

Such settings give rise to *combinatorial* action spaces, where the number of actions grows exponentially with dimensionality, scaling as $\prod_{d=1}^N m_d$ for N sub-action dimensions. In offline reinforcement learning, this structure induces two primary challenges: evaluating and optimizing over a large, discrete action space is computationally demanding; and making accurate decisions requires modeling sub-action dependencies from fixed datasets. Standard value-based methods can, in principle, capture sub-action dependencies but require evaluating or maximizing the Q-function over the full action space, which is intractable in high-dimensional settings. Conversely, recent approaches [3, 27] factorize the Q-function under a conditional independence assumption, reducing computational cost but limiting expressivity by precluding the modeling of sub-action interactions, which are critical in many real-world domains.

We introduce **Branch Value Estimation (BraVE)**, a value-based offline RL method for combinatorial action spaces that avoids the scalability–expressivity tradeoff of prior approaches. BraVE imposes a tree structure over the action space and uses a neural network to guide traversal, evaluating only a linear number of candidate actions while preserving sub-action dependencies. To support this structured selection process, the Q-function is trained with a behavior-regularized temporal difference (TD) loss and a branch value propagation mechanism.

Our experiments show that BraVE consistently outperforms state-of-the-art baselines across a suite of challenging offline RL tasks with combinatorial action spaces containing up to 4 million discrete

*mlanders@virginia.edu

²Code is available at <https://github.com/matthewlanders/BraVE>

actions. In high-dimensional environments with strong sub-action dependencies, **BraVE improves average return by up to $20\times$ over state-of-the-art offline RL methods**. While baseline performance degrades with increasing sub-action dependencies or action space size, BraVE maintains stable performance.

Our contributions are as follows:

1. We propose a behavior-regularized TD loss that captures sub-action dependencies by evaluating complete actions rather than marginal components in combinatorial spaces.
2. We introduce **BraVE**, an offline RL method for discrete combinatorial action spaces that captures sub-action interactions and scales to high-dimensional settings via Q-guided traversal over a tree-structured action space.
3. We empirically show that BraVE outperforms state-of-the-art baselines in environments with combinatorial action spaces, maintaining high returns and stable learning as both action space size and sub-action dependencies increase.

2 Preliminaries

Reinforcement learning problems can be formalized as a Markov Decision Process (MDP), $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma, \mu \rangle$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a function that gives the probability of transitioning to state s' when action a is taken in state s , $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function, $\gamma \in [0, 1]$ is a discount factor, and $\mu : \mathcal{S} \rightarrow [0, 1]$ is the distribution of initial states. A policy $\pi : \mathcal{S} \rightarrow \mathbb{P}(\mathcal{A})$ is a distribution over actions conditioned on a state $\pi(a | s) = \mathbb{P}[a_t = a | s_t = s]$.

While the standard MDP formulation abstracts away the structure of actions in $\mathcal{A}$, we explicitly assume that the action space is combinatorial; that is, $\mathcal{A}$ is defined as a Cartesian product of sub-action spaces. More formally, $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \dots \times \mathcal{A}_N$, where each $\mathcal{A}_d$ is a discrete set. Consequently, $\mathbf{a}_t$ is an N -dimensional vector wherein each component is referred to as a sub-action.

The agent’s goal is to learn a policy π^* that maximizes cumulative discounted returns:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^H \gamma^t r(s_t, a_t) \right],$$

where $s_0 \sim \mu$, $a_t \sim \pi(\cdot | s_t)$, and $s_{t+1} \sim p(\cdot | s_t, a_t)$.

In offline RL, the agent learns from a static dataset of transitions $\mathcal{B} = \{(s_t, a_t, r_t, s_{t+1})^i\}_{i=0}^Z$ generated by, possibly, a mixture of policies collectively referred to as the behavior policy π_{β} .

Like many recent offline RL methods, our work uses approximate dynamic programming to minimize temporal difference error (TD error) starting from the following loss function:

$$L(\theta) = \mathbb{E}_{\mathcal{B}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right], \quad (1)$$

where the expectation is taken over transitions (s, a, r, s') sampled from the replay buffer $\mathcal{B}$, $Q(s, a; \theta)$ is a parameterized Q-function that estimates the expected return when taking action a in state s and following the policy π thereafter, and $Q(s, a; \theta^-)$ is a target network with parameters θ^- , which is used to stabilize learning.

For out-of-distribution actions a' , Q-values can be inaccurate, often causing overestimation errors due to the maximization in Equation (1). To mitigate this effect, offline RL methods either assign lower values to these out-of-distribution actions via regularization or directly constrain the learned policy. For example, TD3+BC [13] adds a behavior cloning term to the standard TD3 [14] loss:

$$\pi = \arg \max_{\pi} \mathbb{E}_{(s,a) \sim \mathcal{B}} [\lambda Q(s, \pi(s)) - (\pi(s) - a)^2], \quad (2)$$

where λ is a scaling factor that controls the strength of the regularization.

More recently, implicit Q-learning (IQL) [19] used a SARSA-style TD backup and expectile loss to perform multi-step dynamic programming without evaluating out-of-sample actions:

$$L(\theta) = \mathbb{E}_{\mathcal{B}} \left[\left(r + \gamma \max_{a' \in \Omega(s)} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right], \quad (3)$$

where $\Omega(s) = \{a \in \mathcal{A} | \pi_{\beta}(a | s) > 0\}$ are actions in the support of the data.

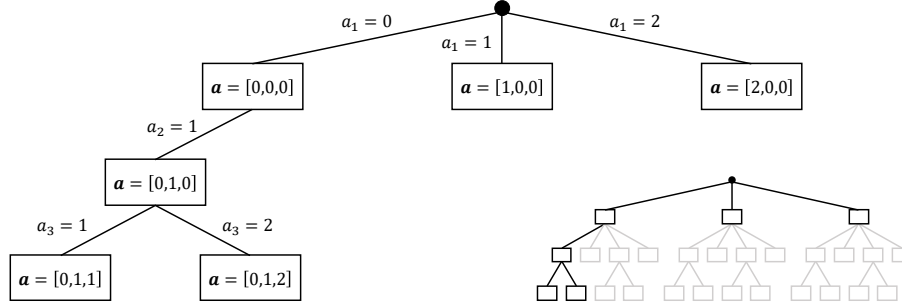


Figure 1: BraVE’s tree representation for a 3-dimensional combinatorial action $\mathbf{a} = [a_1, a_2, a_3]$, where each sub-action $a_i \in \{0, 1, 2\}$. Each node encodes a complete action vector, with explicitly chosen sub-actions set according to the traversal path from the root, and all remaining dimensions filled with a default value (here, 0). At depth k , the value of sub-action a_k is selected, with sibling nodes differing only in that dimension.

3 Branch Value Estimation (BraVE)

The Q-function, $Q(s, \mathbf{a})$ in Equation (1), encodes long-term value by mapping each state–action pair to its expected return. When actions comprise multiple sub-actions $\mathbf{a} = [a_1, \dots, a_N]$, $Q(s, \mathbf{a})$ implicitly captures sub-action dependencies by accounting for interactions that influence both immediate rewards and future transitions, making it a natural target for learning in structured or combinatorial action spaces. However, in combinatorial action spaces, the exponential growth of joint actions renders both forward passes and Q-value estimation computationally intractable [30, 34].

Factorizing the Q-function across action dimensions [3, 27] is a common strategy for improving tractability. While this reduces computational cost, it sacrifices expressivity by ignoring sub-action dependencies, leading to estimation errors when sub-actions jointly influence transitions or rewards. In such settings, including real-world domains like treatment recommendation, summing marginal Q-values can produce divergent estimates (Appendix A).

We introduce **Branch Value Estimation (BraVE)**, an offline RL method that avoids this scalability–expressivity tradeoff. BraVE assigns values to complete action vectors $Q(s, \mathbf{a})$, preserving their interdependencies, but avoids exhaustively scoring every action by imposing a tree structure over the space and using a neural network to guide traversal that selectively evaluates a small subset of actions. This results in only a linear number of evaluations per decision, enabling BraVE to scale to high-dimensional combinatorial settings without sacrificing value function fidelity.

3.1 Tree Construction

BraVE imposes a tree structure over the combinatorial action space, assigning a complete N -dimensional action vector to each node. This vector is constructed by combining the sub-action values explicitly selected along the path from the root with predefined default values (e.g., $a_i = 0$ in the binary case) for all unassigned dimensions.

The tree originates from a root node corresponding to the start of traversal, where no sub-actions have been selected. From this root, branches extend to child nodes by explicitly assigning a value $a_k \in \mathcal{A}_k$ to the k -th sub-action dimension d_k . A node at depth k therefore represents a complete action vector in which the first k sub-actions ($a_1, \dots, a_k$) have been explicitly set, and the remaining sub-actions ($a_{k+1}, \dots, a_N$) retain their default values. Sibling nodes at depth k differ from their parent only in the newly assigned component, and differ from each other only in their choice of a_k . That is, they differ only in the k -th component of the complete action vector they represent, as illustrated in Figure 1. This dimension-by-dimension construction ensures that reaching a leaf node, where all N sub-action values have been explicitly assigned, requires traversing at most N levels.

While this specific N -depth, dimension-ordered tree is adopted for its computational efficiency, the general BraVE framework places no constraints on the tree structure. Any tree in which each node represents a complete joint action, and in which root-to-leaf paths systematically cover the action space, is a valid instantiation. For example, a flat tree with a single level and $|\mathcal{A}|$ children — one for each possible joint action — recovers the naive alternative of exhaustively evaluating

$Q(s, \mathbf{a})$ over the entire space. Our implementation, however, deliberately uses the deeper, N -level structure to guarantee a number of node evaluations linear in N , which is crucial for tractability in high-dimensional spaces.

This structural flexibility extends beyond tree depth and layout. The ordering of sub-actions, the choice of default values, and the tree topology are all implementation details rather than core components of the method. Regardless of the selected configuration, however, the chosen structure should remain fixed throughout training to ensure consistent semantics for traversal and value estimation. The tree itself may be constructed dynamically at each timestep or precomputed and cached.

BraVE imposes no restrictions on the cardinality of individual sub-actions; each a_d may be drawn from an arbitrary discrete set $\mathcal{A}_d$. For clarity, however, all examples and figures that follow assume a multi-binary action space, where each sub-action is either included ($a_i = 1$) or excluded ($a_i = 0$).

3.2 Tree Sparsification

To mitigate overestimation error and stabilize learning, BraVE introduces an inductive bias by **sparsifying the action space tree**, restricting it to include only actions observed in the dataset $\mathcal{B}$. This idea is conceptually related to BCQ [15], which constrains policy evaluation to actions deemed plausible by a learned generative model. BraVE, by contrast, enforces this constraint structurally by pruning branches from the tree that correspond to implausible or unsupported sub-action combinations.

This design is particularly effective in real-world settings where certain sub-actions are incompatible and rarely, if ever, appear together. For example, in treatment planning or medication recommendation, specific drug combinations may be avoided due to known adverse interactions. By limiting the tree to feasible actions observed in the data, BraVE avoids evaluating unrealistic behaviors, reduces computational overhead, and curbs value inflation driven by unsupported extrapolation.

3.3 Q-Guided Tree Traversal

To compute the greedy action $\hat{\mathbf{a}}' = \arg \max_{\mathbf{a}'} Q(s', \mathbf{a}'; \theta^-)$ required by Equation (1), BraVE traverses the action space tree using a neural network $f(\cdot, \cdot; \theta^-)$ to guide decisions at each node. At a node $\mathbf{a}_{\text{node}}$, the network receives as input the current state s , which may be discrete or continuous, and the N -dimensional action vector corresponding to $\mathbf{a}_{\text{node}}$. It outputs a scalar node Q-value $q = Q(s, \mathbf{a}_{\text{node}}; \theta^-)$ and a vector of *branch values* $\mathbf{v} \in \mathbb{R}^{M_{\text{children}}^{\max}}$. While q reflects the value of selecting the current node’s complete action, the branch values $\mathbf{v}$ estimate the maximum Q-value obtainable in the subtree rooted at the j -th child. The network produces both q and $\mathbf{v}$ in a single forward pass, with the branch values $\mathbf{v}$ representing a learned estimate of each subtree’s potential.

Importantly, the network f is a standard MLP requiring no specialized architecture. The tree is not encoded into the network input; instead, it is an external structure that organizes the $\arg \max$ computation.

Though the number of children can vary across nodes, the network output dimensionality remains fixed. Specifically, f returns one scalar for the current node and $M_{\text{children}}^{\max} = \max_d |\mathcal{A}_d|$ branch values. For example, if the maximum sub-action cardinality across all dimensions is three, the network produces four outputs total. Following standard practice in structured prediction and reinforcement learning [18, 26] masking is applied to ignore unused entries in $\mathbf{v}$ for nodes with fewer than $M_{\text{children}}^{\max}$ children.

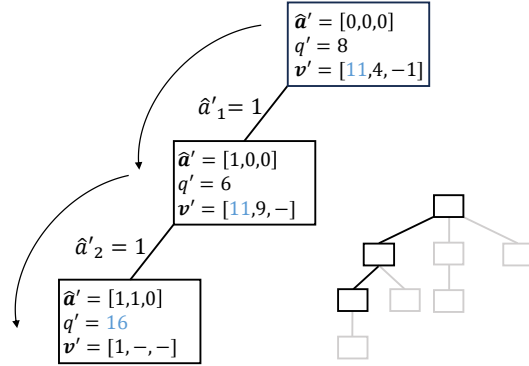


Figure 2: BraVE traversal in a 3-D binary action space (full tree shown bottom-right). Starting from the root $[0, 0, 0]$, the agent selects $\hat{a}'_1 = 1$ since its branch value (11) exceeds both those of alternative children (4, -1) and the root’s Q-value (8). Traversal proceeds until reaching $[1, 1, 0]$, where the Q-value (16) exceeds the child’s branch value (1); a terminal condition. Masked values ($-$) are ignored.

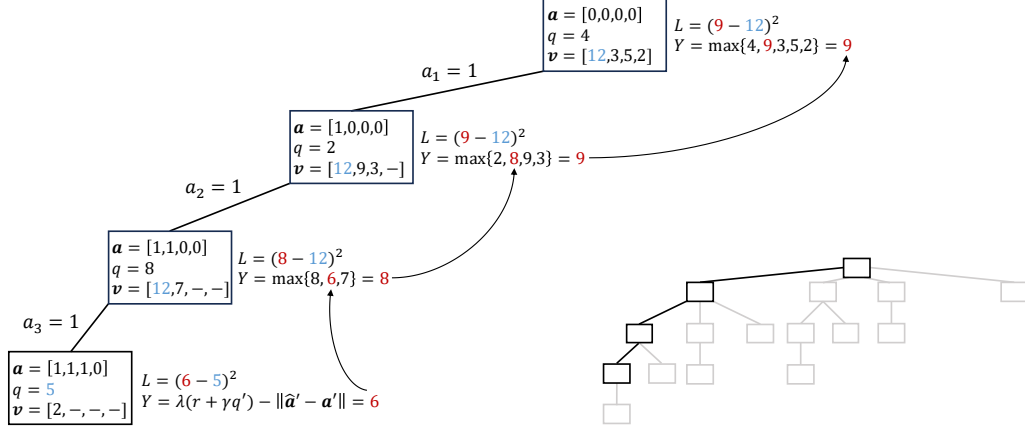


Figure 3: Example of loss propagation in a 4-D binary action space (full tree shown bottom-right). Starting from the node $[1, 1, 1, 0]$ (bottom left), the target (Equation 5) is propagated to its parent $[1, 1, 0, 0]$. The new target is computed as the maximum of the propagated value, the parent’s own Q-value, and the branch values of alternative child nodes. This process recurses up the tree to compute all node losses.

Traversal begins at the root node. At each step, the network evaluates $(q, \mathbf{v})$ at the current node. If $q \geq \max_j v_j$, traversal terminates and the corresponding action $\mathbf{a}_{\text{node}}$ is returned as $\hat{\mathbf{a}}'$. Otherwise, the algorithm proceeds to the child indexed by $\arg \max_j v_j$. This greedy descent continues until either the termination condition is satisfied or a leaf node is reached. Notably, traversal may terminate at internal nodes rather than always proceeding to a fully specified leaf. A full example of this process is shown in Figure 2.

This traversal mechanism enables BraVE’s computational efficiency: it evaluates a single node per sub-action dimension, resulting in $\mathcal{O}(N)$ complexity — a factor of $\prod_d |A_d|/N$ fewer evaluations than exhaustive scoring. The tree serves as the structural scaffold that makes this targeted, linear-time traversal possible.

3.4 Loss Computation

BraVE is trained using a behavior-regularized temporal difference (TD) loss that penalizes value estimates for actions unlikely under the dataset. Given a transition $(s, \mathbf{a}, r, s', \mathbf{a}')$ sampled from the replay buffer $\mathcal{B}$, the TD target is computed using the action $\hat{\mathbf{a}}' = \arg \max_{\mathbf{a}'} Q(s', \mathbf{a}'; \theta^-)$ selected via the tree traversal procedure described in Section 3.3. The loss is defined as:

$$L_{TD}(\theta) = \mathbb{E}_{(s, \mathbf{a}, r, s', \mathbf{a}') \sim \mathcal{B}} \left[\left(\lambda (r + \gamma Q(s', \hat{\mathbf{a}}'; \theta^-)) - \|\hat{\mathbf{a}}' - \mathbf{a}'\| - Q(s, \mathbf{a}; \theta) \right)^2 \right], \quad (4)$$

where λ is a regularization coefficient and $\|\hat{\mathbf{a}}' - \mathbf{a}'\|$ penalizes deviation from the behavior action, following principles introduced in TD3+BC (Equation 2).

We combine this behavior-regularized TD loss with a branch value supervision loss L_{BraVE} , resulting in a total objective $L = \alpha L_{TD} + L_{\text{BraVE}}$, where α controls the relative weighting of the TD term. To compute L_{BraVE} , we begin from a node $\mathbf{a}$ sampled from $\mathcal{B}$ and reuse the TD target

$$Y = \lambda (r + \gamma Q(s', \hat{\mathbf{a}}'; \theta^-)) - \|\hat{\mathbf{a}}' - \mathbf{a}'\| \quad (5)$$

as the supervisory signal. This target is then propagated recursively up the tree. At each step, the target is used to update the parent node’s estimate for the corresponding branch value. The propagated target is updated at each level to reflect the maximum of Y and the existing sibling branch values, ensuring consistency with the max-based traversal logic. This process continues up to the root, training the branch values $\mathbf{v}$ at each node through direct supervision from L_{BraVE} to estimate the highest Q-value accessible through its subtree. The full L_{BraVE} computation procedure is provided in Algorithm 1 and illustrated in Figure 3.

The recursive mechanism underlying L_{BraVE} stabilizes learning by propagating training signals not only to the node corresponding to the sampled action but also to every ancestor on its path. Consequently, internal nodes — including those not directly sampled — receive gradient updates. Because all branches share a single global Q-network, these updates generalize across similar states and actions, enabling BraVE to form reliable estimates even in rarely-sampled regions of the tree.

3.5 Enhancements for Stability and Policy Quality

We mitigate the sensitivity of BraVE’s hierarchical structure to inaccurate branch value estimates using two complementary mechanisms: a **depth penalty** applied *during training* and **beam search** used *during inference*. During training, we apply a depth-based weighting factor δ in the BraVE loss computation. Because the branch value target is propagated upward from a sampled action node to the root, this weighting amplifies the influence of errors closer to the root, where mistakes propagate to more of the tree. At inference time, we use beam search to improve action selection robustness. Instead of committing to a single greedy path, the algorithm retains the top- W actions (ranked by predicted values) at each tree level. The final action is selected from the union of all beams, allowing for broader exploration of high-value combinations and improving policy quality.

4 Experimental Evaluation

We evaluate BraVE in the **Combinatorial Navigation Environment (CoNE)**, a high-dimensional discrete control domain designed to stress-test policy learning under large action spaces and sub-action dependencies. In CoNE, actions are formed by selecting subsets of atomic motion primitives (sub-actions), corresponding to directional moves along orthogonal axes. The agent chooses which primitives to activate at each timestep, resulting in a combinatorial action space of size $|\mathcal{A}| = 2^{2D}$, where D is the number of spatial dimensions.

The agent starts from a fixed origin s_0 and must reach a designated goal g . At each timestep, it receives a negative reward $r = -\rho(s, g)$ proportional to its Euclidean distance from the goal. Episodes terminate upon reaching the goal (with reward +10) or falling into a pit, a failure state that incurs a penalty of $r = -10 \cdot \rho(s_0, g)$. This penalty structure ensures that failure is strictly worse than any successful trajectory, even those that are long or indirect.

CoNE exhibits both combinatorial complexity and tightly coupled action dynamics. Some sub-action combinations are synergistic, such as activating orthogonal directions to move diagonally, opposing movements cancel out, and others lead to failure by directing the agent into hazardous regions of the state space. These structured dependencies mean that the effectiveness of a sub-action often depends critically on the presence or absence of others.

This stands in contrast to discretized variants of popular continuous control environments [28, 31], where sub-actions require coordination but can be learned independently [3]. In CoNE, by contrast, sub-actions that are individually beneficial can be harmful in combination, an effect common in real-world settings such as drug prescription, where treatments may interact antagonistically (see Appendix A).

Dataset Construction We generate offline datasets using a stochastic variant of A*. At each step, the optimal action is selected with probability 0.1, and a random valid action is chosen otherwise. This procedure yields a diverse mixture of trajectories with varying returns, including both near-optimal and suboptimal behavior. The resulting datasets reflect realistic offline settings in which learning must proceed from heterogeneous, partially optimal demonstrations.

Baselines We compare BraVE to two representative baselines: (1) Factored Action Spaces (FAS) [3, 27], the state-of-the-art for offline RL in combinatorial action spaces; and (2) Implicit Q-Learning

Algorithm 1 Compute BraVE Loss

Require:

$f(\theta)$: neural network with parameters θ
 $f(\theta^-)$: target network with parameters θ^-
 $\{s, \mathbf{a}, r, s', \mathbf{a}'\}$: transition from $\mathcal{B}$
 $\hat{\mathbf{a}}'$: action selected via tree traversal given s

- 1: $(q, \mathbf{v}) \leftarrow f(s, \mathbf{a}; \theta)$
- 2: $(q', \mathbf{v}') \leftarrow f(s', \hat{\mathbf{a}}'; \theta^-)$
- 3: $Y \leftarrow \lambda(r + \gamma q') - \|\hat{\mathbf{a}}' - \mathbf{a}'\|$
- 4: total loss $\leftarrow (q - Y)^2$
- 5: node $\leftarrow \mathbf{a}$
- 6: $d \leftarrow 1$
- 7: **while** node is not *null* **do**
- 8: parent $\leftarrow \text{GETPARENT}(\text{node})$
- 9: $q, \mathbf{v} \leftarrow f(s, \text{parent}; \theta)$
- 10: children $\leftarrow \text{GETCHILDREN}(\text{parent})$
- 11: $i \leftarrow \text{index of node in children}$
- 12: loss $\leftarrow ((\mathbf{v}[i] - Y) * \delta d)^2$
- 13: total loss $\leftarrow \text{total loss} + \text{loss}$
- 14: $\mathbf{v}[i] \leftarrow Y$
- 15: $Y \leftarrow \max(q, \mathbf{v})$
- 16: node $\leftarrow \text{parent}$
- 17: $d \leftarrow d + 1$
- 18: **end while**
- 19: **return** total loss/ d

$ \mathcal{A} $	BraVE	FAS	IQL
16	1.5 ± 0.0	1.5 ± 0.0	-0.4 ± 1.5
64	-0.4 ± 0.0	-0.4 ± 0.0	-6.1 ± 3.2
256	-2.0 ± 0.1	-2.3 ± 0.6	-9.8 ± 4.5
1024	-3.4 ± 0.1	-10.8 ± 2.3	-13.1 ± 5.8
4096	-6.9 ± 1.6	-7.3 ± 3.1	-13.5 ± 5.7
$\sim 16k$	-6.1 ± 0.4	-25.6 ± 33.3	-15.4 ± 6.0
$\sim 65k$	-8.2 ± 2.2	-24.4 ± 10.4	-27.0 ± 11.5
$\sim 260k$	-13.8 ± 5.4	-42.2 ± 32.3	-48.4 ± 17.7
$\sim 1M$	-9.6 ± 1.2	-21.4 ± 18.2	-53.7 ± 31.0
$\sim 4M$	-18.6 ± 8.3	-33.9 ± 27.0	-66.9 ± 31.6

Table 1: In environments with non-factorizable reward structures and no sub-action dependencies BraVE and FAS perform similarly in low-dimensional settings, but as action dimensionality increases, BraVE maintains high returns and stable policies, while FAS’s performance deteriorates. IQL performs worst in all settings.

(IQL) [19], a strong general-purpose algorithm included to assess the need for methods purpose-built for combinatorial structures. Following prior work [27], we implement FAS using a factored variant of BCQ [15]. All methods are trained for 20,000 gradient steps and evaluated every 100 steps.

4.1 Performance Across Reward Structures and Sub-Action Dependencies

FAS has shown strong performance in offline RL for combinatorial domains [3, 27], however, its effectiveness depends on the assumption that rewards decompose cleanly across sub-actions and that dependencies among sub-actions are weak. These assumptions are often violated in real-world settings, where the effect of one sub-action may depend critically on the value of another. To evaluate BraVE in precisely these challenging conditions, we construct a suite of CoNE environments that systematically vary along both axes of complexity.

We measure performance across 10 randomly generated environments in each of two settings: the first includes non-factorizable reward structures without sub-action dependencies, and the second includes both non-factorizable rewards and dependent sub-actions. For each task, we vary dimensionality from 2-D ($|\mathcal{A}| = 16$) to 11-D ($|\mathcal{A}| > 4$ million), where each additional dimension introduces two new sub-actions. Each environment has size 5 in every dimension. The agent starts in the top-left corner and must reach a goal in the bottom-right. Results are averaged over five random seeds.

Non-Factorizable Reward Structures Only We begin with environments that contain no pits. In these settings, the transition dynamics are factorizable across sub-actions. However, the reward function cannot be linearly decomposed. As shown in Table 1, FAS performs comparably to BraVE in low-dimensional tasks but degrades as dimensionality increases. Because the reward depends on the state induced by the full action vector, the FAS approach of assigning rewards to individual sub-actions introduces a modeling bias that grows with action dimensionality, causing its learned Q-values to diverge from the true values. BraVE avoids this instability by evaluating complete joint actions directly, maintaining high performance across all action-space sizes. IQL performs significantly worse than BraVE across all action space sizes, suggesting that methods not explicitly designed for combinatorial actions struggle even when transitions are factorizable. Complete learning curves are provided in Appendix B.1.

Non-Factorizable Rewards and Sub-Action Dependencies We next evaluate performance under increasing sub-action dependency induced by hazardous transitions. In an 8-D environment ($|\mathcal{A}| = 65,536$), we vary the number of pits from 5% to 100% of the 6,561 **interior** (non-boundary) states, ensuring that there is always a feasible path to the goal along the boundary of the state space (where pits are never placed). As shown in Table 2, BraVE maintains stable performance across all pit densities. FAS, by contrast, degrades sharply once pits occupy half of the interior, and IQL declines steadily as dependency strength increases. These results highlight BraVE’s ability to handle tightly coupled decision-making, even when adverse interactions between sub-actions are common. These

Pit %	BraVE	FAS	IQL
0	-8.2 $\pm$ 2.2	-24.4 $\pm$ 10.4	-27.0 $\pm$ 11.5
5	-19.0 $\pm$ 2.3	-78.4 $\pm$ 99.7	-85.8 $\pm$ 45.0
10	-16.9 $\pm$ 3.3	-140.4 $\pm$ 177.1	-88.1 $\pm$ 50.1
25	-42.9 $\pm$ 43.3	-178.6 $\pm$ 212.8	-82.8 $\pm$ 52.0
50	-54.8 $\pm$ 47.9	-902.6 $\pm$ 274.1	-106.4 $\pm$ 42.8
75	-42.9 $\pm$ 34.3	-942.8 $\pm$ 278.1	-110.0 $\pm$ 34.3
100	-41.6 $\pm$ 22.4	-1131.4 $\pm$ 0.0	-112.4 $\pm$ 22.8

Table 2: BraVE maintains stable performance across environments with non-factorizable rewards and sub-action dependencies, whereas FAS exhibits a sharp decline when pits occupy at least half of the interior states. IQL performance also degrades as sub-action dependencies increase.

results are further contextualized by the normalized episode-length score reported in Appendix B.2, where complete learning curves are also provided.

Sparse but Critical Sub-Action Dependencies Having established that BraVE performs well under strong sub-action coupling, we examine settings with only five pits, representing a negligible fraction of the state space in high dimensions. We continue scaling the state space until BraVE and FAS converge to similar average returns, allowing us to assess how each method handles **sparse but critical dependencies**. As shown in Table 3, BraVE performs reliably even when the state space is small and each action carries substantial risk. FAS, by contrast, requires a much larger state space before the influence of the pits becomes diluted and its performance approaches that of BraVE. This demonstrates that even a small number of pits is sufficient to invalidate naive factorizations. Complete learning curves are provided in Appendix B.3.

$ \mathcal{S} $	BraVE	FAS	IQL
25	-7.5 $\pm$ 2.4	-531.5 $\pm$ 31.4	-12.1 $\pm$ 5.8
125	-2.9 $\pm$ 0.2	-579.8 $\pm$ 7.2	-14.1 $\pm$ 13.6
625	-5.6 $\pm$ 1.5	-480.3 $\pm$ 152.9	-22.4 $\pm$ 20.6
3125	-6.3 $\pm$ 0.7	-147.4 $\pm$ 292.6	-28.0 $\pm$ 22.8
15625	-12.4 $\pm$ 7.9	-18.9 $\pm$ 4.6	-37.8 $\pm$ 17.4

Table 3: When sub-action dependencies are sparse but critical, BraVE performs reliably even in small state spaces with high-risk actions. FAS approaches BraVE’s performance only once the joint action space exceeds 15 thousand. IQL performs worse than BraVE in all settings.

4.2 Robustness to Action Order

BraVE’s tree structure is defined by a fixed sequence of sub-action dimensions. To assess sensitivity to action ordering, we trained BraVE, FAS, and IQL using five random permutations of this sequence in the 8-D, 50% pit CoNE setting, which is empirically the most challenging 8-D variant (see Table 2). Results averaged over these permutations are reported in Table 4.

These findings are consistent with those in Table 2, showing that BraVE outperforms FAS by an order of magnitude and is roughly twice as effective as IQL. This demonstrates that BraVE’s performance is robust to the ordering of sub-action dimensions and does not rely on a favorable structural arrangement.

4.3 Online Fine-Tuning

BraVE’s data-driven tree sparsification (Section 3.2) inherently constrains action selection to observed behaviors, a principle shared by offline methods that model the behavior policy using generative approaches [15]. However, BraVE imposes this constraint structurally, eliminating the reliance on an explicit behavior model that can complicate online fine-tuning [23]. Consequently, BraVE can

BraVE	FAS	IQL
-53.5 $\pm$ 29.1	-1103.6 $\pm$ 51.4	-96.2 $\pm$ 51.8

Table 4: Performance on the 8-D, 50% pit CoNE task, averaged over five random permutations of the sub-action dimensions. BraVE’s effectiveness is not dependent on a specific tree construction.

be directly fine-tuned online without modification. Appendix C shows that BraVE matches IQL’s performance during fine-tuning when both methods begin from comparable offline initialization.

4.4 Ablations and Hyperparameters

We conduct ablation and hyperparameter studies to quantify the contribution of each of BraVE’s design choices. First, we examine the depth penalty δ , which scales the loss by a node’s position in the tree. A small penalty ($\delta = 1$) consistently yields the best results, confirming the importance of prioritizing accuracy near the root. Second, we vary the loss weight α to analyze the trade-off between the behavior-regularized TD loss and the BraVE loss. Performance remains stable across a range of α values, but omitting the TD term entirely ($\alpha = 0$) leads to significant degradation, particularly in higher-dimensional tasks. Third, we isolate the contribution of BraVE’s tree structure by training a standard DQN baseline, which omits the hierarchical traversal and branch value propagation. This variant fails to learn a viable policy, confirming that BraVE’s structured decomposition is critical to its success. Fourth, we evaluate the effect of beam width on performance and inference time. A width of 10 is sufficient for strong performance, with larger widths offering negligible gains at minimal runtime cost, reflecting the approach’s practical efficiency. Full results are presented in Appendix D.

5 Related Work

5.1 Tree-based RL

Tree-based methods in RL have historically been applied to ordered decision processes. Most notably, Monte Carlo Tree Search (MCTS) [6], as used in AlphaZero [26], recursively selects actions using the PUCT algorithm [2]. However, its sequential structure makes it ill-suited for unordered action spaces like those considered here, where sub-actions must be selected simultaneously.

TreeQN [10] blends model-free RL with online planning by constructing a tree over learned state representations, refining value estimates via tree backups. Differentiable decision trees (DDTs) [25] enable gradient-based learning in RL by replacing the hard splits typical of decision trees with smooth transitions, which can later be discretized for interpretability. In the offline setting, Ernst et al. [9] apply tree-based regression (e.g., CART, Kd-trees) to approximate Q-functions via fitted Q-iteration.

5.2 Combinatorial Action Spaces

Many RL approaches have been developed for combinatorial action spaces, particularly in domain-specific contexts such as text-based games [16, 17, 34], routing [7, 24], TSP [4], and resource allocation [5]. These methods often depend on domain structure, whereas BraVE is domain-agnostic.

General-purpose approaches include distributed action representations [29], curriculum-based action space growth [11], and search-based amortized Q-learning [32], which replaces exact action maximization with optimization over sampled proposals. Wol-DDPG [8] embeds discrete actions into continuous spaces, though this approach has been found to be ineffective in unordered settings [5]. Notably, these are online approaches, and their application to offline settings is often non-trivial. For example, Zhao et al. [35] rely on state-action visitation counts, which are generally infeasible to obtain offline.

Closer to our setting, Tang et al. [27] and Beeson et al. [3] evaluate Factored Action Spaces (FAS in our experiments), which linearly decompose the Q-function by conditioning each term on a single sub-action. While this reduces dimensional complexity, it relies on sub-action independence; a condition that often fails in real-world domains. Crucially, these limitations are algorithm-agnostic and apply to factorized variants of all offline RL methods. BraVE, by contrast, structures the action space to retain sub-action dependencies without incurring exponential cost.

6 Discussion and Conclusion

In many real-world decision-making problems, combinatorial action spaces arise from the simultaneous selection of multiple sub-actions. Offline RL in these settings remains challenging due to the cost of optimization and the difficulty of modeling sub-action dependencies from fixed datasets. Standard methods do not scale, while factorized approaches improve tractability at the expense of expressivity. We present **Branch Value Estimation (BraVE)**, an offline RL method for discrete combinatorial action spaces. By structuring the action space as a tree, BraVE captures sub-action dependencies while reducing the number of actions evaluated per timestep, enabling scalability to large spaces. BraVE outperforms state-of-the-art baselines across environments with varying action space sizes and sub-action dependencies.

While BraVE demonstrates strong performance, several limitations and opportunities for future work remain. First, the tree structure is defined by a fixed ordering of the sub-action dimensions. While Section 4.2 demonstrates robustness to this choice in CoNE, future work could explore methods for learning or adaptively selecting the dimension sequence. Second, while BraVE’s inference complexity is linear in the number of dimensions (N), the size of the network’s output layer and the tree’s branching factor grow with the sub-action cardinality ($|\mathcal{A}_d|$), which may impact performance on tasks with extremely high-cardinality discrete actions. Third, the recursive update for L_{BraVE} requires backpropagation through all parent nodes of a sampled action, incurring a greater computational cost during training than a standard Q-learning update. Fourth, our empirical validation is confined to the synthetic CoNE benchmark; future work should assess BraVE’s generalizability across a broader range of combinatorial tasks and real-world applications. Finally, adapting the BraVE framework to online learning is an important direction for future work, requiring the development of new mechanisms to manage the exploration-exploitation trade-off within the structured action space.

Despite these limitations, BraVE provides a flexible and extensible foundation for future research on combinatorial action spaces. We believe its applicability extends well beyond the specific setting explored in this paper. In particular, constrained RL presents a promising direction, as BraVE’s tree structure naturally supports the exclusion of inadmissible action combinations through pruning.

References

- [1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International conference on machine learning*, pages 104–114. PMLR, 2020.
- [2] David Auger, Adrien Couetoux, and Olivier Teytaud. Continuous upper confidence trees with polynomial exploration–consistency. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23–27, 2013, Proceedings, Part I 13*, pages 194–209. Springer, 2013.
- [3] Alex Beeson, David Ireland, and Giovanni Montana. An investigation of offline reinforcement learning in factorisable action spaces. *arXiv preprint arXiv:2411.11088*, 2024.
- [4] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
- [5] Changyu Chen, Ramesha Karunasena, Thanh Nguyen, Arunesh Sinha, and Pradeep Varakantham. Generative modelling of stochastic actions with arbitrary constraints in reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- [6] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.
- [7] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33:609–620, 2020.
- [8] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

- [9] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.
- [10] Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. *arXiv preprint arXiv:1710.11417*, 2017.
- [11] Gregory Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. Growing action spaces. In *International Conference on Machine Learning*, pages 3040–3051. PMLR, 2020.
- [12] Yuwei Fu, Wu Di, and Benoit Boulet. Batch reinforcement learning in the real world: A survey. In *Offline RL Workshop, NeuroIPS*, 2020.
- [13] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.
- [14] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.
- [15] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.
- [16] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*, 2015.
- [17] Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. Deep reinforcement learning with a combinatorial action space for predicting popular reddit threads. *arXiv preprint arXiv:1606.03667*, 2016.
- [18] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.
- [19] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.
- [20] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33: 1179–1191, 2020.
- [21] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning: State-of-the-art*, pages 45–73. Springer, 2012.
- [22] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- [23] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.
- [24] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.
- [25] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pages 1855–1865. PMLR, 2020.
- [26] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharmashan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

- [27] Shengpu Tang, Maggie Makar, Michael Sjoding, Finale Doshi-Velez, and Jenna Wiens. Leveraging factored action spaces for efficient offline reinforcement learning in healthcare. *Advances in Neural Information Processing Systems*, 35:34272–34286, 2022.
- [28] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.
- [29] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 2018.
- [30] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 connectionist models summer school*, 1993.
- [31] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL <https://arxiv.org/abs/2407.17032>.
- [32] Tom Van de Wiele, David Warde-Farley, Andriy Mnih, and Volodymyr Mnih. Q-learning in enormous action spaces via amortized approximate maximization. *arXiv preprint arXiv:2001.08116*, 2020.
- [33] Jinsung Yoon, James Jordon, and Mihaela Schaar. Asac: Active sensing using actor-critic models. In *Machine Learning for Healthcare Conference*, pages 451–473. PMLR, 2019.
- [34] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. *Advances in neural information processing systems*, 31, 2018.
- [35] Xutong Zhao, Yangchen Pan, Chenjun Xiao, Sarath Chandar, and Janarthanan Rajendran. Conditionally optimistic exploration for cooperative deep multi-agent reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 2529–2540. PMLR, 2023.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: The abstract and introduction (Section 1) present BraVE as a novel offline reinforcement learning method for combinatorial action spaces, designed to model sub-action dependencies while maintaining tractability. The claims of improved performance over prior methods are supported by the algorithmic details in Section 3 and the empirical results in Section 4.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: Section 6 (Discussion and Conclusion) outlines several limitations of BraVE and corresponding directions for future work. These include the reliance on a fixed sub-action ordering for tree construction, scalability challenges with high sub-action cardinality or deep trees during L_{BraVE} computation, dependence on offline dataset quality, and the opportunity for broader empirical evaluation.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not present new theoretical results or proofs; its contributions are primarily empirical.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: BraVE is described in detail in Section 3, with pseudocode provided in Algorithm 1. Section 4 outlines the experimental setup, environments, baselines, and training procedures. Additional environment details are included in Appendix B. To support reproducibility, we have released our code at: <https://anonymous.4open.science/r/BraVE-28CD>.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

- (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
- (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: An anonymized link to our implementation is provided in a footnote on page 1. This supplemental material includes instructions for running the code and reproducing results. It also contains an implementation of CoNE, including the dataset generation process.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Section 4 (Experimental Evaluation) describes the environment setup, dataset construction, baselines, and experimental configurations. Training specifics, including the number of gradient steps and evaluation frequency, are also included in Section 4. Details regarding BraVE’s hyperparameters (e.g., α , δ) and ablation studies are provided in Section 4 and Appendix D. An anonymized implementation has been published to support reproducibility.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All experimental results reported in Tables 1, 2, and 3 include means and standard deviations computed over five random seeds, as stated in Section 4. Learning curves in Appendices B, C, and D include shaded regions indicating one standard deviation.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Appendix B specifies that experiments were run on a single NVIDIA A40 GPU. Appendix D reports wall-clock training times for key experiments conducted on a 10-core CPU with 32 GB of unified memory and no GPU.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: This work involves algorithmic development for reinforcement learning in simulated environments and does not involve human subjects, sensitive data, or applications with direct ethical implications beyond general considerations relevant to AI research. We have reviewed the NeurIPS Code of Ethics and believe our work is in full compliance.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: This paper presents foundational algorithmic contributions for offline reinforcement learning for combinatorial action spaces. While some potential application areas (e.g., healthcare, logistics) carry societal implications, this work does not explore specific deployment scenarios or their broader societal impact, which would be highly dependent on the application context.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper introduces a new algorithm (BraVE) and a simulated environment (CoNE). Neither the algorithm nor the environment/data generation process presents a high risk of misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: This paper does not rely on external assets such as third-party code, datasets, or pre-trained models beyond standard libraries (e.g., PyTorch) that do not require special licensing. The CoNE environment and datasets are generated by the authors.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The primary new asset introduced in this work is the BraVE algorithm and its implementation, described in Section 3 and released with instructions via an anonymized link. The CoNE environment is also a new asset, described in Section 4, and included in the published code.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The research involves neither crowdsourcing nor experiments with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The research does not involve human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs were not used as an important, original, or non-standard component of the core methods in this research.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.

A Limitations of Standard Value-Based Estimation and Q-Function Factorization in Combinatorial Action Spaces with Sub-Action Dependencies

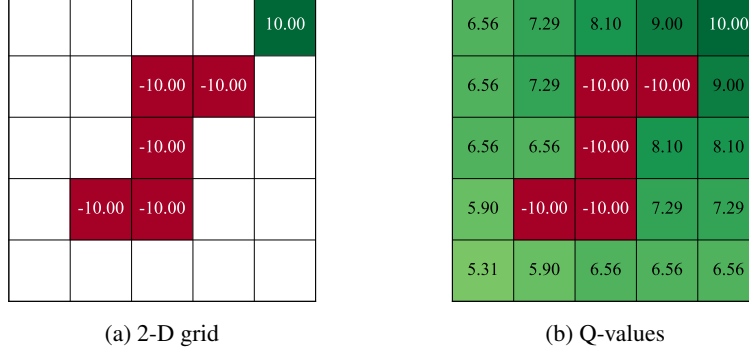


Figure 4: A 2-D grid with five pits and the true maximum Q-values in each state.

Consider a two-dimensional grid with five pits, which provides a simplified instance of CoNE, allowing us to clearly illustrate the limitations of standard methods. At each state, the agent chooses from 16 possible actions ($|\mathcal{A}| = 16$), corresponding to combinations of movement directions (e.g., $\emptyset$, [U], [UD], [UDL], [ULR], [UDLR], [D], [DR], $\dots$). Transitions yield a reward of $r = 0$ unless they lead to the goal ($r = 10$) or a pit ($r = -10$), as shown in Figure 4a.

Sub-action dependencies are inherently captured by the standard Q-function, as they affect both immediate rewards and future returns:

$$Q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s, a, s') \left(\sum_{a' \in \mathcal{A}} \pi(a' | s') Q_\pi(s', a') \right).$$

Thus, the agent can learn the true maximum Q-value in each state (Figure 4b) and, consequently, recover the optimal policy.

Accurately estimating the Q-function is challenging in combinatorial action spaces, where the number of possible actions grows exponentially with the action dimension. This complexity limits the applicability of value-based RL methods, such as IQL, which require estimating values for all actions.

More specifically, IQL’s policy loss is defined as:

$$L_\pi(\phi) = \mathbb{E}_{(s,a) \sim \mathcal{B}} \left[\exp \left(\beta \left(Q(s, a; \theta^-) - V(s; \psi) \right) \right) \log \pi_\phi(a | s) \right].$$

In discrete action settings, the policy $\pi_\phi(a | s)$ is typically parameterized as a flat categorical distribution. That is, a policy network maps the state s to a vector of logits $\ell_\phi(s) \in \mathbb{R}^{|\mathcal{A}|}$, where $|\mathcal{A}|$ is the number of discrete actions. The softmax function then transforms these logits into a probability distribution:

$$\pi_\phi(a | s) = \frac{\exp(\ell_\phi(s)_a)}{\sum_{b=1}^{|\mathcal{A}|} \exp(\ell_\phi(s)_b)}.$$

This formulation requires a forward pass through the policy network to produce logits for all possible actions, incurring computational cost and modeling complexity that scale with $|\mathcal{A}|$.

BraVE evaluates only a subset of possible actions at each timestep, allowing it to exploit the Q-function’s capacity to capture sub-action dependencies without predicting an exponential number of action values or logits simultaneously. Factored approaches (FAS) [3, 27], by contrast, simplify computation by linearly decomposing the Q-function, conditioning each component on a single sub-action and the full state. While this reduces the dimensional complexity of the action space, it imposes structural constraints on the reward and Q-function definitions:

$$r(s, a) = \sum_{d=1}^D r_d(s, a_d) \quad \text{and} \quad Q(s, a) = \sum_{d=1}^D q_d(s, a_d), \quad (6)$$

that may limit expressiveness in settings with sub-action dependencies and cause the Q-function to *diverge*.

More specifically, when sub-actions are combined (e.g., “up” + “left”), FAS retrieves partial Q-values $q_1(s, \text{up})$ and $q_2(s, \text{left})$ from different next states, rather than from a single consistent next state determined by the combined action. This results in artificially inflated values, as the summations in Equation (6) incorrectly assume independence. Iterative Bellman updates, as used in Q-learning, amplify this inconsistency, ultimately causing Q-value divergence.

The independence assumption holds in discretized variants of popular continuous control environments, where sub-actions typically influence distinct parts of the state space [28, 31]. However, this assumption is violated in many real-world settings, a limitation explicitly acknowledged in the FAS literature [3, 27].

B CoNE Learning Curves

We compare BraVE’s performance to two state-of-the-art baselines: Factored Action Spaces (FAS) [27, 3], which learns linearly decomposable Q-functions for offline combinatorial action spaces, and Implicit Q-Learning (IQL) [19], a general-purpose offline RL method. All methods are evaluated across 20 instances of CoNE. In CoNE, the sizes of both the action and state spaces increase exponentially with the environment’s dimensionality: a D -dimensional setting with M positions per axis yields $|\mathcal{A}| = 2^{2D}$ joint actions and $|\mathcal{S}| = M^D$ distinct states. All experiments were conducted on a single NVIDIA A40 GPU using Python 3.9 and PyTorch 2.6.

In all curves, results are averaged over 5 seeds; shaded regions indicate one standard deviation.

B.1 Non-Factorizable Reward Structures Only

In environments without pits, transition dynamics are factorizable across sub-actions. The reward function, however, remains non-linear and cannot be decomposed. Below we present the training curves corresponding to the results in Table 1.

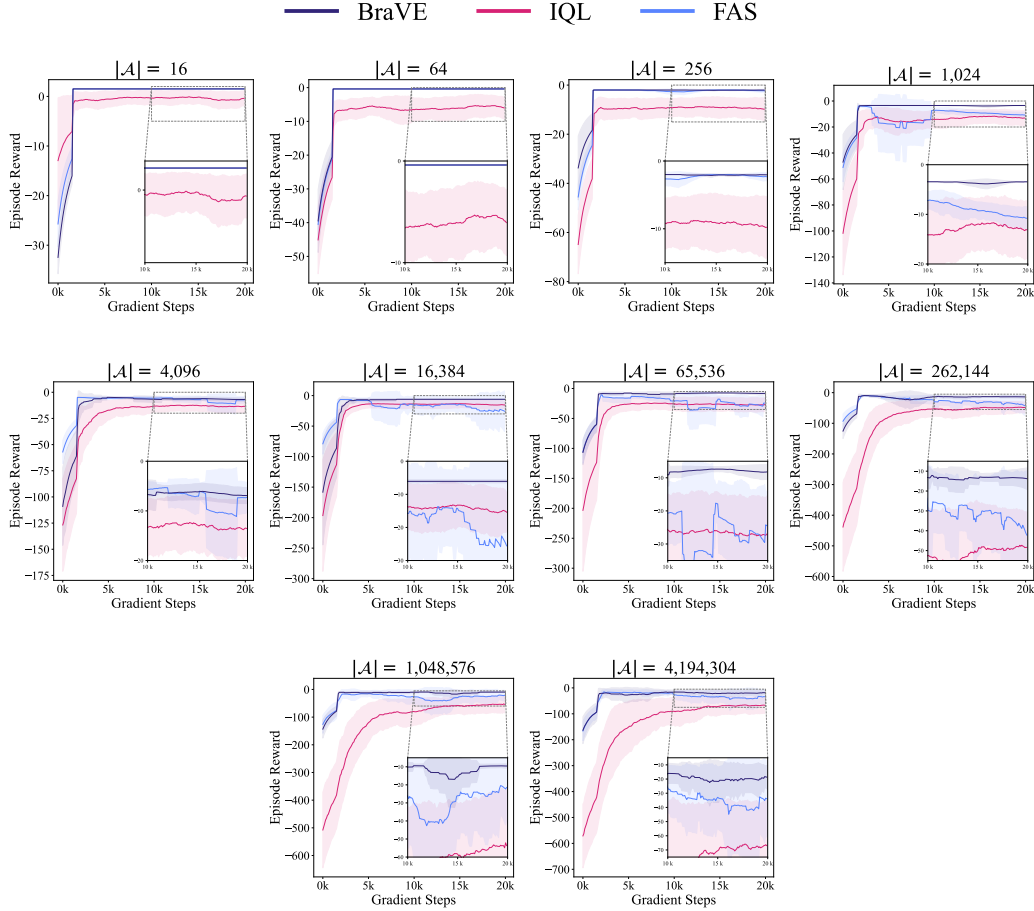


Figure 5: Learning curves for BraVE, FAS, and IQL in environments with non-factorizable reward structures but no sub-action dependencies.

In low-dimensional settings, FAS performs similarly to BraVE, but its performance degrades as dimensionality increases due to the bias introduced by linear Q-function decomposition. BraVE, by contrast, remains stable across all action space sizes by evaluating joint actions directly. IQL performs relatively poorly in all configurations, indicating that general-purpose offline RL methods struggle even when transition dynamics are factorizable.

B.2 Non-Factorizable Reward Structures and Sub-Action Dependencies

To assess performance under increasing sub-action dependencies, we introduce hazardous transitions by varying the density of pits in an 8-D environment ($|\mathcal{A}| = 65,536$). The number of pits ranges from 5% to 100% of the 6,561 interior (non-boundary) states. Below we present the training curves and normalized episode-length scores corresponding to the results in Table 2.

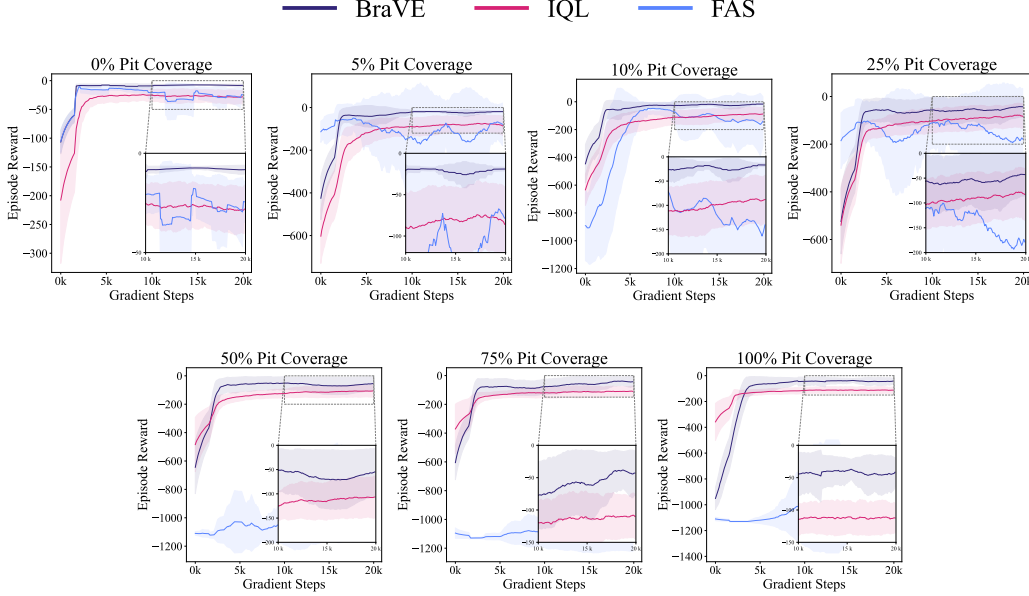


Figure 6: Learning curves for BraVE, FAS, and IQL in environments with non-factorizable reward structures and sub-action dependencies.

Learning Curves BraVE maintains stable returns across all pit densities, whereas FAS degrades sharply when pits cover at least half the interior states. IQL performance degrades as dependency strength increases. These results demonstrate BraVE’s effectiveness in environments with tightly coupled sub-action effects and frequent adverse interactions.

Normalized Episode-Length Score To further contextualize performance, we compute a normalized episode-length score that places each method on a 0–100 scale, where 0 corresponds to a random policy and 100 to an oracle planner. It is defined as:

$$\text{score}_{\text{norm}} = 100 \cdot \frac{\text{length}_{\text{random}} - \text{length}_{\text{agent}}}{\text{length}_{\text{random}} - \text{length}_{\text{oracle}}},$$

where $\text{length}_{\text{agent}}$ is the average episode length of the evaluated policy, $\text{length}_{\text{random}}$ is the episode length of a random policy (which consistently times out at 100 steps), and $\text{length}_{\text{oracle}}$ is the optimal path length computed via A^* search.

The normalized scores in Table 5 reinforce the findings from Table 2. BraVE consistently achieves scores above 70, indicating near-optimal behavior even in the most hazardous settings. FAS, by contrast, collapses to 0 — equivalent to a random policy — once dependencies become sufficiently strong.

Pit %	BraVE	FAS	IQL
0	99.3	89.6	88.0
5	92.8	57.3	52.9
10	94.1	20.2	51.5
25	78.5	0.0	54.7
50	71.4	0.0	40.6
75	78.5	0.0	38.4
100	79.3	0.0	37.0

Table 5: Normalized episode-length scores (0=random, 100=oracle) in the 8-D CoNE environment with varying pit densities. Scores show BraVE maintaining high performance while FAS’s performance collapses to that of a random policy.

B.3 Sparse but Critical Sub-Action Dependencies

To isolate the effect of sparse but critical sub-action dependencies, we consider environments with only five pits, representing a negligible fraction of the state space in higher dimensions. Starting in the 2-D environment, we progressively scale the state space until BraVE and FAS converge to similar average returns, allowing us to evaluate how each method handles **low-frequency but high-impact hazards**. Below we present the training curves corresponding to the results in Table 3.

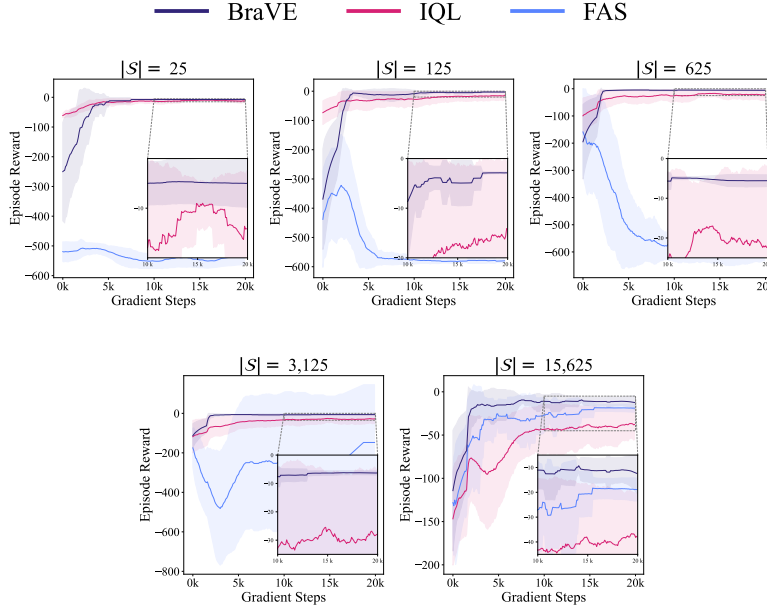


Figure 7: Learning curves for BraVE, FAS, and IQL in environments with sparse but critical sub-action dependencies.

BraVE performs reliably across all settings with sparse but critical sub-action dependencies even when the environment is small and each action carries significant risk. FAS, by contrast, requires much larger state spaces before the influence of the pits diminishes and its performance approaches that of BraVE. These findings indicate that even a small number of hazardous transitions can invalidate simplistic factorizations. IQL performs worse than BraVE in all settings.

C Online Fine-Tuning After Offline Learning

To evaluate BraVE’s capacity for online fine-tuning, we compare it to IQL, a method known for strong online adaptation following offline pretraining. FAS is excluded from this experiment because, as a factorized version of BCQ, it is not suited to online fine-tuning [23].

We conduct this experiment in an 8-D environment with 3,281 pits (50% of interior states), which is empirically the most challenging 8-D CoNE configuration for BraVE. Success in this setting best indicates generalizability to environments where BraVE demonstrates stronger performance.

Both methods are first trained offline for 1,500 gradient steps, after which BraVE achieves an average return of -226.1 compared to IQL’s -259.4. We terminate offline training at this point, as BraVE consistently surpasses IQL beyond this threshold (see Figure 6). Online fine-tuning then proceeds for an additional 5,000 environment steps.

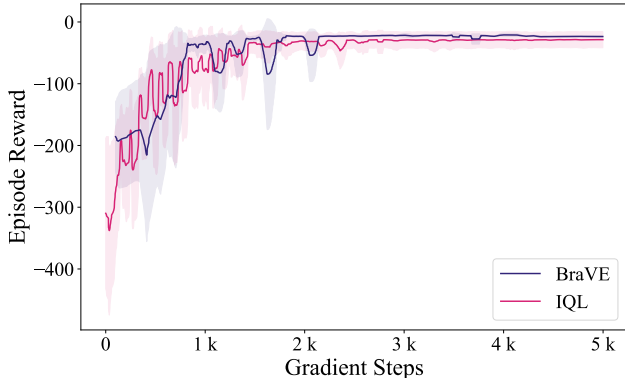


Figure 8: Learning curves for finetuning BraVE and IQL after 1,500 offline gradient steps.

As shown in Figure 8, the methods achieve comparable final performance during fine-tuning, with BraVE slightly outperforming IQL.

D Ablation and Hyperparameter Learning Curves

This section presents learning curves from four ablation and hyperparameter studies: (1) the effect of varying the depth penalty δ , (2) BraVE’s sensitivity to α , the weight of the TD error in the total loss, (3) the importance of BraVE’s tree structure, and (4) the effect of beam width on both return and computational overhead. A summary of these results appears in Figure 9, with full details provided in Appendices D.1, D.2, D.3, and D.4, respectively. All experiments are conducted in five-pit environments.

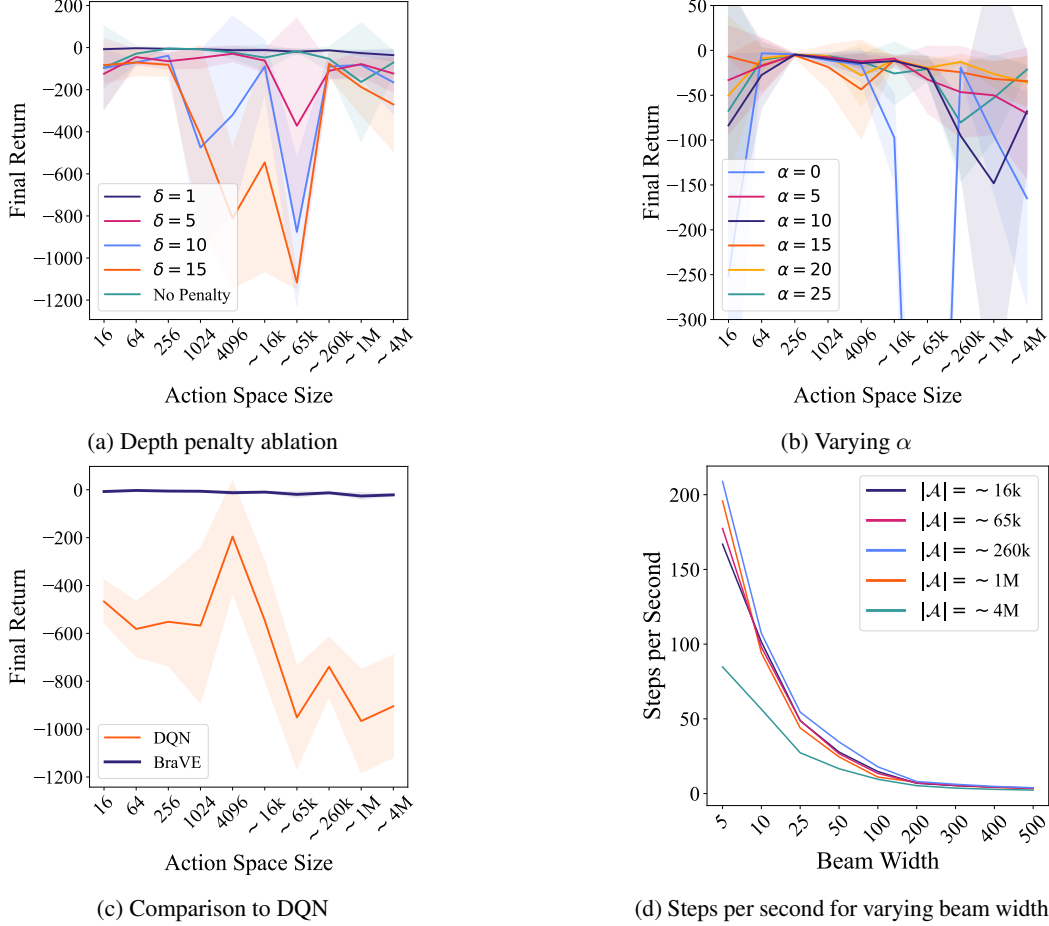


Figure 9: A small depth penalty ($\delta = 1$) yields the best performance across all environments (Figure 9a). Performance remains relatively stable across a range of α values; however, omitting the TD loss entirely ($\alpha = 0$) can result in suboptimal policies (Figure 9b). Although constraining the DQN to select actions within $\mathcal{B}$ introduces an inductive bias, it fails to produce a viable policy (Figure 9c). Beam width has minimal practical impact on runtime as inference remains under one second even with beam widths well above the optimal setting (Figure 9d).

D.1 Depth Penalty Ablation

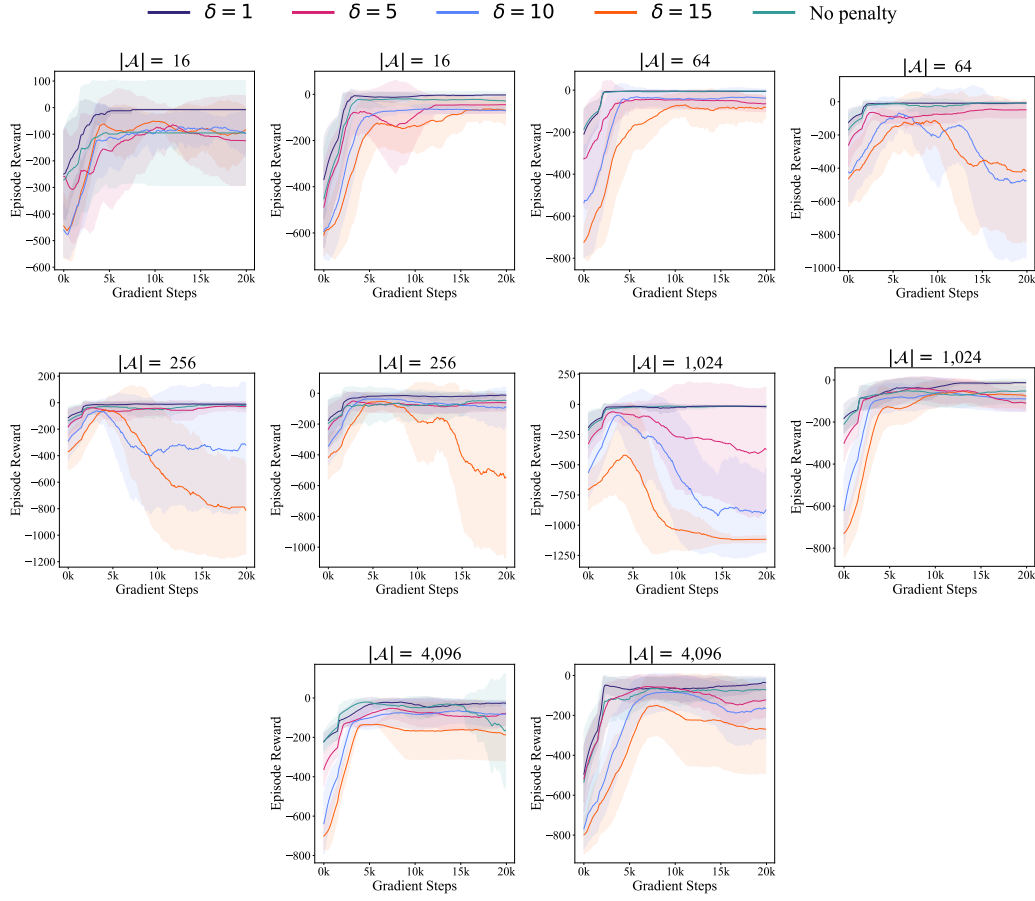


Figure 10: Learning curves for BraVE with varying depth penalty values (δ).

While the depth penalty improves performance, it should remain small; $\delta = 1$ yields optimal results across nearly all environments.

D.2 Varying TD Weight in Loss

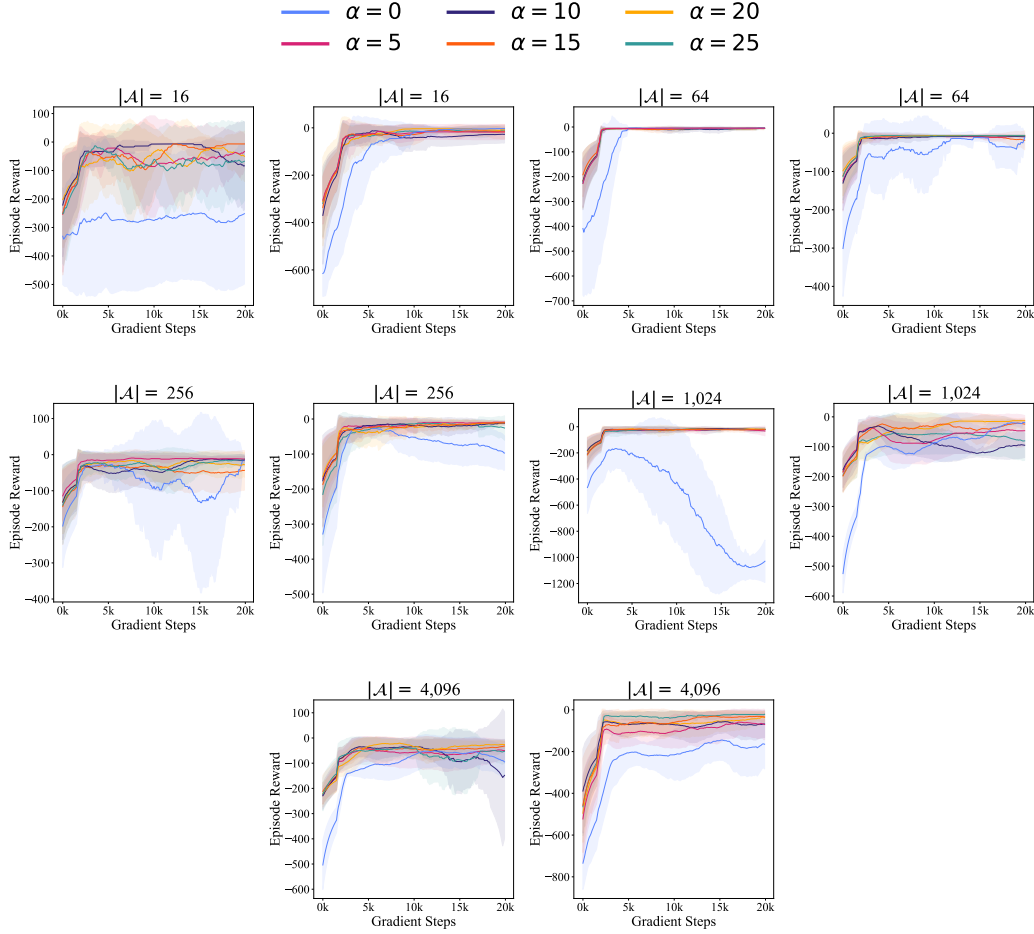


Figure 11: Learning curves for BraVE with varying TD loss weight values (α).

Performance is relatively stable across α values in low-dimensional environments, with the exception of the 2-D case, where the high pit density makes accurate decision-making critical. In higher-dimensional settings, BraVE becomes more sensitive to the choice of α . In particular, omitting the TD loss term ($\alpha = 0$) can result in catastrophic failures, especially in complex environments such as the 8-D setting.

D.3 Comparison to DQN

To isolate the effect of BraVE’s tree structure, we compare it to a DQN constrained to selecting actions from the dataset $\mathcal{B}$ and trained using BraVE’s behavior-regularized TD loss (Equation 4).

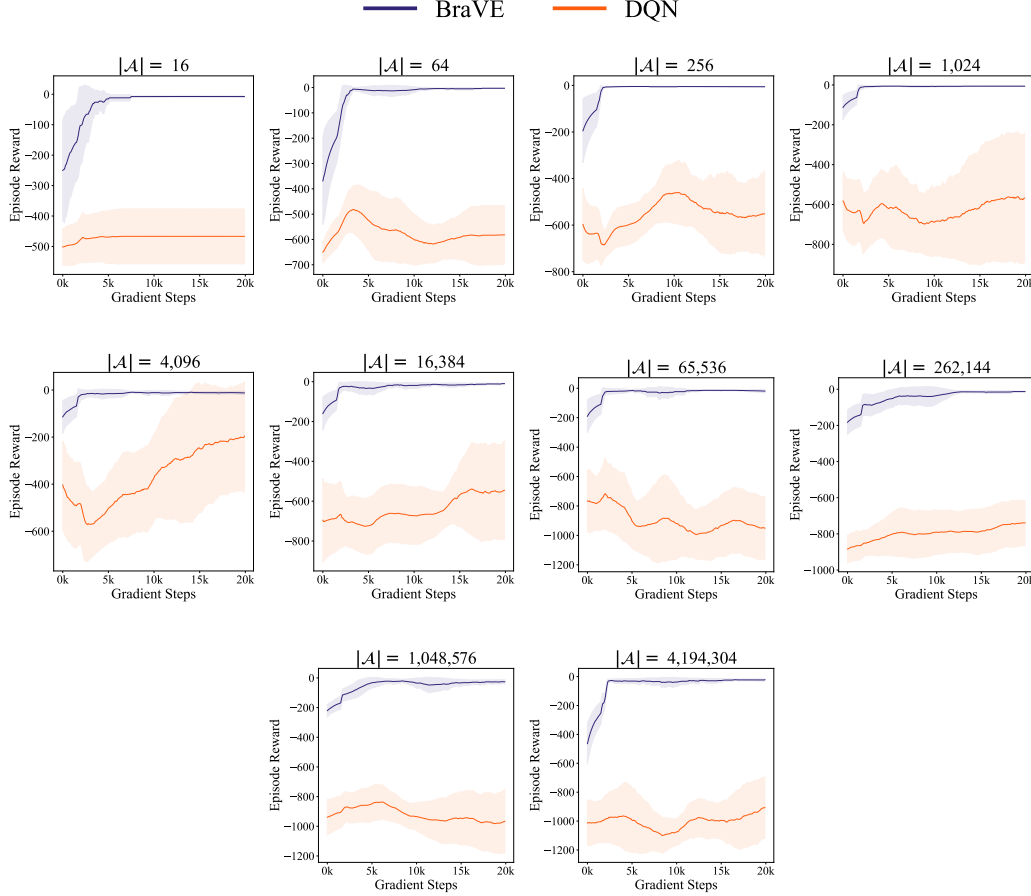


Figure 12: Learning curves for BraVE and a DQN constrained to select actions from $\mathcal{B}$ and trained with BraVE’s behavior-regularized TD loss (Equation 4) — effectively, BraVE without tree-based action selection or branch value propagation.

Although the DQN is trained with the same behavior-regularized TD loss as BraVE and is restricted to selecting actions in $\mathcal{B}$, it fails to learn an effective policy. This suggests that the DQN struggles to capture sub-action dependencies, particularly in large action spaces. For example, in the 11-D environment, it must evaluate all 8,927 unique actions in $\mathcal{B}$ simultaneously. BraVE mitigates this complexity by organizing the action space as a tree, reducing the number of required predictions to a small subset of Q-values at each timestep.

D.4 Beam Width

Because beam search is used only during policy extraction — after value estimates have been learned — it remains independent of the learning process. This decoupling allows for post-training optimization of beam width, provided the environment permits hyperparameter tuning. To evaluate the impact of beam width on both return and computational overhead, we measure return, environment steps per second, and total episode runtime for beam widths ranging from 5 to 500.

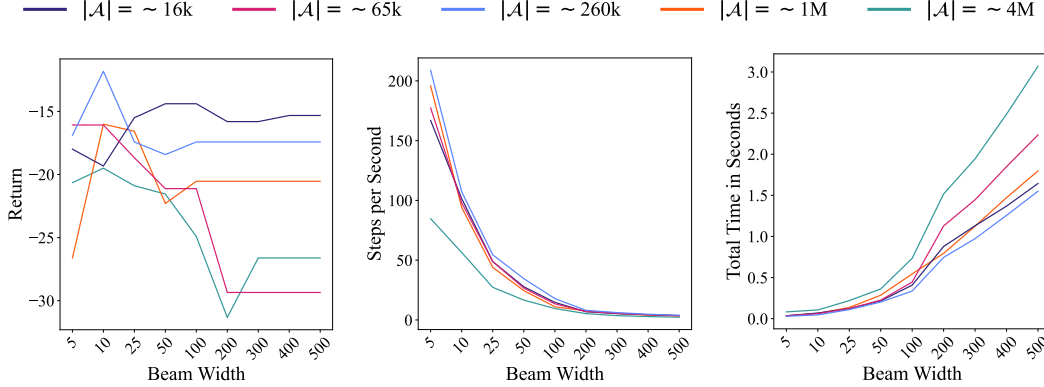


Figure 13: Effect of beam width on return and computational cost, measured in environment steps per second and total episode runtime.

A beam width of 10 is generally sufficient for near-optimal performance. Larger beams can degrade performance by exploring low-confidence regions of the tree, where Q-value estimates are less reliable. Narrower beams act as a form of regularization, limiting search to well-learned regions of the action space.

Beam width has negligible practical impact on runtime: inference times remain well below one second, even for beam widths far exceeding the optimal setting. These results are based on tests conducted on a 10-core CPU with 32 GB of unified memory and no GPU.