Using Gradient-based Optimization for Planning with Deep Q-Networks in Parametrized Action Spaces

Jonas Ehrhardt[1,2,*], Johannes Schmidt[1,2], René Heesch[1,2] and Oliver Niggemann[1,2]

[1]HSU-AI Institute for Artificial Intelligence, Helmut-Schmidt-University, Hamburg, Germany
[2]Institute of Automation, Helmut-Schmidt-University, Hamburg, Germany

Abstract

Many real-world planning problems feature parametrized action spaces, where each action is augmented by continuous parameters. Though deep Reinforcement Learning has achieved remarkable results in solving control and planning problems, it falls short at two central challenges of real-world planning problems with parametrized action spaces: (i) There is an infinite number of action-parameter candidates in every step of solving a planning problem, (ii) interacting with the planning domain is typically prohibitively expensive and available recordings from the planning domain are sparse. To counter these challenges, we introduce our novel Goal-Conditioned Model-Augmented Deep Q-Networks algorithm (GCM-DQN). The intuition behind GCM-DQN is to use gradient-based optimization on the surface of the Q-Function, instead of blunt estimators, to estimate the optimal parameters of an action in a state. In combination with a goal-conditioning of the DQN, and a state transition model, this allows us to find plans for planning problems in planning domains with parametrized action spaces. Our algorithm outperforms state-of-the-art Reinforcement Learning algorithms for planning in parametrized action spaces.

Keywords

Planning, Parametrized Markov Decision Processes, Offline Reinforcement Learning, Deep Q-Networks

1. Introduction

Planning, the combinatorial problem of finding a sequence of actions that transitions an initial state into a goal state, is a fundamental problem in many real-world applications and AI [1, 2]. Conventional planning and Reinforcement Learning methods typically feature either purely discrete action spaces (i.e. a finite set of actions, like moving up, down, left, or right in a grid world) *or* purely continuous action spaces (i.e. an infinite set of actions, like controlling the acceleration of a cart on a slope) [2, 3]. However, many real-world problems feature *parametrized action spaces*. In a parametrized action space, a finite set of actions is augmented by real-valued parameters, which influence the effects of the actions [3, 4, 5]. During planning in parametrized action spaces, a planner hence must not only select from the finite action set, but also real-valued parameters, to reach its goal [3]. For example, consider injection molding, where there is a finite set of actions (e.g. close mold, inject, hold, cool, eject), which are each augmented by real-valued parameters (e.g. heating/cooling energy, velocity, pressure, etc.). Both the combinatorial aspect of finite action selection, e.g., injecting material before closing the mold would lead to a mess, as well as the parametrization aspect, e.g., injecting too cold material leads to poor surface characteristics of the molded product majorly, have a major influence on the molded product. Getting both aspects right is the task of planning in parametrized action spaces. Besides this simplified example, many other real-world problems, from robotics to factory planning, feature parametrized action spaces [4, 3, 6, 7, 8].

There are two central challenges in solving planning problems in real-world parametrized action spaces: *(i)* Due to the continuous nature of the parameter space, there is an infinite number of action-

CAIPI'25: ECAI Workshop on AI-based Planning for Complex Real-World Applications, Bologna, Italy, 2025
*Corresponding author.
jonas.ehrhardt@hsu-hh.de (J. Ehrhardt); johannes.schmidt@hsu-hh.de (J. Schmidt); rene.heesch@hsu-hh.de (R. Heesch); oliver.niggemann@hsu-hh.de (O. Niggemann)
0000-0001-5023-839X (J. Ehrhardt); 0009-0005-6532-5740 (J. Schmidt); 0000-0003-1147-8205 (R. Heesch); 0000-0001-8747-3596 (O. Niggemann)

parameter tuples a planner has to choose from in every state. This *infinite branching* of action-parameter tuples in every state poses a challenge for selecting the optimal action-parameter tuple [9]. Typically, infinite branching is either countered by parameter estimators [10], which have the risk of being imprecise, or search [11], which has the risk of being computationally expensive. *(ii)* Often there is no sufficient model of the planning domain available, interaction with the domain is prohibitively expensive or unsafe, and recorded data is scarce [12]. Hence, solving planning problems typically, either requires a manually crafted, expensive, and error-prone planning domain model [13, 5], or requires advanced Reinforcement Learning algorithms which can be trained offline, meaning without interaction with the planning domain, but strongly rely on the assumption that the distribution of the recorded data does not shift strongly from the application cases [12].

In this paper, we tackle the challenges of infinite branching and training data scarcity in real-world parametrized action spaces. Therefore, we propose to extend the well known Deep Q-Network (DQN) algorithm [14]. DQN uses a Neural Network to approximate the action value function, which returns the expected cumulative return of taking an action in a state. In combination with a greedy policy, DQN can solve even complex planning and control problems [14]. We propose to transfer DQN into a novel, offline and model-augmented Reinforcement Learning setup, which allows us to use it for solving planning problems in planning domains with parametrized action spaces [3] (cf. Figure 1). More precisely, we propose three extension to the DQN algorithm: *(a)* To tackle infinite branching, we introduce *paramOpt*, a novel gradient-based optimization algorithm, to efficiently find optimal parameters for a given action in a given state. *(b)* To make our algorithm applicable to unseen planning problems, we integrate a goal-conditioning to the DQN [15]. *(c)* To allow using the DQN for planning without interacting with the environment, we propose a novel state-transition model, which is trained along the DQN and allows for planning in deterministic and probabilistic domains. We reduce the amount of training data to fit the models, by employing Hindsight Experience Replay [16] and Conservative Q-Learning [17].

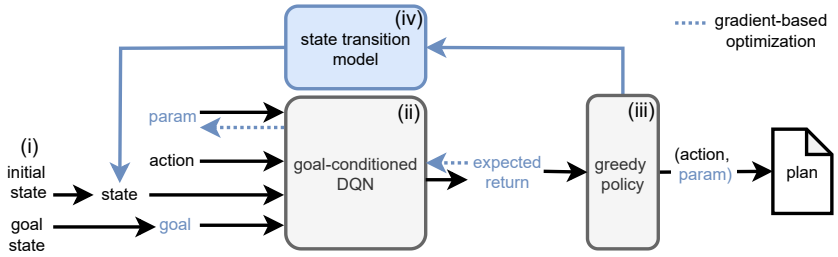

Figure 1: In this paper, we propose the Goal-Conditioned Model Augmented DQN algorithm (GCM-DQN), an offline extension to DQN [14] that allows for planning in domains with parametrized action spaces (novel extensions to DQN are marked in blue). For planning, GCM-DQN takes an input and goal state tuple (i) and iteratively computes optimal action-parameter tuples, using a gradient-based optimization (ii). A greedy policy picks the action-parameter tuple with the highest Q-value (iii). Using a state transition model, the subsequent state is computed (iv). The planning stops, when the current state matches the goal state.

As a result, we present our Goal-Conditioned Model Augmented DQN algorithm (GCM-DQN). GCM-DQN is can be trained on a sparse dataset of recorded plans from a planning domain. It returns a DQN which can either be used as a policy in probabilistic scenarios, or in combination with the parallelly trained state transition model as planner for deterministic domains. In contrast to estimator or search-based algorithms for planning in parametrized action spaces, GCM-DQN converges quickly to optimal parameters due to the gradient-based parameter optimization. The main contributions of our paper are:

- *paramOpt* novel gradient-based optimization algorithm to efficiently counter infinite branching in planning domains with parametrized action spaces.
- A novel integration of *paramOpt*, goal-conditioning, and a novel state-transition model into DQN to allow harnessing it for planning.
- A systematic and comprehensive evaluation of our approach against state-of-the-art Reinforcement Learning paradigms for parametrized action spaces.

The remaining paper is structured as follows: In Section 2 we review related research in the domains of Reinforcement Learning for planning in parametrized action spaces. In Section 3 we introduce the formalization of our problem. Section 4 introduces our solution, followed by its theoretical and empirical evaluation in Section 5 and discussion in Section 6. We conclude our paper in section 7.

2. Related Work

In Deep Reinforcement Learning, there are two directions when handling parameterized action spaces: Using Neural Networks as estimators that suggest parameters for actions, and using search or optimization to find optimal parameters for an action. Typically, policy network approaches are grounded in the Deep Deterministic Policy Gradient (DDPG) paradigm [10]. DDPG is an Actor-Critic approach, in which the actor is a deep policy network that, given a state, suggests actions and the critic is a deep Q-network that calculates the cumulative expected return of the suggested action and state. Using backpropagation over both networks allows for adapting their weights to converge to an optimal policy- and Q-network. To solve planning problems in parametrized action spaces, Hausknecht and Stone [4] extendeded the DDPG paradigm by expanding the deep policy network with an additional non-binary output for suggesting parameters values, resulting in the P-DDPG algorithm. Fan et al. [18] propose a similar approach. They use individual separate heads for selecting an action from the finite action set, and individual separate heads for estimating its numerical parameters [18]. However, both approaches neglect that there is a dependency between an action and the numerical parameters [19]. Hence, Li et al. [19] proposed to encode the finite set of actions and numerical parameters into a joint latent representation space on which the policy operates, and from which discrete and continuous components are decoded for interaction with the environment. While the introduced approaches can handle parametrized action spaces, they remain restricted to online settings, which require the agent to interact directly with the environment, and are not well suited to an offline scenario with only little available training data.

Optimization or search-based approaches typically follow a value-based paradigm, in which a greedy policy selects the action-parameter tuple with the highest expected return. While methods like [20] use a divide-and-conquer approach for complex actions-parameter tuples that operates on a joint latent representation, Xiong et al. [6] uses a separate parameter estimation network which feeds into a DQN, forming a parametrized DQN or P-DQN. Thereby, they can select a discrete action directly using a greedy policy and do not rely on a continuous relaxation of the discrete action components (as, e.g., Hausknecht and Stone [4]) [6]. Finally, Ma et al. [11] uses an evolutionary optimization algorithm for estimating an optimal action from a continuous action space. While such approaches can also be adapted to parametrized action spaces, they are computationally expensive due to the uninformed optimization paradigm.

In contrast to typical Reinforcement Learning tasks, e.g., like control, the reward structure in planning problems sparse. Typically, the reward for solving a planning problem is formalized by a single reward signal upon reaching the goal state. This sparse reward signal hence is exclusively dependent on the goal state, and changes for planning problems with diverging goal states. To make Reinforcement Learning agents applicable to altering reward functions, Schaul et al. [15] introduced Universal Value Function Approximators. Universal Value Function Approximators condition the value function approximator on an embedding of the goal state, hence making it generalizable across altering planning problems within the same domain [15]. Other methods for countering sparsity of reward signals, especially in offline settings, include data augmentation, such as Hindsight Experience Replay [16], or regularization in training by additional loss terms, such as Conservative Q-Learning [17].

3. Formalization

Reinforcement Learning follows the assumption that there is an underlying MDP within all planning domains. As we focus on planning problems in parametrized action spaces, we consider Parametrized

Action Markov Decision Processes (PAMDP) [3].

3.1. Parametrized Action Markov Decision Processes

PAMDPs extend continuous Markov Decision Processes by introducing a hybrid, so-called, parametrized action space. They can be formalized as a tuple

$$\langle \mathcal{S}, A, \Psi, \mathcal{T}, \mathcal{R}, \gamma \rangle, \tag{1}$$

where $\mathcal{S} \subseteq \mathbb{R}^n$ is the continuous state space, $A = \{a_0, ..., a_k, ..., a_K\}, K \in \mathbb{N}$ is a finite set of actions, in which each action a_k is extended by a continuous parameter space $\Psi_k \in \mathbb{R}$ and the union of all parameter spaces is given as $\Psi = \bigcup_{k=1}^{K} \Psi_k$. Together they form the parametrized action space

$$\mathcal{A} = \bigcup_{a_k \in A} \{(a_k, \psi_k) | \psi_k \in \Psi_k\}. \tag{2}$$

$\mathcal{T}$ is the transition function $\mathcal{T} = P(s_{t+1}|s_t, a_t, \psi_t)$ that describes the probability of transitioning into state $s_{t+1} \in \mathcal{S}$ given state $s_t \in \mathcal{S}$, action $a_t \in A$ and a parameter $\psi_t \in \Psi$ at time t. $\mathcal{R}$ is the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that returns the scalar reward r when transitioning from s_t into s_{t+1} using an action a_t, and $\gamma \in \mathbb{R}$ a discount factor. We will further refer to $\mathcal{T}$ as the dynamics of the MDP.

As the transition dynamics in real-world PAMDPs can grow very complex, large models and large datasets are needed to properly capture them. Leveraging on the parametrized action spaces, we propose to manage the complexity of real-world dynamics by a modular factorization of the parametrized action space. Therefore, we split $\mathcal{T}$ into a finite set $\mathcal{T}_d$ of K transition functions $\mathcal{T}_{a_k}$, which each are related to one individual action a_k each:

$$\mathcal{T}_d = \{\mathcal{T}_{a_k} | \mathcal{T}_{a_k} = P_{a_k}(s_{t+1}|s_t, \psi_k), \;\; \psi_k \in \Psi_k, \;\; k = 1, ..., K\} \tag{3}$$

This allows us to model the transition dynamics for each action in one individual model $f_{a_k} \approx \mathcal{T}_{a_k}$, reducing the complexity of the modeling problem, while overall not affecting the PAMDP dynamics. We can denote the collection of all f_{a_k} as $\mathcal{F} = \{f_{a_k}\}_{k=1}^{K}$. During planning, we can infer state transitions by sampling from the transition models

$$s_{t+1} \sim f_{a_k}(s_t, \psi_t). \tag{4}$$

In deterministic scenarios, the transition probabilities of $\mathcal{T}_{a_k}$ collapse to a Dirac delta distribution, which effectively turns f_{a_k} into a deterministic function

$$f_{a_k}(s_t, \psi_t) = s_{t+1}. \tag{5}$$

3.2. Describing Planning Problems with PAMDPs

Planning describes the task of finding a sequence $\tau = \{(a_t, \psi_t)\}_{t=0}^{T-1}$ of T action-parameter tuples, that transition an initial state s_0 into a goal state $g \in G \subset \mathcal{S}$. Hence, a planning problem in a PAMDP can be denoted as

$$\langle \mathcal{S}, A, \Psi, \mathcal{F}, \mathcal{R}_G, \gamma, s_0, G \rangle, \tag{6}$$

where $\mathcal{R}_G$ is a goal conditioned, sparse reward function

$$\mathcal{R}_G(s) = \begin{cases} r, & \text{if } s \in G \\ 0, & \text{else} \end{cases} \tag{7}$$

, with the numerical reward value $r \in \mathbb{R}$.

Reinforcement Learning typically solves planning problems by iteratively applying a policy π on the planning problem. Hence, a plan can be seen as a trajectory-level instantiation of a policy. A policy in a PAMDP is a mapping from the current state s_t and goal state g to an action-parameter tuple. For

deterministic planning domains, the mapping is a function $\pi_{(\text{det})}(s_t, g) = (a, \psi)$, For probabilistic planning domains, the mapping is a conditional distribution $\pi((a, \psi)|s_t, g)$, where $s_t \in \mathcal{S}, \quad g \in G, \quad a, \psi \in \mathcal{A}$.

For deterministic domains, the solution of a planning problem is a plan τ, which, when executed from s_0, reaches a $g \in G$. For probabilistic domains, the solution of a planning problem is a proper policy π. A proper policy optimizes the discounted return of the planning problem and results in a goal state $g \in G$. The sequence of actions-parameter tuples selected by the policy during execution forms a plan τ.

4. Solution

In this section, we introduce our GCM-DQN algorithm. GCM-DQN tackles the challenges of infinite branching, prohibitively expensive domain interactions, and data scarcity in real world planning domains with parametrized action spaces. The intuition of GCM-DQN is to leverage on the differentiability of a DQN [14] during planning for finding the optimal parameters and actions via gradient-based optimization, instead of using estimators or search. Therefore, we add three extensions to the DQN algorithm [14]: *(a)* To tackle the problem of infinite branching, we introduce the *paramOpt* algorithm, a gradient-based optimization algorithm inspired by [21, 8], for finding an (leastwise locally) optimal action-parameter tuple during planning (cf. Section 4.3). *(b)* To make GCM-DQN applicable to any planning problem within the planning domain, we introduce a goal-conditioning to the DQN, as proposed in [15]. We tackle data scarcity in training the goal-conditioned DQN, by using Hindsight Experience Replay [16] and Conservative Q-Learning [17] (cf. Section 4.2). *(c)* Finally, to counter prohibitively expensive domain interaction, we propose a novel state transition model which is parallelly trained to the DQN on the same dataset (cf. Section 4.4), allowing to simulate state transitions without any interaction with the planning domain.

By combining the three proposed extensions, we result in our novel GCM-DQN algorithm (cf. Section 4.1). GCM-DQN can operate in planning domains with parametrized action spaces. It can either be used as a policy for probabilistic planning domains or, when using the state transition model, as a planner for deterministic planning domains (cf. Figure 2).

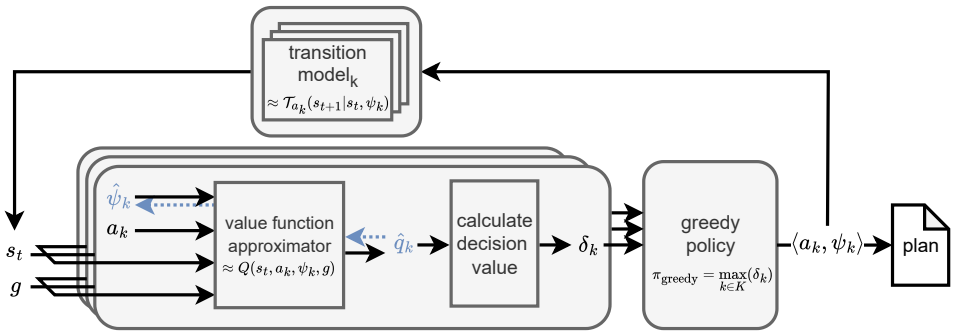

Figure 2: We introduce the GCM-DQN algorithm. A goal-conditioned and model-augmented DQN approach for planning in parametrized action spaces. GCM-DQN leverages on gradient-based optimization during planning time (marked in blue) to find (leastwise locally) optimal action-parameter tuples and uses a modular state transition model to predict state transitions.

4.1. Planning with Goal-Conditioned Model-augmented Deep Q-Networks

In this Section we provide an overview on the GCM-DQN algorithm (cf. Algorithm 1 and Figure 2). In its essence GCM-DQN is a goal-conditioned greedy policy, which is trained in an offline setting. Hence, the first step includes training the DQN Q_θ and the state transition models $\mathcal{F} = \{f_{a_k} | k = 1, ..., K\}$ using a dataset of recorded plans $\mathcal{D}$. During planning, GCM-DQN uses the *paramOpt* algorithm (cf. Algorithm 2) on Q_θ to calculate the optimal parameter $\tilde{\psi}_k^*$ for every action. To guide the selection of optimal action-parameter tuples, we calculate a decision value δ_k for each action. δ_k includes the Q value,

the weighted variance of the succeeding state var_k (cf. Equation 18), and a potential based shaping factor ω [22]:

$$\delta_k = Q_\theta(s_t, a_{k_t}, \tilde{\psi}^*_{k_t}, g) + \lambda_1 \text{var}_{k_t} + \lambda_2 \omega, \quad \lambda_1, \lambda_2 \in \mathbb{R}. \tag{8}$$

Using δ_k instead of the pure Q values counters, the selection of actions which would lead into non-permissible states, e.g., colliding with boundaries. A greedy policy π_{greedy} then picks the highest δ_k and adds the corresponding action-parameter $(a_k, \tilde{\psi}_k)$ tuple to the plan. By sampling from the associated state transition model f_{a_k} the next state s_{t+1} can be inferred and passed to the next iteration. The iterations stop, when s_{t+1} becomes a state within G (or $G \pm \varepsilon$, where ε is an error margin). In cases, in which there is no solution to the planning problem, a stopping criterion ζ can be introduced to bound the maximum number of iterations. The complete GCM-DQN algorithm is outlined in Algorithm 1. The following section introduce the extensions of GCM-DQN in detail.

Algorithm 1: GCM-DQN during planning

Require: $\mathscr{D}$ // recorded plans
s_0 // starting state
G // goal state(s)
ε, ζ // tolerance, max steps
$Q_\theta \leftarrow$ trainGCMDQN($\mathscr{D}$) // cf. Section 4.2
$\mathscr{F} \leftarrow$ trainSTM($\mathscr{D}$) // cf. Section 4.4
$\tau \leftarrow \emptyset$
$s \leftarrow s_0$
for $t \leftarrow 0$ **to** $\zeta - 1$ ***while*** $s \notin G \pm \varepsilon$ **do**
  $(a^*, \psi^*) \leftarrow \arg\max_{a \in A}$ DecisionValue$\big(Q_\theta(s, a, \text{ParamOpt}), s\big)$ // cf. Section 4.3
  append (a^*, ψ^*) to τ
  $s \leftarrow s_{t+1} \sim f_{a^*}(s, \psi^*)$
return τ // trajectory $((a_0, \psi_0), (a_1, \psi_1), \dots)$

4.2. Goal-Conditioned DQN for Parametrized Action Spaces

In this section, we describe our adaptions to DQN to allow using it for planning in planning domains with parametrized action spaces. We achieve this by including the goal state into the input of the DQN, thereby conditioning it on the goal state, and handling continuous per-action parameters via gradient-based optimization.

The original DQN uses a Neural Network to approximate the action value function $Q(s, a)$ of a domain [14], which describes the expected discounted return for taking action a in state s, and satisfies the Bellman equation in the optimal case

$$Q(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot|s_t, a_t)} [\mathscr{R}(s_t) + \gamma \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1})]. \tag{9}$$

For our application, we expand the classical Q-function's input with the goal state of the planning problem [15] and the parametrized actions, so that

$$Q(s, a) \;\rightsquigarrow\; Q(s, a_k, \psi_k, g), \tag{10}$$

where $a_k \in A$ is an action from the finite action set, ψ_k is an associated continuous parameter, and g is the goal state of the planning problem. For our updated Q-function, the Bellman equation becomes

$$Q(s_t, a_{k_t}, \psi_{k_t}, g) = \mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot|s_t, a_{k_t}, \psi_{k_t})} [\mathscr{R}_g(s_t) + \gamma \max_{k_{t+1} \in K} \arg\max_{\psi_{k_{t+1}} \in \Psi_k} Q(s_{t+1}, a_{k_{t+1}}, \psi_{k_{t+1}}, g)]. \tag{11}$$

As the inner maximization over $\psi_{k_{t+1}}$ is non-convex when Q is approximated by a Neural Network, solving it is intractable. Hence, we propose to leverage on global optimization algorithms for finding leastwise local optima for ψ and solve Equation 11 in two steps. In the first step, we find optimal action-parameter tuples for each action in the current state,

$$\psi_k^* = \arg\max_{\psi_k \in \Psi_k} Q(s, a_k, \psi_k, g) \quad \forall\, k \in K, \tag{12}$$

using projected gradient ascent (cf. Section 4.3). As we cannot guarantee a global optimum, we denote the resulting parameters with $\tilde{\psi}_{k_{t+1}}^*$. This first step allows us to reformulate Equation 11 as

$$Q(s_t, a_{k_t}, \psi_{k_t}, g) = \mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot|s_t, a_{k_t}, \psi_{k_t})} [\mathscr{R}_g(s_t) + \gamma \max_{k_{t+1} \in K} Q(s_{t+1}, a_{k_{t+1}}, \tilde{\psi}_{k_{t+1}}^*, g)], \tag{13}$$

which resembles the Bellman equation with a goal conditioning and an approximate inner maximization.

We *train* our goal-conditioned DQN Q_θ, with parameters $\theta \in \mathbb{R}$, for parametrized action spaces in an offline Reinforcement Learning setup, to cater the restrictions on of prohibitively expensive domain interactions in real-world planning domains. Therefore, we assume a training dataset of recorded plans $\mathscr{D} = \{\tau_j\}_{j=0}^J$. A major problem in offline Reinforcement Learning is the distributional shift between training data and the application domain [12]. We counter this problem by augmenting $\mathscr{D}$ with Hindsight Experience Replay [16], and Conservative Action Sampling [23] (cf. Figure 3). Hindsight Experience Replay augments the available dataset by sampling sub-traces from the recorded plans, relabeling the final state as the goal state [16]. Conservative Action Sampling also samples sub-trances from the recorded plans, however, labeling their final state as miss, therefore artificially creating negative samples for the dataset [23]. Using both augmentation techniques, results in the datasets $\tilde{\mathscr{D}}$ [16] and $\tilde{\bar{\mathscr{D}}}$ [23].

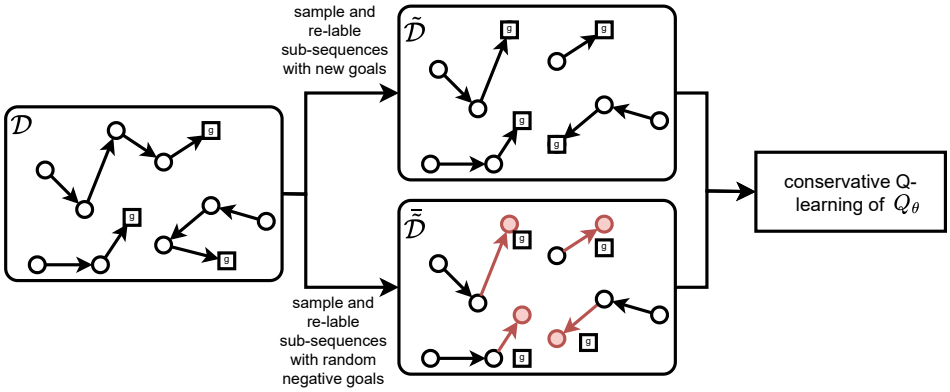

Figure 3: For training the DQN on little data, we use Hindsight Experience Replay [16] and Conservative Action Sampling [23] for augmenting our training dataset.

Following [14] we use an off-policy training setup, using an online network Q_θ and a target network Q_{θ^-}. During training, only the weights of Q_θ are updated via gradient descent, whereas the weights of Q_{θ^-} are copied from Q_θ every η steps. We use a composite loss function

$$\mathscr{L}_{CQL} = \mathscr{L}_Q + \mathscr{L}_P \tag{14}$$

consisting of the squared TD-loss $\mathscr{L}_Q$ [14] and a conservative penalty term $\mathscr{L}_P$ [17]. The conservative penalty term $\mathscr{L}_P$ helps to regularize Q_θ to overestimate Q-values of unseen or underrepresented actions [17]. We denote the squared TD-loss as

$$\mathscr{L}_Q = \mathop{\mathbb{E}}_{(s_t, a_{k_t}, \psi_{k_t}, r_t, s_{t+1}) \sim \tilde{\bar{\mathscr{D}}}} [r_t + \gamma(1 - d_t) \max_{k_{t+1} \in K} Q_{\theta^-}(s_{t+1}, a_{k_{t+1}}, \tilde{\psi}_{k_{t+1}}^*, g) - Q_\theta(s_t, a_{k_t}, \psi_{k_t}, g)]^2, \tag{15}$$

where $d_t \in \{0, 1\}$ indicates whether the plan at time t, so that $d_t = 1$, if $s_{t+1} \in G$. Following [17], we formulate the conservative penalty term as

$$\mathscr{L}_P = \alpha[\log(\sum_{k \in K} \frac{1}{M} \sum_{m=1}^{M} \exp(Q_\theta(s_t, a_{k_t}, \psi_{k_t}^{(m)}, g))) - Q_\theta(s_t, a_{k_t}, \tilde{\psi}_{k_t}^*, g)]. \tag{16}$$

where α is the trade-off factor between Bellman-fit and conservatism, $K = |A|$ is the number of discrete actions, and M is the number of parameter samples per action used in the log-sum-exp penalty. For our offline training, we draw M samples $\psi_{k_t}^{(m)}$ uniformly from the empirical pool of parameters for action a_k to approximate $\int_\psi e^{Q_\theta(s_t, a_{k_t}, \psi_{k_t}, g)} d\psi$.

Regarding $\mathscr{D}$, three edge cases must be considered: *(i)* $\mathscr{D}$ including no data, *(ii)* $\mathscr{D}$ including little data, and *(iii)* $\mathscr{D}$ including infinite data. In case *(i)*, where no data is available, Q_θ cannot be trained. Hence, data must be collected by random exploration or through sampling state transitions from the domain. Case *(ii)* describes the normal operation of GCM-DQN. We note that the higher the variance in the dataset, the better the approximation of Q_θ to the real Q. Case *(iii)* describes a special case, where all data are available. Given a large enough θ, this allows Q_θ to fit Q exactly.

4.3. Gradient-based Parameter Estimation

For finding the optimal parameters for an action in a given state, we propose to leverage on the differentiability of the DQN and use gradient ascent in a nested optimization loop for finding optimal parameters for a given action (cf. Equation 12). Therefore, we introduce the *paramOpt* algorithm, which draws inspiration from [24] and its applications in [9, 8].

The idea of *paramOpt* is to use the same algorithm, which is used to adapt the weights of Q_θ during training, for finding the optimal action-parameter tuples during execution. However, instead of optimizing the weights of the Q_θ, we optimize the parameter component ψ of its input. Therefore, we initialize the parameter component ψ with a guess $\hat{\psi}$, e.g., random numbers, zeros, or values from $\mathscr{D}$. After calculating $Q_\theta(s, a, \hat{\psi}, g)$, we use backpropagation to derive the gradient with respect to $\hat{\psi}$, allowing us to use gradient ascent with a learning rate β to update $\hat{\psi}$ in a direction which increases the Q-value. The optimization stops after the updates of the Q-value, Δ_Q, fall below a threshold ξ, returning the last update of $\hat{\psi}$ as $\tilde{\psi}^*$. Algorithm 2 summarizes our parameter estimation loop through input optimization.

Algorithm 2: paramOpt Gradient-Based Parameter Optimization

Require: s, a, g // state, action, goal
Q_θ // goal-conditioned DQN
β // learning rate
ξ // stopping threshold
$\hat{\psi} \leftarrow$ init() // initial guess
$\Delta_Q \leftarrow +\infty$
$Q^{(\text{prev})} \leftarrow -\infty$
while $\Delta_Q > \xi$ **do**
  $g_\psi \leftarrow \nabla_\psi Q_\theta(s, a, \hat{\psi}, g)$ // backprop wrt. parameters
  $\hat{\psi} \leftarrow \text{clip}_{[\psi_{\min}, \psi_{\max}]}(\hat{\psi} + \beta g_\psi)$ // projected gradient ascent
  $Q^{(\text{val})} \leftarrow Q_\theta(s, a, \hat{\psi}, g)$ // caclulate action value
  $\Delta_Q \leftarrow Q^{(\text{val})} - Q^{(\text{prev})}$
  $Q^{(\text{prev})} \leftarrow Q^{(\text{val})}$
return $\tilde{\psi}^* \leftarrow \hat{\psi}$ // optimized parameter

As we are using gradient ascent as optimization algorithm over the DQN, we cannot guarantee to find the true global optimum ψ^*. This is due to the non-convex shape of Q_θ. The result of the

optimization hence can be strongly dependent on the initialization of $\hat{\psi}$ and the learning rate β. As there are different options for initialization, e.g., zeros, ones, or random numbers, we suggest incorporating prior knowledge from the dataset, in the form of estimators like the mean over observed parameter settings as starting guesses.

Additionally, parameters are typically bound to value ranges, e.g., a temperature cannot fall below 0 Kelvin. To incorporate this, we use projected gradient ascent [25] during optimization, effectively clipping values that exceed the bounds. As one naïve solution for retrieving the bounds, we suggest iterating through the dataset $\mathscr{D}$ and collecting minima and maxima of each parameter.

4.4. Learning State Transition Dynamics

In real-world planning problems, directly interacting with the planning domain to predict action effects is rarely possible or prohibitively expensive [12]. Hence, planning requires a model of the state transition dynamics [1] which maps a current state s_t and parameters ψ_t to a successor state. In deterministic domains this is a function $f(s_t, \psi_t) = s_{t+1}$ (cf. Eq. 5); in probabilistic domains it is a conditional distribution $p(s_{t+1} \mid s_t, \psi_t)$ from which s_{t+1} is sampled (cf. Eq. 4).

Following the modular per–action factorization of PAMDP dynamics (cf. Eq. 3), we learn one transition model action, $\mathscr{F} = \{f_{a_k}\}_{k=1}^{K}$, each predicting the next state for action a_k given (s_t, ψ_t). Thereby, we use the same dataset $\mathscr{D}$, which is also used for training the DQN.

We propose to capture the stochasticity of probabilistic planning domains with a novel conditional latent-variable state transition model, inspired by [26]. Thereby, each per-action model comprises an encoder e_k and a decoder d_k part.

During training , the encoder processes the input s_t, ψ_t, and s_{t+1} into the parameters μ and σ of a latent posterior $q_e(z|s_t, s_{t+1}, \psi_t)$. Using the reparametrization trick, it samples $z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$. The decoder d_k reconstructs s_{t+1} from s_t, ψ_t, and z under a standard normal prior $p_d(z) = \mathcal{N}(0, I)$. As training criterion, we minimize the negative Evidence Lower Bound,

$$\mathscr{L} = - \mathop{\mathbb{E}}_{q_e(z|s_t, s_{t+1}, \psi_t)} [\log p_d(s_{t+1}|s_t, \psi_t, z)] + D_{\mathrm{KL}}(q_e(z|s_t, s_{t+1}, \psi_t) \,|\, p_d(z)), \tag{17}$$

where D_{KL} denotes the Kullback–Leibler divergence.

During planning , the encoder is discarded and only d_k is further used. Given the current state s_t and parameters $\tilde{\psi}_t^*$ (estimated with *paramOpt*), we draw $z \sim \mathcal{N}(0, I)$ and decode samples $\hat{s}_{t+1} = d_k(s_t, \tilde{\psi}_t^*, z)$. Boundaries and non-permissible states can be detected by analyzing the scalar variance var of $\hat{s}_{t+1}^{(n)}$ when sampling the z vector n times:

$$\mathrm{var} = \frac{1}{n-1} \sum_{i=1}^{n} \|\hat{s}_{t+1}^{(i)} - \bar{s}_{t+1}\|^2, \quad \bar{s}_{t+1} = \frac{1}{n} \sum_{i=1}^{n} \hat{s}_{t+1}^{(i)}, \tag{18}$$

A high variance indicates a high predictive uncertainty in $\hat{s}_{t+1}$, which indicates boundaries or non-permissible states, like obstacles.

For deterministic domains, the stochastic latent z can be omitted and d_k reduces to a standard Multilayer Perceptron.

5. Evaluation

We evaluate our GCM-DQN algorithm empirically against offline versions of state-of-the-art baselines for planning in parametrized action spaces [4, 6]. As performance metrics, we use the rate of successfully solved planning problems from a set of unseen planning problems. Therefore, we used domains with navigation problems and domains from the international planning competition's (IPC) reinforcement learning track [27] (cf. Figure 4). We hypothesize that ($H1$) GCM-DQN shows a higher performance

than the baselines, when trained on the same limited dataset of plans $\mathscr{D}$, and ($H2$) GCM-DQN longer maintains a higher performance than the baselines, when systematically reducing the number of samples in $\mathscr{D}$.

5.1. Experimental Setup

For setting up our experiments, we follow the experimental design guidelines for empirical Machine Learning research by Vranješ et al. [28]. We generate samples for the datasets $\mathscr{D}$ by running either an A^* search or JaxPlan [29] for randomly initialized planning problems of the chosen planning domains. We used Optuna [30] for hyperparameter optimization of GCM-DQN and the baselines to allow for a fair comparison. We repeated all experiments on eight different seeds to rule out lucky initializations. All code and datasets for replicating the experiments can be found under https://github.com/j-ehrhardt/gcmdqn. We used the following planning domains for evaluation:

Navigation Domains The navigation domains feature two-dimensional path finding problems in a continuous space with obstacles. The goal is to find a sequence of actions that lead from the start state to the goal state. There is a set of four actions - up, down, left, right - in which each action can be augmented with a plus minus ten-degree tilt. The step-width is fixed and collisions with the obstacles are forbidden. The planning problems are non-trivial, as the reward function is sparse and planners need to deal with linear and non-linear obstacles.

IPC Domains The IPC domains feature domains from the International Planning Competition's Probabilistic and Reinforcement Learning Track from 2023 [27]. We picked the reservoir, powergen and HVAC domains.

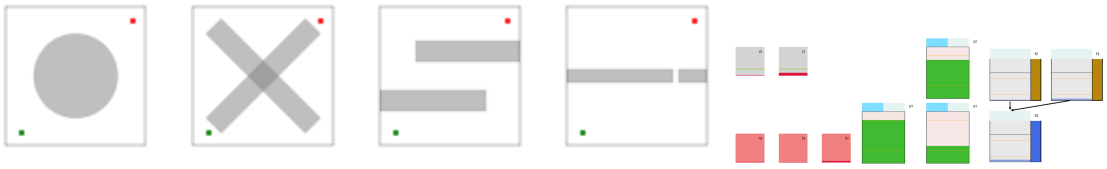

Figure 4: Evaluation domains from left to right **Navigation Domains**: Circle-domain, cross-domain, bars-domain, squeeze-domain (Exemplary start states are green and exemplary goal states are red). **IPC Domains**: HVAC-domain, PowerGen-domain, Reservoir-domain.

While the navigation domains have a stronger focus on the combinatorial aspect of finding a correct action to solve the planning problems, the IPC domains emphasize stronger on finding the correct parameters. As it is highly unrealistic that a learning algorithm on a scarce dataset $\mathscr{D}$ could match the classical evaluation metrics like optimality, soundness, efficiency, and completeness[1], we chose the planning success rate ρ, describing the number of successfully solved planning problems from a set of unseen test planning problems.

As **baselines** we used P-DQN [6] and P-DDPG [4] from literature, as, to our knowledge, there are no offline Reinforcement Learning algorithms for solving planning problems in PAMDPs. While P-DDPG is a policy based approach which is trained in an actor-critic setup [4], P-DQN is closer related to our approach using a DQN for evaluating different action-parameter tuples. However, instead of finding optimal parameter values via gradient-based search, it uses a Neural Network as heuristic for suggesting parameter values [6]. We transferred both baselines in an offline setting, using Conservative Q-learning, Hindsight Experience Replay, and potential-based shaping as for our algorithm.

[1]As our algorithm is grounded in the Bellman Equation, its solutions will converge to optimal, sound, and complete results with an infinitely large dataset $\mathscr{D}$. However, this is not its operational scenario. We hence do not consider very large datasets for evaluation.

Table 1
The table shows the mean success rate $\rho\pm$ standard deviation across eight seeds for our GCM-DQN algorithm and the baselines over different navigation and IPC planning domains.

domain name	P-DQN [6]	P-DDPG [4]	GCM-DQN (ours)
navigation – bars	0.7852 ± 0.2063	0.1952 ± 0.0049	**0.7922 ± 0.1141**
navigation – circle	0.8466 ± 0.1579	0.0653 ± 0.0053	**0.9375 ± 0.0250**
navigation – squeeze	0.5351 ± 0.1624	0.1326 ± 0.1429	**0.9405 ± 0.0308**
navigation – cross	0.7405 ± 0.1203	0.0680 ± 0.0165	**0.8497 ± 0.1171**
IPC – HVAC – instance0	0.1484 ± 0.2209	0.6045 ± 0.2212	**0.6669 ± 0.1687**
IPC – HVAC – instance1	0.5029 ± 0.0058	**0.5410 ± 0.0354**	0.5273 ± 0.0239
IPC – HVAC – instance2	0.4472 ± 0.0055	0.4492 ± 0.0072	**0.4492 ± 0.0055**
IPC – HVAC – instance3	0.1171 ± 0.0011	0.1221 ± 0.0058	**0.1299 ± 0.0102**
IPC – PowerGen – instance1	0.0000 ± 0.0000	**0.3691 ± 0.1268**	0.2910 ± 0.0147
IPC – PowerGen – instance2	0.0000 ± 0.0000	0.0000 ± 0.0000	**0.1171 ± 0.0263**
IPC – Reservoir – instance1	0.0009 ± 0.0027	0.5918 ± 0.1398	**0.6796 ± 0.1048**

5.2. Evaluating the Planning Performance of GCM-DQN

For evaluating the performance of GCM-DQN in comparison to the baselines, we created a training dataset $\mathscr{D}$ of 128 solved planning problems and a test dataset of 100 solved problems per domain. We ran a hyperparameter search with 64 trials for each algorithm and domain and subsequently tested each algorithm with the best hyperparameter setup on eight different seeds, to rule out lucky initialization. The results are reported in Table 1.

We hypothesized that GCM-DQN shows a higher performance than the baselines, when trained on the same limited dataset $\mathscr{D}$. For the navigation domains, our results indicate that GCM-DQN shows a higher mean planning success rate over the eight different seeds than the baselines, when trained on a limited dataset of 128 plans. For the IPC domains, either P-DDPG or GCM-DQN show the highest performance, with only narrow differences. As the IPC domains have a stronger emphasis on the parametrization than on the combinatorial action selection it is expectable that the Actor-Critic approach performs well in the IPC domains, while underperforming in the navigation domains. Overall, all algorithms show declining performance with increasing complexity of the planning domains. Yet, our GCM-DQN shows the most stable results, in comparison to the baselines.

5.3. Evaluating the Planning Performance of GCM-DQN on Succeedingly Scarce Data

The application scenario for GCM-DQN is planning under circumstances where only little data is available and interactions with the environment are not possible. For evaluating the behavior of GCM-DQN on scarce data, we trained the GCM-DQN and the baselines on succeedingly less samples in $\mathscr{D}$. Therefore, we created subsets of $\mathscr{D}$ containing $\{64, 32, 16, 8, 4, 2\}$ samples and trained GCM-DQN and the baselines on the hyperparameter settings from above. For each algorithm and dataset, we repeated the procedure on eight different seeds. Figure 5 shows the results for the navigation and IPC domains.

We hypothesized that GCM-DQN maintains a higher performance under progressive sample reduction compared to the baseline methods. For the navigation domains, we mostly could confirm this. GCM-DQN shows an increase in planning success rates, when increasing the number of plans in the training dataset. In the navigation domains, GCM-DQN gets overtaken by P-DQN in the lower sample area of the circle domains and the very closely in the higher sample area of the bars domain. Yet, it shows in general lower variance across the seeds, suggesting a more stable outcome. In the IPC domains, GCM-DQN and P-DDPG are consistently strong and stable, with an exception for GCM-DQN in the reservoir domain, while P-DQN performs low. Overall, GCM-DQN tends to improve, sometimes sharply in the higher sample area, while P-DDPG is competitive but more variable. P-DQN underperforms across the IPC domains.

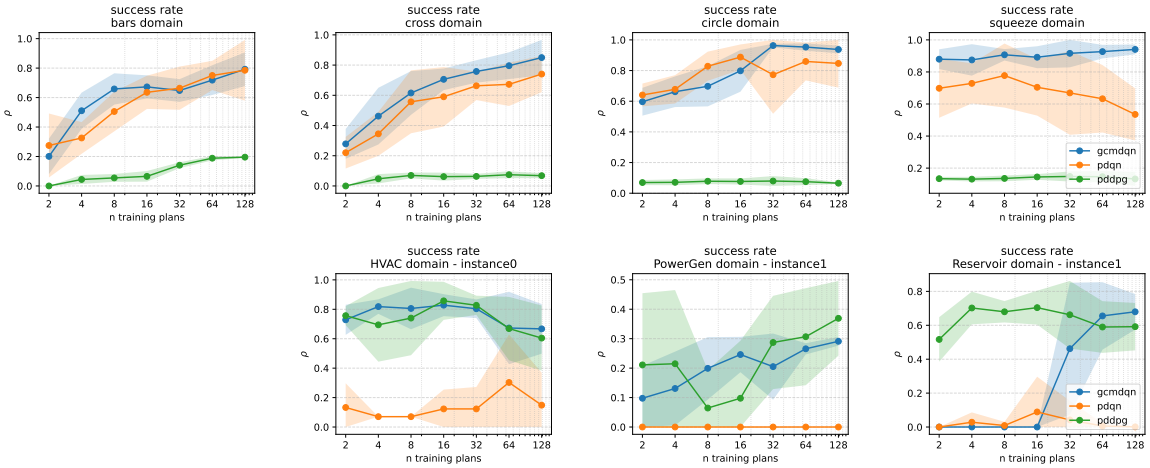

Figure 5: The figure shows the planning success rate for the navigation and IPC domains, when altering the number of plans in the training dataset. We repeated the experiments on eight different seeds. The shaded areas show the variance of the differently seeded runs.

6. Discussion

In the following, we discuss the findings from our Evaluation Section 5. We place special emphasis on discussing architectural limitations of GCM-DQN, the distributional shift of $\mathscr{D}$ to the application scenarios, and the implications of aleatoric uncertainty from latent factors in the planning domains.

Architectural Limitations of GCM-DQN Given the architecture we chose for our GCM-DQN algorithm, there are inherent limitations. Our gradient-based *paramOpt* function for estimating the parameters for actions can converge to local optima in the Q-function. Especially in complex, non-convex Q-functions, this poses a serious problem. Mitigation strategies could include ensemble approaches with differently seeded optimizers, multi-start optimization with different initial guesses, or a combination of both. Additionally, in essence, our GCM-DQN algorithm is one-step greedy (though implicitly operating on the expected returns of the DQN). Especially for domains in which long plans are necessary to reach a goal, the sparse reward signal of training data might lead to wrong results. Using the transition model for look-ahead methods, like Monte Carlo Tree Search, might result in better performance of the planner. Alternatively, a hierarchical perspective where GCM-DQN plans between intermediate goals might lead to increased performance with longer plans. As some hyperparameters, e.g., the α weight of Conservative Q-Learning or the number of Conservative Actions Samples, have a strong impact on the performance and stability of the planner, including them as parameters in the training loop to dynamically adapt the conservatism or data augmentation level of the model during training, might be a future improvement.

Data Quantity and Diversity The quantity and diversity of the training data in the training dataset $\mathscr{D}$ had a significant impact on the performance of the tested algorithms. Our results support the intuition that more and diverse data improves the approximation of the true Q-function and true transition dynamics. The planning success rate of our GCM-DQN algorithm continuously improved as the number of plans in $\mathscr{D}$ increased. All methods struggled in scenarios where only few samples in the training dataset were available. We deliberately focused on scarce data scenarios in our evaluation, as they reflect the real-world application of planners, where collecting more data and an interaction with the environment is prohivitively expensive. In this context, including Conservative Q-Learning an and Hindsight Experience Replay as mitigations for scarce data was important. Even though Hindsight Experience replay did not raise the mean planning success rates, it reduced the outcome variability and thus improved the reliability of GCM-DQN on small data. This implies that when working with

scarce data and the performance is insufficient, adding additional data to $\mathscr{D}$ may be more effective than tweaking the algorithms in isolation.

Distributional Shift in Offline Reinforcement Learning One of the core challenges in Offline Reinforcement Learning is the distributional shift between the training data and application scenarios [12]. Especially in the context of planning, the planner is likely to encounter state, action, parameter combinations that lie outside the support of the training data, which can lead to extrapolation errors. We mitigated this risk, using three mechanisms from the Offline Reinforcement Learning literature: Using Hindsight Experience Replay [16], Conservative Action Sampling [23], and Conservative Q-Learning [17]. Our results indicate that all measures improved training stability and planning performance.

Aleatoric Uncertainty from Latent Factors in the Planning Domain Real-world application scenarios for planners, e.g., industrial processes often show hidden factors and randomness that offline training cannot fully predict. I.e., in a manufacturing domain, tool wear out can alter a system's dynamics, introducing aleatoric uncertainty. Though our GCM-DQN approach attempts to accommodate stochasticity in its state transition models, systematic latent factor shifts over time will lead to mis-predictions of future transitions as the underlying transition dynamics changed. This limitation, however, is not unique to our approach but shared by all offline learning methods. Mitigating it could involve a periodic re-training with "fresh" data or designing the model to model these factors explicitly or in latent variables.

Evaluation Fairness of Offline Baselines Finally, we discuss the evaluation fairness of the employed baselines. The employed baselines P-DDPG [4] and P-DQN [6] were originally designed for online Reinforcement learning, where extensive interactions with the environment shapes the policy and DQNs. Conversely, we evaluated them in an offline setting. However, to ensure a fair evaluation with our GCM-DQN algorithm, we adapted both baselines to the offline setup, by incorporating the same techniques that we used in GCM-DQN to improve the training performance of the models. Namely, we used the same state transition models, Conservative Q-Learning, Hindsight Experience Replay, and Conservative Action sampling, creating a common and fair ground for evaluation.

7. Conclusion & Outlook

In this paper, we introduced the Goal-conditioned Model-augmented DQN algorithm (GCM-DQN), a model-augmented Offline Reinforcement Learning algorithm for planning in parametrized action spaces, where no model of the planning domain and only a limited dataset of recorded plans are available. GCM-DQN tackles three central challenges of planning with Reinforcement Learning in parametrized action spaces: *(i)* infinite branching of action-parameter tuples, *(ii)* goal-dependent reward functions, and *(iii)* substituting domain interactions with a model during planning time. To address the challenges, we introduce *paramOpt*, a novel gradient-based optimization algorithm over the DQN for finding the optimal parameters for an action in a state, a goal-conditioning of the DQN that allows for planning with changing and sparse reward functions, and a novel state transition model that allows to capture the inherent uncertainty in stochastic of probabilistic planning domains. We evaluate GCM-DQN against offline versions of two closely related algorithms. GCM-DQN shows significantly higher performance than the baselines, especially in data scarce scenarios. Future work will include the refinement of GCM-DQNs architecture and its application on real-world industrial planning scenarios.

Acknowledgement

This research as part of the project LaiLa and EKI is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr, which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU.

Declaration on Generative AI

Any use of generative AI in this manuscript adheres to ethical guidelines of IEEE for use and acknowledgement of generative AI. Each author has made a substantial contribution to the work, using LLMs exclusively for language refinement, formatting purposes, and for non-substantial coding, e.g., for creating plots.

References

[1] M. Ghallab, D. Nau, P. Traverso, Automated Planning and Acting, Cambridge University Press, 2016.

[2] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.

[3] W. Masson, P. Ranchod, G. Konidaris, Reinforcement learning with parameterized actions, Proceedings of the AAAI Conference on Artificial Intelligence 30 (2016). doi:10.1609/aaai.v30i1.10226.

[4] M. Hausknecht, P. Stone, Deep reinforcement learning in parameterized action space, 2016. doi:10.48550/ARXIV.1511.04143.

[5] R. Heesch, J. Ehrhardt, O. Niggemann, Integrating machine learning into an smt-based planning approach for production planning in cyber-physical production systems, in: Artificial Intelligence. ECAI 2023 International Workshops, Springer Nature Switzerland, Cham, 2024, pp. 318–331.

[6] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, H. Liu, Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space, 2018. doi:10.48550/ARXIV.1810.06394.

[7] J. Ehrhardt, R. Heesch, O. Niggemann, Learning process steps as dynamical systems for a subsymbolic approach of process planning in cyber-physical production systems, in: Artificial Intelligence. ECAI 2023 International Workshops, Springer Nature Switzerland, Cham, 2024, pp. 332–345.

[8] R. Heesch, A. Cimatti, J. Ehrhardt, A. Diedrich, O. Niggemann, A lazy approach to neural numerical planning with control parameters, in: European Conference on Artificial Intelligence (ECAI), 2024.

[9] G. Wu, B. Say, S. Sanner, Scalable planning with deep neural network learned transition models, Journal of Artificial Intelligence Research 68 (2020) 571–606. doi:10.1613/jair.1.11829.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2016. doi:10.48550/ARXIV.1509.02971.

[11] Y. Ma, T. Liu, B. Wei, Y. Liu, K. Xu, W. Li, Evolutionary Action Selection for Gradient-Based Policy Learning, Springer International Publishing, 2023, p. 579–590. doi:10.1007/978-3-031-30111-7_49.

[12] S. Levine, A. Kumar, G. Tucker, J. Fu, Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. doi:10.48550/ARXIV.2005.01643.

[13] M. Grand, D. Pellier, H. Fiorino, TempAMLSI: Temporal action model learning based on STRIPS translation, Proceedings of the International Conference on Automated Planning and Scheduling 32 (2022) 597–605. doi:10.1609/icaps.v32i1.19847.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533. doi:10.1038/nature14236.

[15] T. Schaul, D. Horgan, K. Gregor, D. Silver, Universal value function approximators, in: Proceedings of the 32nd International Conference on Machine Learning, volume 37 of *Proceedings of Machine Learning Research*, PMLR, Lille, France, 2015, pp. 1312–1320.

[16] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, Hindsight experience replay, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017.

[17] A. Kumar, A. Zhou, G. Tucker, S. Levine, Conservative q-learning for offline reinforcement learning, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 1179–1191.

[18] Z. Fan, R. Su, W. Zhang, Y. Yu, Hybrid actor-critic reinforcement learning in parameterized action space, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 2279–2285. doi:10.24963/ijcai.2019/316.

[19] B. Li, H. Tang, Y. Zheng, J. Hao, P. Li, Z. Wang, Z. Meng, L. Wang, Hyar: Addressing discrete-continuous action reinforcement learning via hybrid action representation, 2021. doi:10.48550/ARXIV.2109.05490.

[20] A. Tavakoli, F. Pardo, P. Kormushev, Action branching architectures for deep reinforcement learning, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018). doi:10.1609/aaai.v32i1.11798.

[21] G. Wu, B. Say, S. Sanner, Scalable planning with tensorflow for hybrid nonlinear domains, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017.

[22] A. Y. Ng, D. Harada, S. J. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, in: Proceedings of the Sixteenth International Conference on Machine Learning, ICML '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, p. 278–287.

[23] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. C. Julian, C. Finn, S. Levine, Actionable models: Unsupervised offline reinforcement learning of robotic skills, in: Proceedings of the 38th International Conference on Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 1518–1528.

[24] D. P. Kingma, S. Mohamed, D. J. Rezende, M. Welling, Semi-supervised learning with deep generative models, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, volume 27, Curran Associates, Inc., 2014.

[25] P. H. Calamai, J. J. Moré, Projected gradient methods for linearly constrained problems, Mathematical Programming 39 (1987) 93–116. doi:10.1007/bf02592073.

[26] K. Sohn, H. Lee, X. Yan, Learning structured output representation using deep conditional generative models, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 28, Curran Associates, Inc., 2015.

[27] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, J. Seipp, The 2023 international planning competition, AI Magazine 45 (2024) 280–296. doi:10.1002/aaai.12169.

[28] D. Vranješ, J. Ehrhardt, R. Heesch, L. Moddemann, H. S. Steude, O. Niggemann, Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research, in: 35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024), volume 125, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2024. doi:10.4230/OASIcs.DX.2024.7.

[29] M. Gimelfarb, A. Taitler, S. Sanner, Jaxplan and gurobiplan: Optimization baselines for replanning in discrete and mixed discrete-continuous probabilistic domains, Proceedings of the International Conference on Automated Planning and Scheduling 34 (2024) 230–238. doi:10.1609/icaps.v34i1.31480.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.