# Sampled-data Control of Probabilistic Boolean Control Networks: A Deep Reinforcement Learning Approach

Amol Yerudkar[a,*], Evangelos Chatzaroulas[b], Carmen Del Vecchio[c] and Sotiris Moschoyiannis[b]

[a]College of Mathematics and Computer Science, Zhejiang Normal University, Jinhua 321004, China.
[b]Department of Computer Science, University of Surrey, Guildford, GU2 7XH, United Kingdom.
[c]Department of Engineering, University of Sannio, Benevento 82100, Italy.

ARTICLE INFO

*Keywords*:
Sampled-data control (SDC)
Probabilistic Boolean control networks (PBCNs)
Markov decision processes (MDPs)
Deep reinforcement learning
Gene regulatory networks (GRNs)

ABSTRACT

The rise of reinforcement learning (RL) has guided a new paradigm: unraveling the intervention strategies to control systems with unknown dynamics. Model-free RL provides an exhaustive framework to devise therapeutic methods to alter the regulatory dynamics of gene regulatory networks (GRNs). This paper presents an RL-based technique to control GRNs modeled as probabilistic Boolean control networks (PBCNs). In particular, a double deep-$Q$ network (DD$Q$N) approach is proposed to address the sampled-data control (SDC) problem of PBCNs, and optimal state feedback controllers are obtained, rendering the PBCNs stabilized at a given equilibrium point. Our approach is based on *options*, i.e., the temporal abstractions of control actions in the Markov decision processes (MDPs) framework. First, we define options and hierarchical options and give their properties. Then, we introduce multi-time models to compute the optimal policies leveraging the options framework. Furthermore, we present a DD$Q$N algorithm: i) to concurrently design the feedback controller and the sampling period; ii) wherein the controller intelligently decides the sampled period to update the control actions under the SDC scheme. The presented method is model-free and offers scalability, thereby providing an efficient way to control large-scale PBCNs. Finally, we compare our control policy with state-of-the-art control techniques and validate the presented results.

## 1. Introduction

Markov decision processes (MDPs) [30] provide a framework for modeling and optimization of stochastic systems, including but not limited to manufacturing, automatic control, robotics, and gene regulatory networks (GRNs). Devising an optimal temporal decision in such systems is one of the important control problems that can be solved by resorting to MDP models. Such models are distinguished by i) state-transition dynamics; ii) a control policy that assigns an action to each state; iii) a transition cost from the current state to the next state. An MDP control problem is to design an optimal control policy that leads to the desired path, sequence of actions and states at the lowest cumulative cost. A well-known example of this type of problem is a reinforcement learning (RL) [34] problem, in which the cost at each step is revealed at the end of each transition. The most prevalent approaches to equip RL problems with an MDP as the underlying structure are dynamic programming [30], value iteration, policy iteration [30], and linear programming [8]. In addition, RL offers a model-free framework, such as $Q$-learning ($Q$L) [42], and solves a discrete-time optimal control problem modeled as an MDP.

In this paper, a model-free RL framework is utilized to design an optimal control policy that feedback stabilizes probabilistic Boolean control networks (PBCNs) [25] modeled as an MDP. Shmulevich *et al.* [33] first introduced probabilistic Boolean networks to model GRNs with stochastic uncertainty, which then extended to PBCNs with Boolean control inputs. PBCNs are a collection of Boolean control networks (BCNs) switching randomly between constituent BCNs with a certain probability distribution. With the probabilistic switching law, PBCNs show innate capabilities to study various regulatory functions of GRNs in the presence of random perturbations. So far many control problems of PBCNs (and BCNs) have been studied in model-based settings, for example, controllability and observability [17, 21], stabilization [15, 11, 48], output regulation [14, 47, 50], detectability [39, 10], optimal control [43, 9, 13], and disturbance decoupling [16, 31].

*Corresponding author
amol_yerudkar@hotmail.com (A. Yerudkar); e.chatzaroulas@surrey.ac.uk (E. Chatzaroulas); c.delvecchio@unisannio.it (C.D. Vecchio); s.moschoyiannis@surrey.ac.uk (S. Moschoyiannis)
ORCID(s): 0000-0003-3994-3842 (A. Yerudkar); 0000-0001-6937-9678 (C.D. Vecchio); 0000-0002-0164-8322 (S. Moschoyiannis)

Stabilization is a fundamental control problem studied by systems biologists and control theorists from a practical point of view. For example, in disease treatment, modeling the development of a disease in terms of tumor (diseased cell) growth and developing therapeutic intervention strategies to steer the organelle's growth towards a healthy state by eradicating the tumor. The feedback stabilization problem of GRNs modeled as PBCNs has been extensively investigated in the literature. The majority of the results are obtained in model-based framework developed with the help of the semi-tensor product (STP) [7] of matrices. For example, authors in [41, 36] studied the finite-time stabilization of PBCNs. Zhou *et al.* [49] and Yang *el al.* [45] addressed the asymptotical feedback stabilization of PBCNs, which was then extended to output tracking problem in [6]. Recently, robust state stability and the effect of stochastic function perturbation on the same were investigated in [40] and [15], respectively. The controller developed in the above-mentioned studies is a traditional state feedback controller that is active at all time instants. Such continuous application of control actions may have detrimental effects on GRNs. To mitigate this issue, the sampled-data control (SDC) has been designed for PBCNs in [22, 20, 44], where the feedback controller is updated after each sampling period. As a result, the SDC can reduce the amount of controller updates while delivering the same effect as a traditional controller. However, there are several shortcomings of the STP-based controller design techniques: i) computationally demanding; ii) limited applicability to small-scale networks; iii) system model is required. Further, for large-scale GRNs, the network models are unavailable. These difficulties limit the application of model-based strategies for controlling PBCNs, instigating interest in finding novel model-free methods to control GRNs modeled as PBCNs.

By following this stream of research, we consider an MDP as the underlying structure to model PBCNs, and study the SDC of PBCNs by using double deep-$Q$ network (DD$Q$N) in a model-free framework. In particular, we update DD$Q$N algorithm using prioritized experience replay (PER) and well-known RL concept called *options* to design a state feedback controller and sampled period to update the controller in tandem. The options framework provides a temporal abstraction of control actions which forms the basis for designing an SDC strategy in a model-free setting. Recently, several model-free algorithms have been proposed to solve various control-theoretic problems of PBCNs. For example, a DD$Q$N with PER algorithm has been applied to the controllability of probabilistic Boolean networks by authors in [27] and of Kauffman's standard BNs in [26]. In addition, authors in [12] have applied rule-based $Q$L, in the form of an extended classifier system to the controllability of the BN model of the cell-cycle fission yeast GRN. Another $Q$L-based algorithm to design state-flipped control of BNs is presented in [23]. Acernese *et al.* presented $Q$L algorithm to find a shortest-path state feedback controller [3] and DD$Q$N algorithm for solving output tracking problem [1] of PBCNs. Further, deep $Q$ network algorithm [4] and random forest-based $Q$L algorithm [5] were presented to design aperiodic sampled-data controller and shorted path controller, respectively, for PBCNs. Recently, a self-triggering control co-design algorithm has been presented in [2] wherein the aim of the controller was to minimize the communication with the system (i.e., unknown environment) by keeping the same control action and stabilize the system to a given equilibrium point. Nonetheless, a generalized scalable algorithm that can provide a versatile control design framework is still missing and deserves further investigation. Motivated by the above discussion, we present a comprehensive model-free algorithm to control large-scale PBCNs. The main contributions of the paper are as follows:

i) We adopt the MDP framework to represent the PBCN dynamics and state the model-free SDC problem for feedback stabilizing GRNs modeled as PBCNs;
ii) We introduce options, hierarchical options, and augmented action-space to setup the SDC problem. Further, we prove various mathematical properties of options with MDP framework and derive the Bellman optimality equation for the same.
iii) We present modified DD$Q$N algorithm by utilizing the concepts introduced in ii) and design an SDC strategy to stabilize the PBCNs at a given equilibrium point. Under this strategy, the presented algorithm concurrently designs the controller and sampled period. Our algorithm generalises to different networks and can easily handle large-scale PBCNs to solve control-theoretic problems in model-free framework.

The rest of the paper is organized as follows. Section 2 introduces definitions and basics of PBCNs, MDPs, $Q$L, and DD$Q$Ns. Section 3 presents the main results of the paper. The SDC problem is defined first. Then options are introduced, followed by hierarchical options. Further, a multi-time model is presented to compute optimal SDC policy utilizing the options framework. Next, a DD$Q$N algorithm with PER and options is presented that delivers an optimal SDC policy (i.e., control action and sampled period in tandem) to stabilize unknown PBCNs at a given equilibrium point. In Section 4 simulation results are given wherein three different PBCN models of GRNs are considered to verify the presented DD$Q$N algorithm. Finally, Section 5 concludes the paper.

## 2. Preliminaries

In this section, we introduce PBCNs and MDPs. Then, we extend the discussion to a model-free RL framework for $Q$L and DD$Q$N methods which will be utilized to solve the control problem of PBCNs.

**Notation.** $\mathbb{R}$, $\mathbb{Z}$ and $\mathbb{Z}_+$ denote the sets of real numbers, positive integers and nonnegative integers, respectively. $\mathcal{B} := \{0, 1\}$, and $\mathcal{B}^n := \underbrace{\mathcal{B} \times \ldots \times \mathcal{B}}_{n}$. We denote by $\neg$, $\wedge$, $\vee$, the logical operators Negation, And, Or, respectively. $\mathbb{E}[\cdot]$ is the expected value operator. $\top$ denotes the transpose of a vector or matrix. $|\cdot|$ represents the cardinality of a set.

### 2.1. Probabilistic Boolean Control Networks

A PBCN with $n$ nodes and $m$ control inputs is defined as follows:

$$\mathcal{X}^i(t+1) = f_i^{\sigma(t)}\left(\mathcal{U}(t),\ \mathcal{X}(t)\right),\ i = 1, \ldots, n, \tag{1}$$

where $\mathcal{X}(t) = \mathcal{X}_t := \left(\mathcal{X}^1(t), \ldots, \mathcal{X}^n(t)\right) \in \mathcal{B}^n$ and $\mathcal{U}(t) = \mathcal{U}_t := \left(\mathcal{U}^1(t), \ldots, \mathcal{U}^m(t)\right) \in \mathcal{B}^m$ are the state and input variables at time $t \in \mathbb{Z}_+$, respectively. The logical functions $f_i \in \mathcal{F}_i = \{f_i^1,\ f_i^2,\ \ldots,\ f_i^{l_i}\} : \mathcal{B}^{n+m} \to \mathcal{B}$, $i = 1, \ldots, n$, are randomly chosen with probability $\{\mathsf{P}_i^1,\ \mathsf{P}_i^2,\ \ldots,\ \mathsf{P}_i^{l_i}\}$, where $\sum_{j=1}^{l_i} \mathsf{P}_i^j = 1$ and $\mathsf{P}_i^j \geq 0$. We denote by $\Lambda = \prod_{i=1}^{n} l_i$, the total number of sub-networks of (1). The switching signal $\sigma(t) \in [1,\ \Lambda]$ is an independently and identically distributed (i.i.d.) process that governs switching among the sub-networks. Given an initial state $\mathcal{X}(0) \in \mathcal{B}^n$ and a control sequence $\boldsymbol{\mathcal{U}}_t := \{\mathcal{U}(\cdot)_{[0,\,t-1]}\}$ in the discrete time interval $[0,\ t-1]$, denote the solution to PBCN (1) by $\mathcal{X}\left(t;\ \mathcal{X}(0),\ \boldsymbol{\mathcal{U}}_t\right)$. A state $\mathcal{X}(0) = \mathcal{X}_0 \in \mathcal{B}^n$ is called an equilibrium point if, there exists a control $\mathcal{U}(0) \in \mathcal{B}^m$ such that $\mathsf{P}\{\mathcal{X}(1;\ \mathcal{X}(0),\ \mathcal{U}(0)) = \mathcal{X}(0)\} = 1$.

**Definition 2.1 ([3]).** *A PBCN (1) is said to be asymptotically stabilizable at a given equilibrium point $\mathcal{X}_e \in \mathcal{B}^n$ in distribution, if for every $\mathcal{X}_0 \in \mathcal{B}^n$ there exists $\boldsymbol{\mathcal{U}}_t$ such that $\lim_{t\to\infty} \mathsf{P}\{\mathcal{X}(t;\ \mathcal{X}_0,\ \boldsymbol{\mathcal{U}}_t) = \mathcal{X}_e\} = 1$.*

In the following, we present MDPs, which have been adopted as a popular framework for modeling PBCNs.

### 2.2. Markov Decision Processes

A discrete-time Markov decision process (MDP) [30] is a tuple $(\mathbf{X},\ \mathbf{U},\ \mathbf{P},\ \mathbf{G})$, where $\mathbf{X}$ is the state-space, $\mathbf{U}$ is the action-space. $\mathbf{P} : \mathbf{X} \times \mathbf{U} \times \mathbf{X} \to [0, 1]$ is the function of *state-transition probabilities* describing the conditional probability $\mathbf{P}^{u_t}_{x_t,x_{t+1}} = \mathsf{P}\{x_{t+1}|x_t, u_t\}$ of transitioning from $x_t$ to $x_{t+1}$ when $u_t$ is taken, for each state $x_t \in \mathbf{X}$ and action $u_t \in \mathbf{U}$, where $t \in \mathbb{Z}_+$ is the discrete time-step, and state and action values at $t$ are $x_t$ and $u_t$, respectively. Moreover, let $\mathbf{G} : \mathbf{X} \times \mathbf{U} \times \mathbf{X} \to \mathbb{R}$ denote the cost function; given $x_t$ and $u_t$ is selected, the expected cost paid after transitioning to state $x_{t+1}$ is $\mathbf{G}^{u_t}_{x_t,x_{t+1}} = \mathbb{E}[g_{t+1}|x_t, u_t]$, with $g_{t+1} = g_{t+1}(x_t,\ u_t,\ x_{t+1})$. In an infinite-horizon discounted cost setting, the objective is to find a policy $\pi : \mathbf{X} \to \mathbf{U}$ that minimizes the expected total discounted cost, i.e.,

$$\min_{\pi} \mathbb{E}_{\mathbf{P}}\left[\sum_{i=t+1}^{\infty} \gamma^{i-t-1} g(x_i, u_i)\right], \tag{2}$$

where $\gamma \in [0, 1)$ is the discount factor weighting costs along the trajectories, $\mathbb{E}_{\mathbf{P}}[\cdot]$ is the expected value operator w.r.t. the dynamics. If $\pi$ admits only one control action for each state with probability 1 (w.p.1), it is called deterministic policy, i.e., of the form $\mu(x_t)$, mapping states $x_t$ into controls $u_t = \mu(x_t), \forall x_t$.

Given an initial state $x_0$ and following the acting behavior $\pi$, the value function of the state $x_0$ is defined in terms of the expected future cost as:

$$v_\pi(x_0) := \mathbb{E}_{\mathbf{P}}\left[\sum_{i=1}^{\infty} \gamma^{i-1} g(x_i, u_i)\middle| x_0\right],\ \text{for all } x_0 \in \mathbf{X}. \tag{3}$$

Similarly, let the action-value function $q_\pi(x_t, u_t)$ be defined as the cost-to-go function when starting from state $x_0$, performing action $u_0$, and following policy $\pi$ thereafter. A fundamental property of $q_\pi(x_t, u_t)$ is that it satisfies the recursive Bellman equation [34], of the form

$$q_\pi(x_t, u_t) = \sum_{x\in\mathbf{X}} \mathsf{P}^{u_t}_{x_t,x}\left[\mathbf{G}^{u_t}_{x_t,x} + \gamma \sum_{u\in\mathbf{U}} \pi(u|x) q_\pi(x, u)\right], \tag{4}$$

for all $x_t \in \mathbf{X}$ and $u_t \in \mathbf{U}$. This equation has a unique fixed point at which the action-value function is minimum, denoted by $q^*(x_t, u_t) := q_{\pi^*}(x_t, u_t)$, where $\pi^*$ is the policy that minimizes (2). Given a solution $q^*(x_t, u_t)$, one can obtain an optimal deterministic policy as:

$$\mu^*(x_t) := \arg\min_{u \in \mathbf{U}} q^*(x_t, u), \text{ for all states } x_t \in \mathbf{X}. \tag{5}$$

It is worth highlighting that the action-value function remains the same in the case of multiple optimal deterministic policies. Further, we assume that the agent has no knowledge of the transition probability distribution, which is realistic for large-scale GRNs. Thus, we resort to an approach, wherein the expected value of a state-action pair $q_\pi$ is approximated with an estimation $Q : \mathbf{X} \times \mathbf{U} \to \mathbb{R}$ computed along trajectories. This idea is known as temporal-difference (TD) learning [34].

## 2.3. $Q$-learning Algorithm

$Q$L is a model-free RL algorithm introduced by [42]. It is an off-policy TD control algorithm, where the agent finds an optimal policy to maximize the cumulative reward by exploring the unknown environment. In $Q$L, the only information available to the agent is the state-space and possible actions. A value is assigned to each state-action pair, namely a $Q$-value, which, for a given state, represents the quality of the action. Given a pair $(x_t, u_t)$, $q_\pi(x_t, u_t)$ is estimated as $Q(x_t, u_t) = g_{t+1} + \gamma\, Q_\pi(x_{t+1}, \mu(x_{t+1}))$, where $x_t \in \mathbf{X}$ and $u_t \in \mathbf{U}$. The estimates are improved using Bellman equation [34] as follows

$$Q_{t+1}(x_t, u_t) = Q_t(x_t, u_t) + \alpha_t \underbrace{\left[g_{t+1} + \gamma \min_{u \in \mathcal{U}} Q_t(x_{t+1}, u) - Q_t(x_t, u_t)\right]}_{TDE_{t+1}}, \tag{6}$$

where:

- $TDE_{t+1}$: difference between the old estimated value $Q_t(x_t, u_t)$ and the new estimate $[g_{t+1} + \gamma \min_u Q_t(x_{t+1}, u)]$;
- $\alpha_t$: learning rate or step-size rule of convergence and $0 < \alpha_t \leq 1$.

The agent explores the environment for a sufficiently large number of episodes[1]. Watkin *et al.* [42] proved the convergence of $Q$L to an optimal value $Q^*(x_t, u_t) := q^*(x_t, u_t)$, $\forall x_t \in \mathbf{X}$, $\forall u_t \in \mathbf{U}$, w.p.1. Sufficient exploration of the environment is achieved by choosing the actions $u_t$ in a proper way, e.g., by following an $\epsilon$-greedy policy, i.e., choosing a greedy action $u_t = \arg\min_{u \in \mathbf{U}} Q_t(x_t, u)$ w.p.$(1-\epsilon)$, and a random action $u_t = \texttt{rand}(\mathbf{U})$ w.p.$\epsilon$, thereby devising better optimal policy, where $\texttt{rand}(\cdot)$ is the discrete uniform distribution and $0 < \epsilon \leq 1$.

## 2.4. Double Deep-$Q$ Network

In $Q$L method, a $Q$-table (or look-up table) is created which performs well for environments with small state-space. For large-scale GRNs, an exponential increase in the state-space makes $Q$L intractable [3]. To mitigate this issue, artificial neural networks (ANNs) were introduced as function approximators that can take vector input (state information) and learn to map them to $Q$-values for all possible actions. Since ANNs are employed to create a parameterized model that estimates $Q$-values, this method is called deep $Q$-learning. Particularly, in deep $Q$-learning ANN provides an estimation $Q : \mathbf{X} \times \mathbf{U} \times \mathcal{W} \to \mathbb{R}$ of $q_\pi(\cdot, \cdot)$, where $\mathcal{W}$ represents the set of tunable parameters responsible for the quality of the approximation. Given $x_t$ and a set of values for $\mathcal{W}$, for every possible action $u$, deep $Q$-network provides the output vector of approximated action-values as follows

$$Q(x_t, u, \mathcal{W}) = \psi^{(L)}(W^{(L)}(\dots \psi^{(2)}(W^{(2)}\psi^{(1)}(W^{(1)}x_t + b^{(1)}) + b^{(2)}) \dots) + b^{(L)}), \tag{7}$$

where $L$ is the number of layers of the network, $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$ are the ANN weights, $b^{(l)} \in \mathbb{R}^{n^{(l)}}$ is the bias vector and $\psi^{(l)}$ is the activation function in the $l$-th layer. We denote by $\{n^{(l)}\}_{l=0}^{L}$, the number of neurons for each layer, i.e., $n^{(L)} = |\mathbf{U}|$. The aim of D$Q$Ns is to learn the network parameters $\mathcal{W} = \{W^{(l)}, b^{(l)}\}_{l=0}^{L}$ such that $Q(x_t, u_t, \mathcal{W})$

[1] An episode is a structured agent-environment (i.e., controller-unknown system, respectively) interaction process that aims to improve the agent's understanding of the environment by imparting information (in the form of a reward) to the agent.

converges to an optimal $q^*(x_t, u_t)$, for all $x_t \in \mathbf{X}$ and $u_t \in \mathbf{U}$. Thus, given a tuple $(x_t,\ u_t,\ g_{t+1},\ x_{t+1})$, the parameters of the network are updated to minimize the following differentiable loss function called the Bellman error

$$\mathscr{L}(\mathcal{W}) = \frac{1}{2}\left|\left|Q(x_t, u_t, \mathcal{W}) - y_{t+1}\right|\right|^2, \tag{8}$$

i.e., the error between the estimated $Q$-value and the target value $y_{t+1} = g_{t+1} + \gamma \min_{u\in\mathcal{U}} Q(x_{t+1}, u, \mathcal{W})$ that this is the parameterized form of *TDE* in (6). Then, the stochastic gradient descent (SGD) method can be used to update $\mathcal{W}$ as follows

$$\mathcal{W} = \mathcal{W} - \alpha\nabla_{\mathcal{W}}\mathscr{L}(\mathcal{W}) = \mathcal{W} - \alpha\big[Q(x_t, u_t, \mathcal{W}) - y_{t+1}\big]\nabla_{\mathcal{W}}Q(x_t, u_t, \mathcal{W}). \tag{9}$$

The convergence of deep $Q$-network is a challenging issue which mainly occurs due to i) samples are uncorrelated in the SGD method. However, according to (9), the network parameters are updated sequentially over tuples $\{(x_t, u_t, g_{t+1}, x_{t+1})\}_{t=0,1,\ldots}$, that, in general, are temporally correlated. This leads to locally over-fit data to each region of the state-space; ii) $y_{t+1}$ in (9) depends on the ANN parameters $\mathcal{W}$, and consequently its value changes over time-steps.

An important feature added to deep $Q$-network is experience replay [19], which can alleviate the stability issues. Specifically, let $\mathcal{M}$ be a data-set of transitions $\{(x_j,\ u_j,\ g_{j+1},\ x_{j+1})\}_{j=0,1,\ldots,|\mathcal{M}-1|}$ stored by the agent. Then, a mini-batch $\mathcal{M}_1$, where $\mathcal{M}_1 \subseteq \mathcal{M}$, can be uniformly sampled by following the loss function

$$\mathscr{L}'(\mathcal{W}) = \mathbb{E}_{(\bar{x}_j,\bar{u}_j,\bar{g}_{j+1},\bar{x}_{j+1})\in\mathcal{M}}\left[\sum_{j=0}^{|\mathcal{M}_1-1|}\mathscr{L}_j(\mathcal{W})\right], \tag{10}$$

where henceforth the notation $\bar{(\cdot)}_j$ denotes the $j$-th experience and not to the value at time-step $j$, and $\mathscr{L}_j(\cdot)$ is the loss function (8) computed on the experience $j$. Thus, if $\mathcal{M}$ is large, experience replay uses an exploratory policy to sample independent transitions. Moreover, the prioritized experience replay (PER) was introduced in [32], following the idea that some experiences can be more informative than others. Let $\bar{\phi}_{j+1} := Q(\bar{x}_j, \bar{u}_j, \mathcal{W}) - \bar{y}_{j+1}$ be the error of an experience $(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1})$, where $\bar{x}_j \in \mathbf{X}$ and $\bar{u}_j \in \mathbf{U}$. Consequently, the probability of sampling such tuple from $\mathcal{M}$ can be computed with the following proportional prioritization criterion

$$\mathsf{P}\{(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1}) \in \mathcal{M}\} = \frac{(\bar{p}_{j+1})^\omega}{\sum_{k=0}^{|\mathcal{M}-1|}(\bar{p}_{k+1})^\omega}, \tag{11}$$

where $\bar{p}_{j+1} := |\bar{\phi}_{j+1}| + \zeta$, $\omega$ decides the magnitude of prioritization — $\omega = 0$ gives a uniform experience replay —, and $\zeta$ is a small positive constant. However, PER may create a bias in learning process due to more often sampling of the experiences with high errors. To compensate for this bias, the following weighted importance-sampling (W-IS) [24] can be used

$$\bar{\theta}_{j+1} = \left(\frac{1}{|\mathcal{M}|}\frac{1}{\mathsf{P}\{\bar{(\cdot)}_j \in \mathcal{M}\}}\right)^\beta, \tag{12}$$

where $\mathsf{P}\{\bar{(\cdot)}_j \in \mathcal{M}\}$ is the probability of sampling the experience $(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1})$ from $\mathcal{M}$, and $\beta$ is a parameter used to anneal the amount of importance sampling over the episodes.

Authors in [38] suggested the implementation of DD$Q$N to mitigate the issue that the target value $y_{t+1}$ changes over time-steps in (9). The traditional Double $Q$L [37] addresses the overestimation problem of $Q(\cdot,\cdot)$ linked to the "min" operator in (6), thereby preventing the instabilities to propagate quickly by using a double estimator, namely $Q^A$ and $Q^B$. One of the estimators is selected randomly for the evaluation of $u_t$ at each time-step $t$, followed by an update in terms of the other estimator. The DD$Q$N extends the technique of two networks as follows: i) an online network with parameters $\mathcal{W}$, responsible for selecting the policy; ii) a target network with parameters $\mathcal{W}^-$, used to evaluate the current action. Thus, given a mini-batch $\{(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1})\}_{j=0,1,\ldots,|\mathcal{M}_1-1|}$, the parameter update (9) in PER based DD$Q$N with W-IS is replaced by

$$\mathcal{W} = \mathcal{W} - \alpha\sum_{j=0}^{|\mathcal{M}_1-1|}\bar{\theta}_{j+1}\bar{\phi}'_{j+1}\nabla_{\mathcal{W}}Q(\bar{x}_j, \bar{u}_j, \mathcal{W}), \tag{13}$$

where

$$\bar{\phi}'_{j+1} = [Q(\bar{x}_j, \bar{u}_j, \mathcal{W}) - \bar{g}_{j+1} - \gamma Q(\bar{x}_{j+1}, \arg\min_{\bar{u}\in\mathcal{U}} Q(\bar{x}_{j+1}, \bar{u}, \mathcal{W}), \mathcal{W}^-)]. \tag{14}$$

Authors in [38] proposed one way of performing the update rule of the target network, wherein $\mathcal{W}^-$ remains constant over time-steps and updated as $\mathcal{W}^- = \mathcal{W}$ after every $k$ iterations. Another possibility presented in [18], considers the use of the soft update through Polyak averaging, i.e., $\mathcal{W}^- = (1-\tau)\mathcal{W}^- + \tau\mathcal{W}$, with $0 < \tau \leq 1$ a parameter that constrains the target ANN to change slowly, greatly improving the stability of learning.

## 3. Sampled-data Control Design for PBCNs

In this section, we formulate the SDC problem of PBCNs and find an optimal solution in a model-free context. In particular, we present a modified DD$Q$N algorithm that utilizes the temporal abstraction of actions, namely *options*, and devises a feedback control law as well as the sampling periods to stabilize the PBCNs at a given equilibrium point.

### 3.1. Sampled-data Control using DD$Q$N and Options

Let the PBCN (1) be the environment unknown to the agent. Further, knowledge of the equilibrium point $\mathcal{X}_e \in \mathcal{B}^n$ and the dimensions of the state-space are available to the agent. Under the SDC implementation, the control input $\mathbf{u}(t)$ is obtained as

$$\mathbf{u}(t) = \mathcal{K}\mathbf{x}(t_k), \quad t \in [t_k,\ t_{k+1}),\ k \in \mathbb{Z}_+, \ \text{ and } \ t_{k+1} = t_k + \tau(\mathbf{x}(t_k)), \tag{15}$$

where the state $\mathbf{x}(t_k)$ and $\mathbf{u}(t)$ are in canonical vector form[2], and $\mathcal{K} \in \mathcal{L}_{2^m\times 2^n}$ is gain matrix of the sampled-data state feedback controller. It is worth noting that $\mathcal{K}$ is not necessarily unique. Further, $\{t_k\}_{k\in\mathbb{Z}_+}$ is the sampling instant, i.e., the time instant at which the sampled-data controller (15) is updated. With $t_0 = 0$, and $\tau(\mathbf{x}(t_k)) \in \mathbb{N}$ is called the sampling period, i.e., $\tau(\mathbf{x}(t_k)) = t_{k+1} - t_k$. In this paper, during the sampling period $[t_k,\ t_{k+1})$, the control $\mathbf{u}(t)$ is held constant allowing no communication between the system and the controller.

The aim of this paper is the co-design of:

i. a sampled-data state feedback gain matrix $\mathcal{K}$ that stabilizes the closed-loop system at a given equilibrium point;
ii. an SDC scheme i.e., a map $\tau : \mathcal{B}^n \to \mathbb{N}$ determining the next update time $t_{k+1}$ of the controller $\mathcal{K}$ as a function of the state $\mathbf{x}(t_k)$ at the time $t_k$.

The following assumption is made to achieve the above-mentioned SDC co-design aim in a model-free framework.

**Assumption 1.** *Definition 2.1 is valid for the unknown PBCN of the form* (1)*.*

**Definition 3.1.** *Given an equilibrium point $\mathcal{X}_e \in \mathcal{B}^n$, the PBCN* (1) *unknown to the agent is asymptotically sampled-data stabilized to $\mathcal{X}_e$ with probability one if there exists a feedback control law* (15) *such that*

$$\lim_{t\to\infty} \mathsf{P}\{\mathcal{X}(t;\ \mathcal{X}_0,\ \boldsymbol{\mathcal{U}}_t) = \mathcal{X}_e\} = 1, \quad \forall \mathcal{X}_0 \in \mathcal{B}^n,$$

*where $\boldsymbol{\mathcal{U}}_t$ denotes the control sequence generated by the feedback* (15)*.*

In the following, we introduce *options* that are studied in the literature to solve various planning and control problems in the RL framework. Here, we utilize options with the DD$Q$N model to devise an SDC co-design strategy.

[2] $\delta^i_n$ denotes the $i$-th canonical vector of size $n$, and $\mathcal{L}_n$ the set of all $n$-dimensional canonical vectors. A matrix $L = [\delta^{i_1}_m \ \dots \ \delta^{i_n}_m]$ for suitable indices $i_1, \dots, i_n \in \{1, 2, 3, \dots, m\}$, is called logical matrix, and briefly expressed as $L = \delta_m[i_1\, i_2\, \dots\, i_n]$. $\mathcal{L}_{m\times n}$ is the set of all $m \times n$ logical matrices. A bijective correspondence between a Boolean variable $\mathcal{X} \in \mathcal{B}$ and a vector $\mathbf{x} \in \mathcal{L}_2$ is defined by the relationship $\mathbf{x} = \begin{bmatrix}\mathcal{X}\\ \neg\mathcal{X}\end{bmatrix}$. A bijective correspondence between $\mathcal{B}^n$ and $\mathcal{L}_{2^n}$ is defined as: given $\mathcal{X} = [\mathcal{X}^1 \ \cdots \ \mathcal{X}^n]^\top \in \mathcal{B}^n$, we have $\mathbf{x} := \begin{bmatrix}\mathcal{X}^1\\ \neg\mathcal{X}^1\end{bmatrix} \ltimes \cdots \ltimes \begin{bmatrix}\mathcal{X}^n\\ \neg\mathcal{X}^n\end{bmatrix}$, where "$\ltimes$" is called the semi-tensor product (STP) [7] of matrices.

### 3.1.1. Options

Options were first introduced by Sutton *et al.* [35] and are a generalization of macro-actions and temporally abstract actions developed for MDPs. An option is a way of behaving – a closed-loop policy for taking action. Options are initiated, decisions regarding which actions to take for some time steps are made, and then terminated. When an option terminates, the agent selects another option to be executed. In general, an option is triple $(\mathcal{I},\ \pi,\ \xi)$ with: i) initiation set $\mathcal{I} \subseteq \mathbf{X}$ that contains the states in which option may initiate; ii) policy $\pi : \mathbf{X} \times \mathbf{U} \to [0,\ 1]$ that determines the way in which the option selects actions; iii) stochastic termination condition $\xi$. An option is available at state $x_t$ if and only if $x_t \in \mathcal{I}$. Once the option is initiated, the agent selects the actions according to policy $\pi$ until the termination of the option, which follows the stochastic condition $\xi$. However, in this paper, we consider that option can initiate at any initial condition and the termination condition is deterministic. Moreover, the actions taken by the agent are Boolean values; thus, the policy results in a control action, either one or zero.

Considering the PBCN model and the MDP formulation introduced earlier, we formally define the option as follows.

**Definition 3.2 (Options).** *Given a state $\mathcal{X}_t \in \mathcal{B}^n$, an option $\mathscr{U}_t$ is a couple*

$$\mathscr{U}_t := \big(\mu_{\text{cn}}(\mathcal{X}_t),\ \mu_{\text{sp}}(\mathcal{X}_t)\big), \tag{16}$$

*where $\mu_{\text{cn}} : \mathcal{B}^n \to \mathcal{B}^m$ is the control policy that determines the optimal actions to be executed, and $\mu_{\text{sp}} : \mathcal{B}^n \to \mathbb{Z}$ is a termination condition that specifies the number of discrete time-steps to complete the current option.*

It is worth to highlight that, in our study $\mu_{\text{cn}}$ and $\mu_{\text{sp}}$ are directly mapped into the sampled-data state feedback controller and sampling period, respectively.

Once a particular option is selected, action $\mu_{\text{cn}}$ is kept constant by the agent with no other decision-making activities until the termination of the option. Then, a new action can be devised with the selection of the next option. Note that primitive actions $\mathcal{U} \in \mathcal{B}^m$ are special case of options. Each action $\mathcal{U}$ corresponds to an option that selects $\mathcal{U}$ everywhere $\mathcal{U}$ is available, and that always lasts exactly one step. In other words, an option of duration 1 is called basic or primitive action. Because the primitive actions are options, the agent's choice at each decision point is entirely among options, some of which persist for a single step, while others are temporally extended. Further, for the sampling period $\tau(\cdot)$, we set a user-defined upper bound $\mathcal{T}$, i.e., $\tau(\cdot) \in [1,\ \mathcal{T}]$. With this setting, we introduce the augmented action-space $\mathcal{B}^{m^+}$ that comprises all options, i.e., $\mathcal{B}^{m^+} := \bigcup_{\tau=1}^{\mathcal{T}} \bigcup_{i=1}^{2^m} (\mathcal{U}_i, \tau)$, with $|\mathcal{B}^{m^+}| = 2^m\mathcal{T}$.

### 3.1.2. Hierarchical Options

The options framework discussed in the previous subsection provides one level of abstraction on top of the primitive actions. We considered that an option makes choices among the primitive actions and termination conditions. In this subsection, we introduce a natural extension to options, namely hierarchical options, which make their choices among other options. Before defining the hierarchical options formally, we first introduce some notation as follows.

- A partial history $h_{t,T}$ is the sequence of all states, actions and rewards from time $t$ up to time $T \geq t$, defined as $h_{t,T} := (\mathcal{X}_t, \mathcal{U}_t, g_{t+1}, \mathcal{X}_{t+1}, \dots, \mathcal{X}_T)$.
- We denote by $\mathcal{H}$ the set of all possible partial histories, and $\mathcal{H}_t$ the set of all possible partial histories from time $t$ onward.
- We denote by $\eta$ the explicit representation of option (16). $\eta$ is a mapping from partial histories[3] and actions to probabilities of taking each action after each partial history: $\eta : \mathcal{H} \times \mathcal{B}^m \to [0, 1]$, where $\eta\big((\mathcal{X}_{t_0}, \dots, \mathcal{X}_t), \mathcal{U}\big) = \mathsf{P}\big\{\mathcal{U}_t = \mathcal{U} | \eta \text{ is initiated at } t_0, (\mathcal{X}_{t_0}, \dots, \mathcal{X}_t)\big\}$.

Because an option specifies a probability distribution over actions after each partial history, we have[4]

$$\sum_{\mathcal{U} \in \mathcal{B}^m} \eta(h_{t,T}, \mathcal{U}) = 1,\ \forall h_{t,T} \in \mathcal{H}.$$

[3] Here, $\mu_{\text{sp}}(\mathcal{X}_t)$, i.e., the termination condition in (16) acts as a reset action, thereby generating partial histories.

[4] In episodic tasks, the termination of an episode also terminates any option that could be executed at that time: $\eta(h_{t,T}, \mathcal{U}) = 1$ for all partial histories $h_{t,T}$ for which $s_T$ is a terminal state of the MDP.

Let $\mathcal{O}$ be a set of options and $\mathcal{O} \subseteq \mathcal{B}^{m^+}$. A history over options $\chi_{t,T}$ is a sequence of states, options and rewards: $\chi_{t,T} = (\mathcal{X}_t, \mathscr{U}_t, g_{t+1}, \mathcal{X}_{t+1}, \dots, \mathcal{X}_T, \mathscr{U}_T)$, where $\mathscr{U}_T$ denotes the option chosen at $\mathcal{X}_T$ and which is currently executing. It is worth noting that in a stochastic environment like PBCNs, a history over options can lead to many corresponding partial histories. Given a set of options $\mathcal{O}$, we denote by $\mathcal{H}_{\mathcal{O}}$ the set of all possible histories over options.

**Definition 3.3 (Hierarchical Options).** *Given an option $\mathscr{U}_t$, a hierarchical option $\hat{\mathscr{U}}_t$ is a couple $\hat{\mathscr{U}}_t : \big(\mu(\mathscr{U}_t), \beta(\mathscr{U}_t)\big)$, where $\mu : \mathcal{H}_{\mathcal{O}} \times \mathcal{O} \to [0,1]$ is an internal policy over options, and $\beta : \mathcal{H}_{\mathcal{O}} \to [0,1]$ is a termination condition.*

Compared to the Definition 3.2, the main difference is that the internal policy chooses among general options instead of only among primitive actions. We now show that for any hierarchical option, there is an explicit representation that generates the same distribution over histories. This property enables us to treat hierarchical options in the same way as (16), thereby providing an effective way of designing SDC for unknown PBCNs.

Executing hierarchical options involves two basic operations: sequencing of two or more options, and stochastic choice from the given set of of options. In the following, we show that for each of these basic operations, there is an explicit representation that generates the same history distribution.

**Lemma 3.1 (Sequencing).** *Given an unknown PBCN environment. For any two explicit representations of options, $\eta_1$ and $\eta_2$, there exists $\eta$ whose execution from any state $\mathcal{X}_t \in \mathcal{B}^n$ produces the same distribution of histories as the execution of $\eta_1$ at $\mathcal{X}_t$, followed by the execution of $\eta_2$.*

PROOF. Let $h_{t,T} \in \mathcal{H}$ be a history obtained by executing $\eta_1$ followed by $\eta_2$. If no additional information is available about the option that is executing, then in general, two situations are possible:

a. $\eta_1$ is still executing, in which case it is still picking primitive actions;
b. $\eta_1$ terminated at some intermediate time step $k$ and $\eta_2$ has taken over from there.

By resorting to these observations, $\eta$ can be specified as follows:

$$\eta(h_{t,T}, \mathcal{U}_j) = \eta_1(h_{t,T}, \mathcal{U}_j) + \sum_{k=t}^{T} \eta_1(h_{t,k}, \mu_{\mathrm{sp}})\eta_2(h_{h_{k,T}}, \mathcal{U}_j),\ \forall \mathcal{U}_j \in \mathcal{B}^m;$$

$$\eta(h_{t,T}, \mu_{\mathrm{sp}}) = \sum_{k=t}^{T} \eta_1(h_{t,k}, \mu_{\mathrm{sp}})\eta_2(h_{h_{k,T}}, \mu_{\mathrm{sp}}).$$

By this consideration, the distribution of actions after any given history is the same for $\eta$ as it is for the execution of $\eta_1$ followed by $\eta_2$. Then, it is straightforward to prove by induction that, if two options generate the same action choices after each history, then they will generate the same distribution over histories. This completes the proof. □

**Lemma 3.2 (Stochastic Choice).** *Let $\mathcal{O} = \{\eta_1, \dots, \eta_n\}$ be a finite set of options and let $\mu : \mathcal{B} \times \mathcal{O} \to [0,1]$ be a Markov policy that chooses from the options in $\mathcal{O}$. Then, for a given unknown PBCN environment, there exists an explicit representation of option $\eta$ such that, the execution of $\eta$ starting from any state $\mathcal{X}_t \in \mathcal{B}^n$ generates the same distribution over histories as a single choice of option performed at $\mathcal{X}_t$ according to distribution $\mu\big(\mathcal{X}_t, \cdot\big)$.*

PROOF. Let $h \in \mathcal{H}$ be a history over primitive actions since $\mu$ was initiated. Then $\eta$ should take into account the likelihood of each of the possible options being active, given the observed history, and average the suggested choices of action from each option. Formally, $\eta$ can be defined as follows:

$$\eta = \sum_{\eta_i \in \mathcal{O}} \mathsf{P}\{\eta_i | h\}\eta_i(h, \mu_{\mathrm{sp}}),\ i = 1, \dots, n;$$

$$\eta(h, \mathcal{U}_j) = \sum_{\eta_i \in \mathcal{O}} \mathsf{P}\{\mathcal{U}_j | \eta_i, h\} = \sum_{\eta_i \in \mathcal{O}} \mathsf{P}\{\eta_i | h\}\eta_i(h, \mathcal{U}_j).$$

Because $h$ has been observed, using Bayes' rule, we have

$$\mathsf{P}\{\eta_i | h\} = \mu(\mathcal{X}_t, \eta_i)\mathsf{P}\{h | \eta_i\},$$

where $\mathcal{X}_t$ is the first state of $h$. The second factor can be computed immediately from $\eta_i$ and from the dynamics of the PBCN.

By this construction, $\eta$ makes the same actions choices for any history as applying $\mu$ for one step. By induction on the length of the history, $\eta$ and $\mu$ will generate the same distribution over histories. □

Based on the above results, we now show that any hierarchical option has an explicit representation. This property enables us to treat hierarchical options in the same as options in (16), without designing special-purpose methods.

**Theorem 3.3.** *For an unknown PBCN environment modeled as an MDP, let* $\hat{\mathscr{U}}_t = \big(\mu(\mathscr{U}_t), \beta(\mathscr{U}_t)\big)$ *be a hierarchical option that chooses an explicit representation of options* $\mathscr{U}_t$ *from the set* $\mathcal{O} = \{\eta_1, \dots, \eta_n\}$*. Then, there exists an explicit representation* $\eta$ *such that the execution of* $\eta$ *from any state* $\mathcal{X}_t \in \mathcal{B}^n$ *produces the same distribution of histories as the execution of* $\hat{\mathscr{U}}_t$ *starting at* $\mathcal{X}_t$*.*

PROOF. The proof is based on the results and proofs of Lemmas 3.1 and 3.2. Consider a history $h_{t,T}$. We have to determine all the possible histories over options that could have generated this real history in the environment. Such histories can have from one option to at most $T - t$ options in them, and we have to consider all possible breakdown points. So the choice of $\eta$ for each history can be written as a sum of probabilities, taking into account all these possible breakdown points:

$$\eta(h_{t,T}, \mu_{\text{sp}}) = \sum_{k=0}^{T-t} \mathsf{P}_k(h_{t,T}, \mu_{\text{sp}}); \quad \eta(h_{t,T}, \mathcal{U}_j) = \sum_{k=0}^{T-t} \mathsf{P}_k\big(h_{t,T}, \mathcal{U}_j)\big),$$

where $\mathsf{P}_k(h_{t,T}, \mu_{\text{sp}})$ and $\mathsf{P}_k(h_{t,T}, \mathcal{U}_j))$ denote the probabilities of taking termination action $\mu_{\text{sp}}$ and action $\mathcal{U}_j$, respectively after $h_{t,T}$ assuming that $k$ reset points have occurred between $t$ and $T$.

Now consider that the first option initiated by $\eta$ is still executing. This is the stochastic choice case presented in Lemma 3.2, so $\mathsf{P}_0$ can be expressed as follows:

$$\mathsf{P}_0(h_{t,T}, \mathcal{U}_j) = \sum_{\eta_i \in \mathcal{O}} \mathsf{P}\{\eta_i | h_{t,T}, \mu\} \eta_i(h_{t,T}, \mathcal{U}_j), \ \forall \mathcal{U}_j \in \mathcal{B}^m;$$
$$\mathsf{P}_0(h_{t,T}, \mu_{\text{sp}}) = \sum_{\eta_i \in \mathcal{O}} \mathsf{P}\{\eta_i | h_{t,T}, \mu\} \eta_i(h_{t,T}, \mu_{\text{sp}}) \beta\big((\mathcal{X}_t, \eta_i, g_{t,T}, \mathcal{X}_T)\big),$$

where $g_{t,T}$ is the total discounted reward observed during the period from $t$ to $T$, i.e., $g_{t,T} = g_t + \ldots + \gamma^{T-t} g_T$. Based on the proof of Lemma 3.2 we have

$$\mathsf{P}\{\eta_i | h_{t,T}, \mu\} = \mu(\mathcal{X}_t, \eta_i) \mathsf{P}\{h_{t,T} | \eta_i\}.$$

Let us now consider the case in which one termination occurred between $t$ and $T$. In this case, the probabilities $\mathsf{P}_1$ can be determined analogously to Lemma 3.1 as follows:

$$\mathsf{P}_1(h_{t,T}, \mathcal{U}_j) = \sum_{k=t+1}^{T} \sum_{\eta_i, \eta_j \in \mathcal{O}} \mathsf{P}\{(\mathcal{X}_t, \eta_i, g_{t,k}, \mathcal{X}_k, \eta_j) | h_{t,T}, \mu\} \eta_j(h_{k,T}, \mathcal{U}_j);$$
$$\mathsf{P}_1(h_{t,T}, \mu_{\text{sp}}) = \sum_{k=t+1}^{T} \sum_{\eta_i, \eta_j \in \mathcal{O}} \mathsf{P}\{(\mathcal{X}_t, \eta_i, g_{t,k}, \mathcal{X}_k, \eta_j) | h_{t,T}, \mu\} \eta_j(h_{k,T}, \mathcal{U}_j) \beta\big((\mathcal{X}_t, \eta_i, g_{t,k}, \mathcal{X}_k, \eta_j, g_{k,T}, \mathcal{X}_T)\big).$$

The conditional probability of the history over options can be decomposed as follows:

$$\begin{aligned}\mathsf{P}\{\mathcal{X}_t, \eta_i, g_{t,k}, \mathcal{X}_k, \eta_j) | h_{t,T}, \mu\} &= \mathsf{P}\{\eta_i | h_{t,k}, \mu\} \eta_i(h_{t,k}, \mu_{\text{sp}}) \mathsf{P}\{\eta_j | (\mathcal{X}_t, \eta_i, g_{t,k}, \mathcal{X}_k), h_{k,T}, \mu\} \\ &= \mathsf{P}\{\eta_i | h_{t,k}, \mu\} \eta_i(h_{t,k}, \mu_{\text{sp}}) \mu\big((\mathcal{X}_t, \eta_i, g_{t,k}, \mathcal{X}_k), \eta_j\big) \mathsf{P}\{h_{k,T} | \eta_j\}.\end{aligned}$$

Likewise, any term $\mathsf{P}_k$ can be computed by using the unrolling technique as above.

By this construction, and by resorting to Lemmas 3.1 and 3.2, $\eta$ produces the same distribution of histories as hierarchical option $\hat{\mathscr{U}}_t$. □

Henceforth, we use the general term "options" regardless whether the representation is standard (i.e., (16)) or hierarchical or explicit. The above results describe how options can be specified and executed. But knowing how an option is executed is not enough for an agent that is trying to make decisions about the selection of options. In order

to make decisions, the agent needs long-term predictions about the consequences of behaving according to different policies over options. In the following, we generalize the conventional value functions to apply to options and policies over options.

Let $\Xi(\pi, \mathcal{X}, t)$ denote the event of policy $\pi$ being initiated in $\mathcal{X}$ at time $t$. Similarly, let $\Xi(\eta, h, t)$ be the event of an option $\eta$ continuing from history $h$ at time $t$, where $h$ is a history ending in $\mathcal{X}_t$.

**Definition 3.4.** *The value of a state $\mathcal{X} \in \mathcal{B}^n$ under a policy $\pi$ is the expected return if the policy is initiated in $\mathcal{X}$:*

$$v_\pi(\mathcal{X}) := \mathbb{E}\{g_{t+1} + \gamma g_{t+2} + \gamma^2 g_{t+3} + \cdots | \Xi(\pi, \mathcal{X}, t)\}.$$

Further, we generalize the action-value functions to *option-value functions*. Given an explicit representation of option $\eta$ and a policy over options $\mu$, let $\eta\mu$ denote the policy that first follows $\eta$ until it terminates and then starts choosing according to $\mu$ in the resulting state.

**Definition 3.5.** *The value of taking options $\eta$ in state $\mathcal{X} \in \mathcal{B}^n$ under policy $\mu$ is*

$$q_\mu(\mathcal{X}, \eta) := \mathbb{E}\{g_{t+1} + \gamma g_{t+2} + \gamma^2 g_{t+3} + \cdots | \Xi(\eta\mu, \mathcal{X}, t)\}.$$

With this setting, we now discuss results on how to compute optimal policies for different sets of options. To this aim, we present the *multi-time models* that allow us to plan ways of behaving using options.

### *3.1.3. Multi-time Models*

In order to reach our aim of SDC co-design with options we require a model of their consequences. For each state in which an option may be initiated, such models predict the state in which the option will terminate and the total reward received along the way. These quantities are discounted in a particular way. For any option $\mathscr{U}_t$, let $\Xi(\mathscr{U}_t, \mathcal{X}_t)$ denote the event of $\mathscr{U}_t$ being initiated in state $\mathcal{X}$ at time $t$. Then the reward part of the model of $\mathscr{U}_t$ for any state $\mathcal{X}_t \in \mathcal{B}^n$ is

$$\mathbf{G}_{\mathcal{X}_t}^{\mathscr{U}_t} = \mathbb{E}\{g_{t+1} + \gamma g_{t+2} + \ldots + \gamma^{k-1} g_{t+k} | \Xi(\mathscr{U}_t, \mathcal{X}_t)\}, \tag{17}$$

where $t + k$ is the sampling instant at which the controller (15) is updated or $\mathscr{U}_t$ terminates. The state prediction part of the model of $\mathscr{U}_t$ for state $\mathcal{X}_t$ is

$$\mathcal{P}_{\mathcal{X}_t, \mathcal{X}_{t+1}}^{\mathscr{U}_t} = \sum_{k=1}^{\infty} \mathsf{P}(\mathcal{X}_{t+1}, k)\gamma^k, \ \forall \mathcal{X}_{t+1} \in \mathcal{B}^n, \tag{18}$$

where $\mathsf{P}(\mathcal{X}_{t+1}, k)$ is the probability that the option terminates in $\mathcal{X}_{t+1}$ after $k$ steps. Equations (17) and (18) together are called as *multi-time model* because it describes the outcome of an option not at a single time but at potentially different times, appropriately combined. For the detailed discussion on multi-time models we refer to [28, 29]. It is worth highlighting that, under the multi-time model the transition from $\mathcal{X}_t$ to $\mathcal{X}_{t+1}$ for an action $\mathscr{U}_t \in \mathcal{B}^{m^+}$ is not simply the corresponding transition probability, but the transition probability times the discount factor $\gamma$.

Based on the definition of multi-time model, we now derive the following Bellman equation.

$$\begin{aligned} q_\mu(\mathcal{X}_t, \mathscr{U}_t) &= \mathbb{E}\{g_{t+1} + \ldots + \gamma^{k-1} g_{t+k} + \gamma^k \sum_{\mathscr{U} \in \mathcal{B}^{m^+}} \mu(\mathcal{X}_{t+k}, \mathscr{U}) q_\mu(\mathcal{X}_{t+k}, \mathscr{U}) | \Xi(\mathscr{U}_t, \mathcal{X}_t)\} \\ q_\mu(\mathcal{X}_t, \mathscr{U}_t) &= \mathbf{G}_{\mathcal{X}_t}^{\mathscr{U}_t} + \sum_{\mathcal{X}_{t+1}} \mathcal{P}_{\mathcal{X}_t, \mathcal{X}_{t+1}}^{\mathscr{U}_t} \sum_{\mathscr{U} \in \mathcal{B}^{m^+}} \mu(\mathcal{X}_{t+1}, \mathscr{U}) q_\mu(\mathcal{X}_{t+1}, \mathscr{U}). \end{aligned} \tag{19}$$

Similarly, the optimal Bellman equation can be expressed as:

$$\begin{aligned} q_{\mu^*}(\mathcal{X}_t, \mathscr{U}_t) &= \mathbb{E}\{g_{t+1} + \ldots + \gamma^{k-1} g_{t+k} + \gamma^k \min_{\mathscr{U} \in \mathcal{B}^{m^+}} q_{\mu^*}(\mathcal{X}_{t+k}, \mathscr{U}) | \Xi(\mathscr{U}_t, \mathcal{X}_t)\}, \\ q_{\mu^*}(\mathcal{X}_t, \mathscr{U}_t) &= \mathbf{G}_{\mathcal{X}_t}^{\mathscr{U}_t} + \sum_{\mathcal{X}_{t+1}} \mathcal{P}_{\mathcal{X}_t, \mathcal{X}_{t+1}}^{\mathscr{U}_t} \min_{\mathscr{U} \in \mathcal{B}^{m^+}} q_{\mu^*}(\mathcal{X}_{t+k}, \mathscr{U}). \end{aligned} \tag{20}$$

Each of the Bellman equations for options, (19), (20), defines a system of equations whose unique solution is the corresponding value function. Based on the options framework, we now present an algorithm to design an SDC policy to control unknown PBCNs.

### 3.2. DDQNs with Options

As outlined in Section 2.4 and (7), a primary function of DDQNs is to output the approximated action-values for each possible action $\mathcal{U}_t \in \mathcal{B}^m$, with each action corresponding to a single output neuron of a DQN. Leveraging this functionality of DDQNs, we extend the options framework for DDQNs to reach our aim. Although options are action tuples of the form $\mathscr{U}_t = \big(\mu_{\text{cn}}(\mathcal{X}_t), \mu_{\text{sp}}(\mathcal{X}_t)\big)$, they are easily applicable to DDQNs, when sub-policies $\mu_{\text{cn}}$ and $\mu_{\text{sp}}$ are learned in tandem, such as in the co-design scheme presented in this paper.

In our implementation, we set the output layer of the DDQN to contain $|\mathcal{B}^{m^+}|$ neurons, with each neuron corresponding to a single option. For the sake of practicality, we encode between mapping an option $\mathscr{U}_i = (\mathcal{U}, \tau)$ to the $i$-th neuron, where $i = (\tau - 1) \cdot 2^m + \texttt{bin}(\mathcal{U})$, with $\texttt{bin} : \mathcal{B}^m \to \mathbb{Z}_+$ being a function that returns the unsigned integer as a Boolean vector in $\mathcal{B}^m$.

Thus, for a mini batch of transitions $\mathcal{M}_1 = (\mathcal{X}_t, \mathcal{B}^{m^+}, \mathbf{G}, \mathcal{X}_{t+1}, \bar{\mathbf{\Theta}})$ we can perform a weight update in a very similar fashion to (13), without decomposing the *option* into its primitive counterparts:

$$\begin{aligned}
&\mathcal{W} \leftarrow \mathcal{W} - \alpha \nabla_{\mathcal{W}}(\mathscr{L}(\mathcal{W})) \cdot \bar{\mathbf{\Theta}});\\
&\mathscr{L}(\mathcal{W}) = Q(\mathcal{X}_t, \mathscr{U}_t; \mathcal{W}) - \mathbf{G} - \gamma Q(\mathcal{X}_{t+1}, \underset{\mathscr{U} \in \mathcal{B}^{m^+}}{\arg\min}\, Q(\mathcal{X}_{t+1}, \mathscr{U}, \mathcal{W}); \mathcal{W}^-).
\end{aligned} \tag{21}$$

It is worth noting that the temporal abstraction does not affect the DDQN-learning algorithm, and hence the lack of a dramatic change to the TD-error $\mathscr{L}(\mathcal{W})$. When the DDQN+PER based RL agent selects an option, it can be certain with regards to the sampling period that will elapse. Therefore, there is no need to either aggressively discount costs received in the cost trajectory learned in the unknown PBCN environment as a result of the option, or further discount the expected cost within the TD-error. In the following, we discuss the cost function to be optimized using option-based DDQN algorithm to devise the SDC policy.

#### *3.2.1. Cost Function*

Given a state $\mathcal{X}_t \in \mathcal{B}^n$, applying the option $\mathscr{U}_t$ will result in a cost of $g_t = \sum_{j=t}^{t+k} g_j$, where $k$ is the sampling period, and

$$g_j = \begin{cases} f \cdot (-s_c - 1) + 1, \ \mathcal{X}_j = \mathcal{X}_e, \\ 1, \ \mathcal{X}_j \neq \mathcal{X}_e \wedge \mathcal{X}_j \neq \mathcal{X}_n, \ \forall \mathcal{X}_n \in \mathcal{B}^n, \\ 2, \ \exists \mathcal{X}_n \Rightarrow \mathcal{X}_j = \mathcal{X}_n, \end{cases} \tag{22}$$

where $\mathcal{X}_e$ and $\mathcal{X}_n$ denote the desired and undesired equilibrium points, respectively. Additionally, $f$ is a special variable with the following behavior:

$$f = \begin{cases} 1, \ \mathcal{X}_e \text{ not reached yet}, \\ 0, \ \text{otherwise}. \end{cases} \tag{23}$$

In practice, this means that for all time steps where the PBCN is not in a desired, or undesired equilibrium point, the cost is 1. For time steps where the PBCN is in an undesired equilibrium point, the cost is 2. For the *first time step*, where the desired equilibrium point is reached, the cost is "$-s_c$". For all others, the cost is the same as the second case of (23). $s_c$ is a hyperparameter that can affect the learned SDC policy. A typical value considered in the paper is $s_c = 5$ or $s_c = 10$.

With these settings of options-based DDQN algorithm and such selection of cost function, the SDC co-design problem in a model-free framework can be formulated as

$$\min_{\mu(\cdot)} \mathbb{E}_{\mu}\left[\sum_{t=0}^{\infty} \gamma^t g_{t+1}(\mathcal{X}_t, \mathcal{U}_t, \mathcal{X}_{t+1})\right], \ \forall \mathcal{X}_0 \in \mathcal{B}^n \tag{24}$$

subject to (unknown) system (1) and Assumption 1.

To solve the problem (24), we here present a modified DDQN algorithm (Algorithm 1) that utilizes PER and options with multi-time model to devise an SDC strategy for PBCNs in a model-free framework. The output of the algorithm produces two concurrent policies $\mu^*_{\text{cn}}(\mathcal{X}_t)$ and $\mu^*_{\text{sp}}(\mathcal{X}_t)$ corresponding to an optimal control action and sampling period to update the devised control action at each state, respectively.

**Algorithm 1** Co-design of SDC policy using DD$Q$N+PER and Options

**Input:** $\mathcal{X}_e$, $N$, $\gamma$, $\beta$, $\min_\epsilon$, `horizon`, $|\mathcal{M}_1|$, $\alpha$, `updateInterval`
**Output:** $\mu^*_{\text{cn}}(\mathcal{X}_t)$, $\mu^*_{\text{sp}}(\mathcal{X}_t)$, $\forall \mathcal{X}_t \in \mathcal{B}^n$

$\mathcal{W} \leftarrow \texttt{rand}(\{0,1\})$, $\mathcal{W}^- \leftarrow \mathcal{W}$ ▷ Initialize network weights
$\mathcal{M} \leftarrow \emptyset$, $\texttt{inc}_\beta \leftarrow \frac{\beta}{0.75 \cdot N}$, $\max_p \leftarrow 1$ ▷ Initialize prioritized experience replay
$\epsilon \leftarrow 1, \texttt{dec}_\epsilon \leftarrow \frac{1-\min_\epsilon}{N}$ ▷ Initialize $\epsilon$-greedy
$tc \leftarrow 0$
**for** `episode` $\in [0, N]$ **do**
  $t \leftarrow 0, \mathcal{X}_t \leftarrow \texttt{rand}(\mathcal{B}^n)$ ▷ Initialize the network in a random state
  **while** $t \neq$ `horizon` $\wedge\ \mathcal{X}_t \neq \mathcal{X}_e$ **do**
    $\mathscr{U}_t \leftarrow \epsilon$-greedy$(\epsilon, \mathcal{X}_t)$
    $t+1, \mathcal{X}_{t+1}, g_{t+1} \leftarrow \texttt{apply}(\mathscr{U}_t)$ ▷ Apply the action to the environment
    Store $(\mathcal{X}_t, \mathscr{U}_t, g_{t+1}, \mathcal{X}_{t+1})$ in the replay buffer $\mathcal{M}$ with $p_{t+1} = \max_p$
    **if** $|\mathcal{M}| \geq$ `batchSize` **then**
      Sample a batch of transitions $\mathcal{M}_1 = (\mathcal{X}_t, \mathcal{B}^{m^+}, \mathbf{G}, \mathcal{X}_{t+1}, \bar{\mathbf{\Theta}})$ of size $|\mathcal{M}_1|$,
      where $\forall i \in \mathcal{M}_1$ was sampled with probability $\mathsf{P}\{i\}$ according to (11)
      and $\bar{\theta}_i \in \bar{\mathbf{\Theta}}$ is the normalized IS weight for the transition, computed according to (12).
      $\mathscr{L}(\mathcal{W}) \leftarrow Q(\mathcal{X}_t, \mathscr{U}_t; \mathcal{W}) - \mathbf{G} - \gamma Q(\mathcal{X}_{t+1}, \arg\min_{\mathscr{U} \in \mathcal{B}^{m^+}} Q(\mathcal{X}_{t+1}, \mathscr{U}, \mathcal{W}); \mathcal{W}^-)$
      $\forall j \in \mathcal{M}_1$ update priorities with $\bar{p}_{j+1} \leftarrow |\mathscr{L}(\mathcal{W})_{j+1}| + \zeta$
      $\mathcal{W} \leftarrow \mathcal{W} - \alpha \nabla_{\mathcal{W}} (\mathscr{L}(\mathcal{W}) \cdot \bar{\mathbf{\Theta}})$ ▷ Stochastic gradient descent
      $tc \leftarrow tc + 1$
      **if** $tc \mod$ `updateInterval` $= 0$ **then** ▷ Update target DQN
        $\mathcal{W}^- \leftarrow \mathcal{W}$
      **end if**
    **end if**
  **end while**
  $\beta \leftarrow \min(\beta + \texttt{inc}_\beta, 1)$, $\epsilon \leftarrow \max(\epsilon - \texttt{dec}_\epsilon, \min_\epsilon)$ ▷ Update Hyperparameters
**end for**
$\left(\mu^*_{\text{cn}}(\mathcal{X}_t), \mu^*_{\text{sp}}(\mathcal{X}_t)\right) \leftarrow \arg\min_{\mathscr{U}} Q(\mathcal{X}_t, \mathscr{U}, \mathcal{W}); \mathcal{W}^-), \forall \mathcal{X}_t \in \mathcal{B}^n$. ▷ Devising control action and sampling period.

## 4. Simulation Results and Discussion

To evaluate the performance of the presented SDC co-design algorithm, we implemented three different GRNs of varying sizes modeled as PBCNs. Network dynamics of these PBCNs is given below. Further, we compare the results with state-of-the-art feedback controllers for PBCNs such as shortest path controller and self-triggered controller.

**Example 4.1 (Bacteriophage $\lambda$: 4-1).** *We consider a 5-gene (four nodes and one control input) PBCN model of the bacteriophage $\lambda$ as:*
$\mathcal{X}^1_+ = \neg\mathcal{X}^2 \wedge \neg\mathcal{X}^4$*;* $\mathcal{X}^2_+ = \neg\mathcal{X}^4 \wedge \neg\mathcal{U} \wedge (\mathcal{X}^2 \vee \mathcal{X}^3)$*;* $\mathcal{X}^3_+ = \neg\mathcal{X}^2 \wedge \neg\mathcal{X}^4 \wedge \mathcal{X}^1$*,* $\mathsf{P} = 0.7$ *and* $\mathcal{X}^3_+ = 0$*,* $\mathsf{P} = 0.3$*;* $\mathcal{X}^4_+ = \neg\mathcal{X}^2 \wedge \neg\mathcal{X}^3$*, where $\mathcal{X}^i_+$ represents the value of $i$-th node at the next time-step. $\mathcal{X}^1, \mathcal{X}^2, \mathcal{X}^3$, and $\mathcal{X}^4$, represent four phage genes $N$, $cI$, $cII$, $cro$, respectively. $\mathcal{U}$ is the control input which represents the environmental conditions. $\mathcal{X}_e = (0\ 0\ 0\ 1)$ is given, which represents the lytic pathway of bacteriophage. Our aim is to find a sampled-data controller with sampling periods to stabilize the system at a given $\mathcal{X}_e$. We refer to [46] for the detailed description of the genes in the network.*

**Example 4.2 (Ara Operon in the Escherichia Coli: 9-4).** *The network dynamics with a total of 13 genes is shown below. The network has nine genes which are represented by variables $\mathcal{X}^1$ to $\mathcal{X}^9$, respectively as follows: intracellular arabinose ($A$), intracellular arabinose ($A_m$), arabinose-bound AraC protein ($Ara_+$), cAMP–CAP protein complex ($C$), enzymes AraA, AraB, and AraD ($E$), DNA loop ($D$), mRNA of the structural genes ($M_s$), mRNA of the transport genes ($M_T$), and transport proteins ($T$). The parameters extracellular arabinose ($A_e$), extracellular arabinose ($A_{em}$), AraC protein ($Ara\_$), and extracellular glucose ($G_e$) are considered as control inputs and represented by $\mathcal{U}_1$ to $\mathcal{U}_4$,*

*respectively. We consider a given equilibrium point* $\mathcal{X}_e = (0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1)$ *as the ON state of the network and design an SDC scheme to stabilize the network at* $\mathcal{X}_e$.

$$\mathcal{X}^1 = f_1^{(1)} : (\mathcal{U}^1 \wedge \mathcal{X}^8),\ p_1^{(1)} = 1$$
$$\mathcal{X}^2 = f_2^{(1)} : (\mathcal{U}^2 \wedge \mathcal{X}^9) \vee \mathcal{U}^1,\ p_2^{(1)} = 1$$
$$\mathcal{X}^3 = \begin{cases} f_3^{(1)} : (\mathcal{U}^2 \vee \mathcal{X}^1) \wedge \mathcal{U}^3,\ p_3^{(1)} = 0.8 \\ f_3^{(2)} : \mathcal{X}^3,\ p_3^{(2)} = 0.2 \end{cases}$$
$$\mathcal{X}^4 = f_4^{(1)} : \neg\mathcal{U}^4,\ p_4^{(1)} = 1$$
$$\mathcal{X}^5 = \begin{cases} f_5^{(1)} : \mathcal{X}^7,\ p_5^{(1)} = 0.8 \\ f_5^{(2)} : \neg\mathcal{X}^7,\ p_5^{(2)} = 0.2 \end{cases}$$
$$\mathcal{X}^6 = \begin{cases} f_6^{(1)} : \neg\mathcal{X}^3 \wedge \mathcal{U}^3,\ p_6^{(1)} = 0.7 \\ f_6^{(2)} : \mathcal{X}^6,\ p_6^{(1)} = 0.3 \end{cases}$$
$$\mathcal{X}^7 = \begin{cases} f_7^{(1)} : \mathcal{X}^3 \wedge \mathcal{X}^4 \wedge \neg\mathcal{X}^6,\ p_7^{(1)} = 0.8 \\ f_7^{(2)} : \mathcal{X}^3 \wedge \mathcal{X}^4 \wedge \mathcal{X}^6,\ p_7^{(2)} = 0.2 \end{cases}$$
$$\mathcal{X}^8 = f_8^{(1)} : \mathcal{X}^3 \wedge \mathcal{X}^4,\ p_8^{(1)} = 1$$
$$\mathcal{X}^9 = f_9^{(1)} : \mathcal{X}^8,\ p_9^{(1)} = 1.$$

**Example 4.3 (28-gene Escherichia Coli: 28-3).** *Consider the following 28-gene PBCN model with three control inputs:*
$\mathcal{X}_+^1 = \mathcal{X}^6 \wedge \mathcal{X}^{13}$; $\mathcal{X}_+^2 = \mathcal{X}^{25}$; $\mathcal{X}_+^3 = \mathcal{X}^2$; $\mathcal{X}_+^4 = \mathcal{X}^{28}$; $\mathcal{X}_+^5 = \mathcal{X}^{21}$; $\mathcal{X}_+^6 = \mathcal{X}^5$; $\mathcal{X}_+^7 = (\mathcal{X}^{15} \wedge \mathcal{U}^2) \vee (\mathcal{X}^{26} \wedge \mathcal{U}^2)$; $\mathcal{X}_+^8 = \mathcal{X}^{14}$; $\mathcal{X}_+^9 = \mathcal{X}^{18}$; $\mathcal{X}_+^{10} = \mathcal{X}^{25} \wedge \mathcal{X}^{28}$; $\mathcal{X}_+^{11} = \neg\mathcal{X}^9$; $\mathcal{X}_+^{12} = \mathcal{X}^{24}$; $\mathcal{X}_+^{13} = \mathcal{X}^{12}$; $\mathcal{X}_+^{14} = \mathcal{X}^{28}$; $\mathcal{X}_+^{15} = (\neg\mathcal{X}^{20}) \wedge \mathcal{U}^1 \wedge \mathcal{U}^2$; $\mathcal{X}_+^{16} = \mathcal{X}^3$; $\mathcal{X}_+^{17} = \neg\mathcal{X}^{11}$; $\mathcal{X}_+^{18} = \mathcal{X}^2$; $\mathcal{X}_+^{19} = (\mathcal{X}^{10} \wedge \mathcal{X}^{11} \wedge \mathcal{X}^{25} \wedge \mathcal{X}^{28}) \vee (\mathcal{X}^{11} \wedge \mathcal{X}^{23} \wedge \mathcal{X}^{25} \wedge \mathcal{X}^{28})$; $\mathcal{X}_+^{20} = \mathcal{X}^7 \vee \neg\mathcal{X}^{26}$; $\mathcal{X}_+^{21} = \mathcal{X}^{11} \vee \mathcal{X}^{22}$; $\mathcal{X}_+^{22} = \mathcal{X}^2 \wedge \mathcal{X}^{18}$; $\mathcal{X}_+^{23} = \mathcal{X}^{15}$; $\mathcal{X}_+^{24} = \mathcal{X}^{18}$; $\mathcal{X}_+^{25} = \mathcal{X}^8$; $\mathcal{X}_+^{26} = \neg\mathcal{X}^4 \wedge \mathcal{U}^3$, $\mathsf{P} = 0.5$, *and* $\mathcal{X}_+^{26} = \mathcal{X}^{26}$, $\mathsf{P} = 0.5$; $\mathcal{X}_+^{27} = \mathcal{X}^7 \vee (\mathcal{X}^{15} \wedge \mathcal{X}^{26})$; $\mathcal{X}_+^{28} = \neg\mathcal{X}^4 \wedge \mathcal{X}^{15} \wedge \mathcal{X}^{24}$.

*The model is a reduced-order model of the 32-gene T-cell receptor kinetics model given in [13]. We aim to design an SDC strategy such that the network is stabilized at* $(0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,0\,0\,0\,1\,1\,0)$.

## 4.1. Training & DD$Q$N Architecture

For the purposes of evaluation, we trained DD$Q$N+PER agents over 250,000 episodes on the sampled-data control problem applied on the aforementioned networks using $\mathcal{T} = 10$. The discount factor $\gamma$ for $Q$-Learning was set to 0.99. $\min_\epsilon$ for the $\epsilon$-greedy policy followed during training was set to 0.01. For PER, $\beta$ was set to 0.4 at the start of training and increased up to a maximum of $\max_\beta = 1$ through the course of training.

The decision `horizon` was set to 11. The decision horizon denotes the maximum number of environment interactions the Agent is given to control the network with before the episode is reset. That is, the Agent can select up to `horizon` $= 11$ options sequentially to attempt to steer the PBCN's state trajectory into the desired equilibrium point $\mathcal{X}_e$.

In terms of ANN Architecture, the DD$Q$Ns were initialized with three hidden layers of 50 neurons each. While a larger scale ANN could have been used, the aforementioned architecture did not seem to showcase issues such as over-fitting or inability to learn the control problem, and as such it did not seem necessary to do so. Each layer also includes an additive bias, and passes through a rectified linear activation function (ReLU). Regarding the outer layers of the ANN, the input layer was set to $N$ neurons, while the output layer contains $2^m\mathcal{T}$ neurons – one corresponding to each possible SDC action $\mathscr{U} \in \mathcal{B}^{m^+}$.

Simulations were performed using a Python implementation of the sampled-data PBCN environment, and a PyTorch implementation of the DD$Q$N+PER Agent. The training was carried out on a standard personal computer with an NVIDIA RTX 3070 GPU and an AMD Ryzen 5 5600X CPU.

## 4.2. Evaluation Scenario

An episode in our evaluation scenario is defined as follows:

(1) Initialize the PBCN with a state $\mathcal{X}_0$.
(2) Select a control *option* $\mathcal{U}_t$ using the DD$Q$N.
(3) Read in the new network state $\mathcal{X}_{t+1}$.
(4) Repeat until $\mathcal{X}_{t+1} = \mathcal{X}_e$ or the `horizon` is reached.

We initialize and run an episode $\forall\mathcal{X}_0 \in \mathcal{B}^n \setminus \{\bar{\mathcal{X}}_e\}$ and record metrics of interest, such as the following:

(a) Winrate: the percentage of initial states $\mathcal{X}_0$ that can be steered to $\mathcal{X}_e$ with an application of a possible SDC policy.

(b) Number of interactions: the average number of options the agent had to select in its attempt to control the unknown PBCN, per episode.

(c) Number of time steps $t$: the average number of time steps the PBCN iterated for.

For networks where the size of the state-space did not permit us to run an episode for every possible initial state, such as Example 4.3, we instead randomly select 200,000 initial states to evaluate from.

## 4.3. Comparison with Shortest Path Controller and Self-Triggering Control

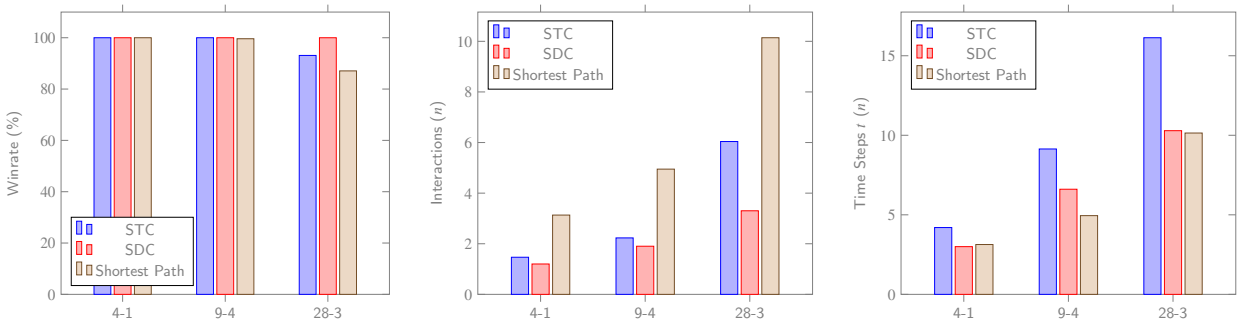


**Figure 1:** Comparison of the proposed SDC strategy with STC and shortest path controller to stabilize the PBCN at a given equilibrium point. [left] winrate (higher is better); [center] number of interactions (lower is better); [right] number of time steps (lower is better).

The baseline for comparison is a shortest path controller for the three PBCNs, derived using the DD$Q$N+PER method introduced by [27], but using the training settings outlined in the previous section. The results on the three aforementioned key metrics, as seen in Figure 1 confirms that the SDC algorithm does indeed drastically reduce the number of controller-environment interactions across the board. Most notably, in the third example, the PBCN with 28 nodes, where the number of interactions has been cut by two thirds.

At the same time, the sampled-data controller is almost identical, again across all examples, to the optimal shortest-path controller, as the time step comparison figure reveals. As a result, we have managed to, as proposed, cut down the number of interactions with the PBCN without compromising in terms of optimality. In terms of success rate both seem to achieve similar results, with the exception of the third example where the shortest path controller seems to struggle for a small number of states, indicating that the temporal abstraction of SDC allows for more controllability in certain cases.

In the very same figures, we also compare SDC with self-triggering control, another approach that aims to cut down the amount of agent-environment interaction and provide a temporal abstraction over the control actions. We adapted the method outlined by [2] to our DD$Q$N+PER framework in much the same way we implemented the SDC option scheme within deep reinforcement learning. The training settings used were identical to the ones used for the sampled-data controller training, with the main self-triggering control hyperparameter being $\mathcal{T}$ which we set to 5, much like in the original paper. The two approaches seem to perform similarly, with the primary difference being that SDC is much closer to the shortest path controller than STC is. The SDC also pulls ahead of STC as far as the number of interactions goes in all examples.

Figure 2 shows how the designed sampled-data controller (red line) utilizes the temporal abstraction technique to reach the equilibrium point with only one change in the control action. Whereas the traditional STP-based controller (blue line) changes control action twice in order to steer the initial state to $\mathcal{X}_e$.

Further, in Figure 3 we show the effectiveness of the designed SDC policy for Example 4.2. State trajectories for all genes reach the desired equilibrium point $\mathcal{X}_e = (0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1)$ in less than 10 steps.

## 4.4. The $\mathcal{T}$ hyperparameter

A factor that greatly seems to affect the derived SDC policy is the hyperparameter $\mathcal{T}$, which defines the upper bound for the sampling period delay the agent can choose to impose. We compare between three $\mathcal{T}$ values: 5, 10 and 15, for Example 4.3, as depicted in Figure 4.

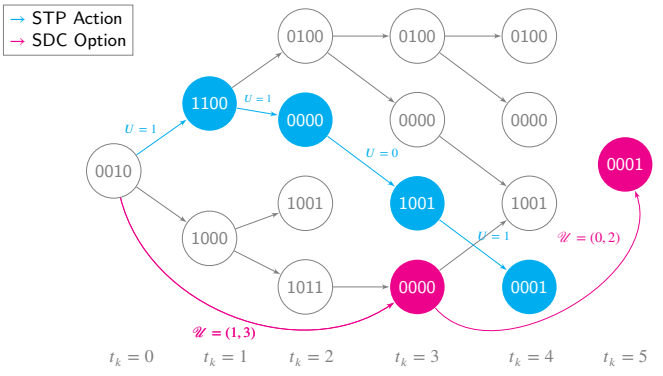


**Figure 2:** Demonstration of sampled-data control versus a shortest path controller (i.e., STP Action) for Example 4.1. Nodes represent network states, and edges represent the ability for the network to evolve from one state to another in the next time-step $t_k$. Each highlighted arrow represents an agent-environment interaction and is annotated with the control action that took place.

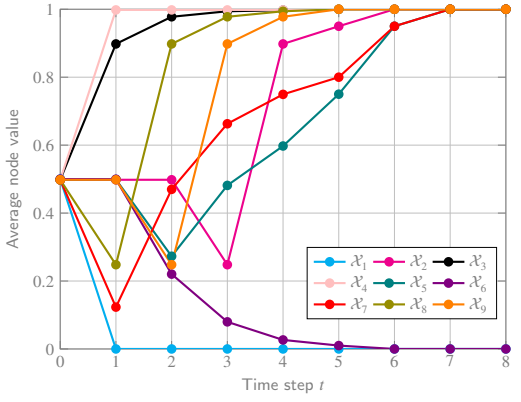


**Figure 3:** Demonstration of the evolution of genes (or nodes) under the SDC scheme for Example 4.2, averaged across multiple simulations. The system is stabilized at $\mathcal{X}_e = (0\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1)$ under the designed SDC.

As one would expect, lowering the $\mathcal{T}$ value leads to a larger number of interactions with the environment, as the agent is not allowed to pick larger intervals that would help minimize that metric. However, raising it seems to provide diminishing returns, as for $\mathcal{T} = 15$ we do not observe a drastic decrease in average number of environment interactions during evaluation, and instead we do observe a larger number of time steps taken until convergence to the desired equilibrium point, which is undesired behavior. Nonetheless, we do not observe any difference in success rate across all three cases, however. Overall, it becomes clear that setting the $\mathcal{T}$ value properly is crucial to deriving the best sampled-data controllers possible.

## 5. Conclusions

In this paper, a model-free and scalable control technique has been presented to control GRNs modeled as PBCNs. In particular, a model-free SDC algorithm has been developed by utilizing DD$Q$N, PER and the concept of temporal abstractions called *options*. The devised controllers are energy efficient since the sampling period to update the controller is obtained smartly by utilizing the temporal abstraction framework in model-free setting. The performance of the proposed method has been compared with the state-of-the-art techniques such as state feedback control and self-triggering control. A brief discussion on the reward setting has been included in Section 4 to highlight how a change in the reward could affect the overall control policy.

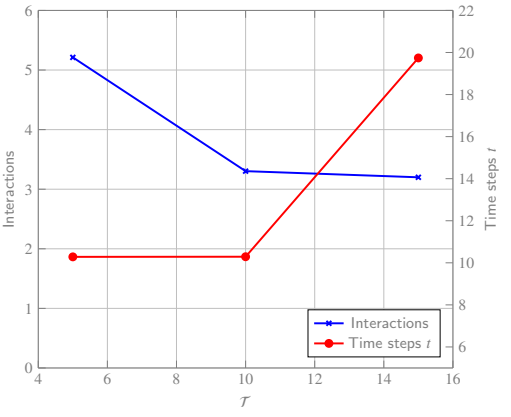


**Figure 4:** Comparison of the number of time steps and number of interactions for different $\mathcal{T}$ values (Example 4.3). Larger $\mathcal{T}$ allows the agent to take more steps in the environment while agent-environment interactions do not change drastically.

# References


[1] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio. Double deep-q learning-based output tracking of probabilistic Boolean control networks. *IEEE Access*, 8:199254–199265, 2020.

[2] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio. Model-free self-triggered control co-design for probabilistic Boolean control networks. *IEEE Control Systems Letters*, 5(5):1639–1644, 2020.

[3] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio. Reinforcement learning approach to feedback stabilization problem of probabilistic Boolean control networks. *IEEE Control Systems Letters*, 5(1):337 – 342, 2020.

[4] P. Bajaria, A. Yerudkar, and C. Del Vecchio. Aperiodic sampled-data stabilization of probabilistic Boolean control networks: Deep q-learning approach with relaxed Bellman operator. In *2021 European Control Conference (ECC)*, pages 836–841. IEEE, 2021.

[5] P. Bajaria, A. Yerudkar, and C. Del Vecchio. Random forest q-learning for feedback stabilization of probabilistic Boolean control networks. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1539–1544. IEEE, 2021.

[6] B. Chen, J. Cao, Y. Luo, and L. Rutkowski. Asymptotic output tracking of probabilistic Boolean control networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(8):2780–2790, 2020.

[7] D. Cheng, H. Qi, and Z. Li. *Analysis and control of Boolean networks: a semi-tensor product approach*. London, U.K.: Springer, 2011.

[8] D. P. De Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6): 850–865, 2003.

[9] E. Fornasini and M. E. Valcher. Optimal control of Boolean control networks. *IEEE Transactions on Automatic Control*, 59(5):1258–1270, 2014.

[10] X.-G. Han, W.-D. Yang, X.-Y. Chen, Z.-W. Li, and Z.-Q. Chen. Detectability vverification of probabilistic Boolean networks. *Information Sciences*, 548:313–327, 2021.

[11] C. Huang, J. Lu, D. W. Ho, G. Zhai, and J. Cao. Stabilization of probabilistic Boolean networks via pinning control strategy. *Information Sciences*, 510:205–217, 2020.

[12] M. R. Karlsen and S. Moschoyiannis. Evolution of control with learning classifier systems. *Applied Network Science*, 3(1):30, 2018.

[13] S. Kharade, S. Sutavani, S. Wagh, A. Yerudkar, C. Del Vecchio, and N. Singh. Optimal control of probabilistic Boolean control networks: A scalable infinite horizon approach. *International Journal of Robust and Nonlinear Control*, 2021, DOI:10.1002/rnc.5909.

[14] H. Li, Y. Wang, and P. Guo. State feedback based output tracking control of probabilistic Boolean networks. *Information Sciences*, 349:1–11, 2016.

[15] L. Li, A. Zhang, and J. Lu. Robust set stability of probabilistic Boolean networks under general stochastic function perturbation. *Information Sciences*, 582:833–849, 2022.

[16] Y. Li, J. Zhu, B. Li, Y. Liu, and J. Lu. A necessary and sufficient graphic condition for the original disturbance decoupling of Boolean networks. *IEEE Transactions on Automatic Control*, 66(8):3765–3772, 2020.

[17] Y. Li, J.-e. Feng, and B. Wang. Output feedback observability of switched Boolean control networks. *Information Sciences*, 612:612–625, 2022.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971v6*, 2016.

[19] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293 – 321, 1992.

[20] J. Liu, Y. Liu, Y. Guo, and W. Gui. Sampled-data state-feedback stabilization of probabilistic Boolean control networks: A control Lyapunov function approach. *IEEE Transactions on Cybernetics*, 50(9):3928–3937, 2019.

[21] Y. Liu, H. Chen, J. Lu, and B. Wu. Controllability of probabilistic Boolean control networks based on transition probability matrices. *Automatica*, 52:340–345, 2015.

[22] Y. Liu, L. Wang, J. Lu, and J. Cao. Sampled-data stabilization of probabilistic Boolean control networks. *Systems & Control Letters*, 124: 106–111, 2019.

[23] Z. Liu, J. Zhong, Y. Liu, and W. Gui. Weak stabilization of Boolean networks under state-flipped control. *IEEE Transactions on Neural Networks and Learning Systems*, 2021, DOI: 10.1109/TNNLS.2021.3106918.

[24] A. R. Mahmood, H. P. van Hasselt, and R. S. Sutton. Weighted importance sampling for off-policy learning with linear function approximation. *Advances in Neural Information Processing Systems*, 27:3014–3022, 2014.

[25] R. Pal, A. Datta, M. L. Bittner, and E. R. Dougherty. Intervention in context-sensitive probabilistic Boolean networks. *Bioinformatics*, 21(7): 1211–1218, 2005.

[26] G. Papagiannis and S. Moschoyiannis. Learning to control random boolean networks: A deep reinforcement learning approach. In *International Conference on Complex Networks and Their Applications*, pages 721–734. Springer, 2019.

[27] G. Papagiannis and S. Moschoyiannis. Deep reinforcement learning for control of probabilistic Boolean networks. In *International Conference on Complex Networks and Their Applications*, pages 361–371. Springer, 2020.

[28] D. Precup and R. S. Sutton. Multi-time models for reinforcement learning. In *Proceedings of the ICML'97 Workshop on Modelling in Reinforcement Learning*, 1997.

[29] D. Precup and R. S. Sutton. Multi-time models for temporally abstract planning. *Advances in Neural Information Processing Systems*, 10, 1997.

[30] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[31] K. Sarda, A. Yerudkar, and C. Del Vecchio. Disturbance decoupling control design for Boolean control networks: a Boolean algebra approach. *IET Control Theory & Applications*, 14(16):2339–2347, 2020.

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[33] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, 2002.

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

[35] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[36] H. Tian and Y. Hou. State feedback design for set stabilization of probabilistic Boolean control networks. *Journal of the Franklin Institute*, 356(8):4358–4377, 2019.

[37] H. van Hasselt. Double q-learning. *Advances in Neural Information Processing Systems*, 23:2613–2621, 2010.

[38] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[39] B. Wang and J.-e. Feng. On detectability of probabilistic Boolean networks. *Information Sciences*, 483:383–395, 2019.

[40] J. Wang, W. Liu, S. Fu, and J. Xia. On robust set stability and set stabilization of probabilistic Boolean control networks. *Applied Mathematics and Computation*, 422:126992, 2022.

[41] L. Wang, Y. Liu, Z.-G. Wu, J. Lu, and L. Yu. Stabilization and finite-time stabilization of probabilistic Boolean control networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(3):1559–1566, 2019.

[42] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[43] Y. Wu, Y. Guo, and M. Toyoda. Policy iteration approach to the infinite horizon average optimal control of probabilistic Boolean networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(7):2910–2924, 2020.

[44] M. Xu, Y. Liu, J. Lou, Z.-G. Wu, and J. Zhong. Set stabilization of probabilistic Boolean control networks: A sampled-data control approach. *IEEE Transactions on Cybernetics*, 50(8):3816–3823, 2019.

[45] X. Yang and H. Li. On state feedback asymptotical stabilization of probabilistic Boolean control networks. *Systems & Control Letters*, 160: 105107, 2022.

[46] A. Yerudkar, C. Del Vecchio, and L. Glielmo. Control of switched Boolean control networks by state feedback. In *2019 18th European Control Conference (ECC)*, pages 1999–2004. IEEE, 2019.

[47] A. Yerudkar, C. Del Vecchio, and L. Glielmo. Output tracking control of probabilistic Boolean control networks. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2109–2114. IEEE, 2019.

[48] A. Yerudkar, C. Del Vecchio, and L. Glielmo. Sampled-data set stabilization of switched Boolean control networks. *IFAC-PapersOnLine*, 53 (2):6139–6144, 2020.

[49] R. Zhou, Y. Guo, Y. Wu, and W. Gui. Asymptotical feedback set stabilization of probabilistic Boolean control networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4524–4537, 2019.

[50] S. Zhu, J. Lu, Y. Liu, T. Huang, and J. Kurths. Output tracking of probabilistic Boolean networks by output feedback control. *Information Sciences*, 483:96–105, 2019.