Received October 20, 2020, accepted October 28, 2020, date of publication November 2, 2020, date of current version November 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035152

Double Deep-Q Learning-Based Output Tracking of Probabilistic Boolean Control Networks

ANTONIO ACERNESE, (Student Member, IEEE), AMOL YERUDKAR, (Member, IEEE), LUIGI GLIELMO, (Senior Member, IEEE), AND CARMEN DEL VECCHIO, (Member, IEEE)
Department of Engineering, University of Sannio, 82100 Benevento, Italy

Corresponding author: Antonio Acernese (antonio.acernese@unisannio.it)

ABSTRACT In this article, a reinforcement learning (RL)-based scalable technique is presented to control the probabilistic Boolean control networks (PBCNs). In particular, a double deep-Q network (DDQN) approach is firstly proposed to address the output tracking problem of PBCNs, and optimal state feedback controllers are obtained such that the output of PBCNs tracks a constant as well as a time-varying reference signal. The presented method is model-free and offers scalability, thereby provides an efficient way to control large-scale PBCNs that are a natural choice to model gene regulatory networks (GRNs). Finally, three PBCN models of GRNs including a 16-gene and 28-gene networks are considered to verify the presented results.

INDEX TERMS Double deep-Q learning, model-free technique, output tracking, probabilistic Boolean control networks, scalability.

I. INTRODUCTION

Reinforcement learning (RL) provides effective strategies to bridge the gap between model-based and model-free controls, by combining features of optimal control and adaptive control. In particular, optimal controllers are normally designed offline by solving Hamilton-Jacobi-Bellman (HJB) equations, using complete knowledge of the system dynamics; on the other hand, adaptive controllers learn online how to control an unknown system but usually are not optimal. The framework of RL allows designing adaptive controllers that learn online, without the full knowledge of the system dynamics, optimal feedback laws. Specifically, RL considers an agent-environment interplay, whereby the agent, i.e., the controller, receives feedback (*stimuli*) in the form of rewards or penalties by interacting with the environment, i.e., the system, and modifies its acting behavior accordingly. For this reason, RL can be seen as an *action-based learning*.

One framework for studying RL is based on Markov decision processes (MDPs), which is interpreted to be the environment. Based on the MDP and other frameworks, many RL algorithms have been studied extensively in literature, both model-free and model-based [1]–[6]. One of the first and most popular model-free methods for solving RL problems, namely the Q-learning (QL), was proposed by Watkins in 1989 [7]. This technique has been proved to be a powerful solution for controlling small-scale environments. However, when RL deals with large-scale MDPs, i.e., the case of more realistic scenarios, QL turns out vulnerable in terms of slow or no convergence towards an optimal solution. The issue of large state-action space is known as the *curse of dimensionality*. One of the most productive ways to solve this limitation is the combination of QL and machine learning techniques. Recently, deep neural networks spurred significant interest in the scientific community and Google DeepMind [8] proposed deep-Q network (DQN), a combination of deep learning and QL. Moreover, Mnih *et al.* [8] introduced two main concepts in DQN, namely experience replay and target network thereby offering a viable solution to complex and high-dimensional MDP problems. Researchers have developed diverse applications based on the advantages of DQN. For instance, control of autonomous vehicles [9], [10], smart grids [11], disease identification [12] and gene regulatory networks (GRNs) [13]. Furthermore, van Hasselt *et al.* [14] proposed double DQN (DDQN) to overcome problems of overestimation, improving the convergence and performance of the DQN algorithm.

In this article, we consider an MDP as the underlying structure to model probabilistic Boolean control networks (PBCNs) [15], and study the output tracking problem of PBCNs by using DDQNs. A PBCN is a collection of Boolean control networks (BCNs) switching randomly between constituent BCNs with certain probability distribution. PBCNs offer a natural framework to model the development of

The associate editor coordinating the review of this manuscript and approving it for publication was Radu-Emil Precup.

cellular systems and large biological networks from a practical perspective. PBCN models of GRNs developed within the framework of MDPs have gained considerable attention from systems and control theorists. For example, Vahedi *et al.* [16] and Wu and Shen [17] designed optimal therapeutic methods by solving an optimal control problem.

Several control-theoretic problems have been investigated by various researchers for BCNs and PBCNs, see for example [18]–[29] and the references therein. Specifically, the output tracking problem so far has been thoroughly investigated. The main aim of the output tracking (or model reference control) problem is to design a controller that renders the system output to follow a constant or time-varying reference signal. Recently, the output tracking of PBCNs was studied in [30]–[32] to track a constant reference signal whereas Chen *et al.* [33] examined asymptotic time-varying tracking problem. The techniques in [30]–[33] resort to the semi-tensor product of matrices developed by Cheng *et al.* [34], [35]. These techniques are model-based and heavily rely on the qualitative model of the systems under consideration. However, such model is often unavailable or difficult to build [36]–[38] from laboratory experiments due to gigantic nature of GRNs. Furthermore, model-based techniques suffer due to exponential computational complexity and not suitable for large-scale PBCN models of GRNs. The development of new GRN (modeled as PBCN) control techniques that are model-free and computationally efficient for large networks is therefore crucial.

In the literature, model-free *Q*L [39], fitted *Q*L [40], [41] techniques have been employed to control GRNs. Authors in [42] recently proposed a DD*Q*N based intervention technique to address the controllability problem of probabilistic Boolean networks. Nonetheless, the above results focused only on altering the activity levels of specific genes by controlling one or more genes in the network and do not provide any specific feedback law to control the network. Lately, Acernese *et al.* [43] presented a *Q*L-based technique to stabilize small-scale PBCNs. However, the output tracking problem of PBCNs (both large and small-scale) using model-free techniques is still a significant open problem and indeed deserves further investigation.

Motivated by the above discussion, in this article we utilize a model-free DD*Q*N technique to investigate the output tracking problem of PBCNs. The main contributions of this article are as follows.

1) We pose PBCN dynamics in the MDP framework to investigate the output tracking problem. In our approach, the controller has no knowledge of the environment (the underlying transition probabilities of PBCNs).
2) We employ DD*Q*Ns with prioritized experience replay (PER) to formulate the output tracking problem and design optimal state feedback controllers rendering the output of PBCNs track constant as well as time-varying reference signals.
3) Further, we compare the performance of DD*Q*N + PER controllers with the traditional model-free *Q*L controllers.
4) Finally, to test the scalability of the presented results we consider three different models, i.e., 10-gene, 16-gene and 28-gene PBCN models of GRNs and validate the main results presented in the paper through a computer simulation.

The rest of the paper is organized as follows.

First, we state the set of preliminaries required for defining the problem statement. In the same Section II, we explain RL fundamentals, i.e., MDP, *Q*L and DD*Q*L, to help reader motivate in the direction of the research. In Section III, we propose a DD*Q*L algorithm for the output tracking objectives. Finally, we use software tools in Section IV to illustrate the ideas mentioned in Section III of the manuscript.

II. PRELIMINARIES AND PROBLEM FORMULATION

In this section, we formalize the problem of RL as the optimal control of MDPs. We also refer to the *Q*L methodology and its variants for solving the MDP problem.

Notation. $\mathbb{R}$ and $\mathbb{Z}_+$ denote the sets of real numbers and nonnegative integers, respectively. $\mathcal{B} := \{0, 1\}$, and $\mathcal{B}^n := \underbrace{\mathcal{B} \times \ldots \times \mathcal{B}}_{n}$. The basic logical operators Negation, And, Or are denoted by $\neg$, $\wedge$, $\vee$, respectively.

A. MARKOV DECISION PROCESS

A discrete-time Markov decision process (MDP) [44] is a quadruple $(\mathbf{X}, \mathbf{U}, \mathbf{P}, \mathbf{G})$, where $\mathbf{X}$ is the state-space, $\mathbf{U}$ is the action-space, and $\mathbf{P} : \mathbf{X} \times \mathbf{U} \times \mathbf{X} \rightarrow [0, 1]$ is the function of *state-transition probabilities* that describes, for each state $x_t \in \mathbf{X}$ and action $u_t \in \mathbf{U}$, 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. We consider the index $t \in \mathbb{Z}_+$ to be the discrete time-step, and state and action values at t are x_t and u_t, respectively. Moreover, let $\mathbf{G} : \mathbf{X} \times \mathbf{U} \times \mathbf{X} \rightarrow \mathbb{R}$ 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 $g_{t+1} = g_{t+1}(x_t, u_t, x_{t+1})$ and $\mathbb{E}[\cdot]$ is the expected value operator.

Define a performance index at time-step t as the sum of future costs,

$$\mathcal{J}_t := \sum_{i=t+1}^{\infty} \gamma^{i-t-1} g_i, \tag{1}$$

where $0 \leq \gamma < 1$ is the discount factor weighting costs along the trajectories. The basic RL problem is to find a closed-loop control or policy $\pi : \mathbf{X} \times \mathbf{U} \rightarrow [0, 1]$ that minimizes the long-term expectation of the performance index (1), i.e., $\mathbb{E}_\pi[\mathcal{J}_t]$, at each t. We refer to the optimal policy as $\pi^*(x_t, u_t)$, $\forall x_t \in \mathbf{X}$, $\forall u_t \in \mathbf{U}$. If π 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 π, the value function of the state x_0 is defined in terms of the expected future cost as:

$$v_\pi(x_0) := \mathbb{E}_\pi\left[\sum_{i=1}^{\infty}\gamma^{i-1}g_i\middle|x_0\right], \text{ for all } x_0 \in \mathbf{X}. \quad (2)$$

A fundamental property of $v_\pi(\cdot)$ is that it satisfies the recursive Bellman equation:

$$v_\pi(x_t) = \sum_{u_t\in\mathbf{U}}\pi(u_t|x_t)\sum_{x\in\mathbf{X}}\mathsf{P}^{u_t}_{x_t,x}\left[\mathbf{G}^{u_t}_{x_t,x} + \gamma v_\pi(x)\right], \quad (3)$$

for all $x_t \in \mathbf{X}$, where $\pi(u_t|x_t)$ represents the conditional probability of taking the action u_t given that the system is in the state x_t. Similarly, given a state x_t, an action u_t and following the policy π thereafter, let the action-value function be defined as $q_\pi(x_t, u_t) := \mathbb{E}_\pi[\mathcal{J}_t|x_t, u_t]$. It can be shown that also $q_\pi(x_t, u_t)$ takes a recursive form similar to (3), namely:

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

for all $x_t \in \mathbf{X}$ and $u_t \in \mathbf{U}$.

The policy that minimizes the value function for any initial state is the optimal policy, namely $\pi^*(x_t, u_t) := \arg\min_{\pi\in\Pi} v_\pi(x_t)$, $\forall x_t \in \mathbf{X}$, where Π is the set of all admissible policies, and the corresponding optimal value function is $v^*(x_t) := v_{\pi^*}(x_t)$. It can be shown [1] that (3) and (4) can be expressed in terms of the Bellman optimality equation as:

$$v^*(x_t) = \min_{u_t\in\mathbf{U}}\sum_{x\in\mathbf{X}}\mathsf{P}^{u_t}_{x_t,x}\left[\mathbf{G}^{u_t}_{x_t,x} + \gamma v^*(x)\right], \quad (5)$$

$$q^*(x_t, u_t) = \sum_{x\in\mathbf{X}}\mathsf{P}^{u_t}_{x_t,x}\left[\mathbf{G}^{u_t}_{x_t,x} + \gamma\min_u q^*(x, u)\right], \quad (6)$$

where $v^*(x_t) = \min_{u_t} q^*(x_t, u_t)$. Given a solution $q^*(x_t, u_t)$, one can obtain an optimal deterministic policy as:

$$\mu^*(x_t) := \arg\min_{u\in\mathbf{U}} q^*(x_t, u), \text{ for all states } x_t \in \mathbf{X}. \quad (7)$$

Note that in case there exist more than one optimal deterministic policy, they will share the same action-value function.

When a complete knowledge of an MDP is available, several algorithms referred to the general term of dynamic programming (DP) can be used to compute exact solutions for the optimal policies, e.g., value iteration and policy iteration methods, [45]. In this article, we consider the more realistic case where the transition probability distribution is unknown to the agent, or where it is known but it is too complex to be represented. In such cases, an approach to solve the MDP problem is to approximate the expected value of a state-action pair q_π with an estimation $\widetilde{Q} : \mathbf{X}\times\mathbf{U} \to \mathbb{R}$ computed along trajectories. This idea is known as temporal-difference (TD) learning and it resorts to bootstrapping [1].

B. Q-LEARNING ALGORITHM

The original QL [7], [46] is an off-policy TD control algorithm in which, given a pair (x_t, u_t), then $q_\pi(x_t, u_t)$ is estimated as $\widetilde{Q}(x_t, u_t) = g_{t+1} + \gamma\,\widetilde{Q}_\pi(x_{t+1}, \mu(x_{t+1}))$. The estimates are improved at each time-step t with the following update rule:

$$\begin{aligned}\widetilde{Q}_{t+1}(x_t, u_t) &= \widetilde{Q}_t(x_t, u_t) + \alpha_t[TDE_{t+1}],\\ TDE_{t+1} &= g_{t+1} + \gamma\min_{u\in\mathcal{U}}\widetilde{Q}_t(x_{t+1}, u) - \widetilde{Q}_t(x_t, u_t),\end{aligned} \quad (8)$$

where the quantity TDE_{t+1} can be seen as an estimation error given by the difference between the old estimated value $\widetilde{Q}_t(x_t, u_t)$ and the new better estimate $g_{t+1} + \gamma\min_u\widetilde{Q}_t(x_{t+1}, u)$; moreover, the learning rate or step-size rule $0 < \alpha_t \leq 1$ is a time-varying parameter responsible for how much the learning process is affected by newly acquired information. Under the assumption that all states are visited infinitely often and some other common stochastic approximation conditions on the sequence α_t, QL has been shown to converge toward the optimal solution $\widetilde{Q}^*(x_t, u_t) := q^*(x_t, u_t)$, $\forall x_t \in \mathbf{X}$, $\forall u_t \in \mathbf{U}$, w.p.1 [46]. The problem of maintaining sufficient exploration can be addressed by choosing the actions u_t in a proper way, e.g., by following an ϵ-greedy policy, i.e., choosing a greedy action $u_t = \arg\min_{u\in\mathbf{U}}\widetilde{Q}_t(x_t, u)$ w.p.$(1-\epsilon)$, and a random action $u_t = \texttt{rand}(\mathbf{U})$ w.p.ϵ, where $\texttt{rand}(\cdot)$ is the discrete uniform distribution.

C. DOUBLE DEEP-Q NETWORK

QL creates a look-up table for all state-action pairs, thus requiring a big amount of memory when $|\mathbf{X}\times\mathbf{U}|$ is very large, e.g., in continuous time problems and in large GRNs, because the state-action space grows exponentially in the number of genes and inputs. To overcome this tabular representation limit, function approximators emerged as an alternative. In the case of Deep Q-Learning, an artificial neural network (ANN) is used to create a parameterized model that estimates online the action-value function. In particular, a DQN is a multi-layered ANN that provides an estimation $\widetilde{Q} : \mathbf{X}\times\mathbf{U}\times\mathcal{W} \to \mathbb{R}$ of $q_\pi(\cdot,\cdot)$, where $\mathcal{W}$ is the set of "tunable" parameters that is responsible of how good is the approximation.[1] Given a state x_t and some set of values for $\mathcal{W}$, the output of a DQN is a vector of approximated action-values, one for each possible action u:

$$\begin{aligned}\widetilde{Q}(x_t, u, \mathcal{W}) = \psi^{(L)}(W^{(L)}(\ldots\psi^{(2)}(W^{(2)}\psi^{(1)}(W^{(1)}x_t\\ + b^{(1)}) + b^{(2)})\ldots) + b^{(L)}),\end{aligned} \quad (9)$$

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, respectively. We refer to the number of neurons for each layer as $\{n^{(l)}\}_{l=0}^{L}$, i.e., $n^{(L)} = |\mathbf{U}|$. Using ANNs to approximate action-value functions, the goal of DQNs become to

[1]With a slight abuse of notation, we use the same symbol $\widetilde{Q}$ for both QL and DQN to emphasize the fact that both methodologies approximate the same function $q_\pi(\cdot,\cdot)$.

learn the network parameters $\mathcal{W} = \{W^{(l)}, b^{(l)}\}_{l=0}^{L}$ such that $\widetilde{Q}(x_t, u_t, \mathcal{W})$ converges toward the optimal $q^*(x_t, u_t)$, for all $x_t \in \mathbf{X}$ and $u_t \in \mathbf{U}$. Thus, given a tuple $(x_t,\ u_t,\ g_{t+1},\ x_{t+1})$, in D*Q*N the parameters of the network are updated to minimize a differentiable loss function, namely the Bellman error

$$\mathscr{L}(\mathcal{W}) = \frac{1}{2}\left|\left|\widetilde{Q}(x_t, u_t, \mathcal{W}) - y_{t+1}\right|\right|^2, \tag{10}$$

i.e., the error between the current estimated state-action value and the target value $y_{t+1} = g_{t+1} + \gamma \min_{u \in \mathcal{U}} \widetilde{Q}(x_{t+1}, u, \mathcal{W})$. Then, the training of the network, i.e., the update of $\mathcal{W}$, can be performed through stochastic gradient descent (SGD) method:

$$\begin{aligned}\mathcal{W} &= \mathcal{W} - \alpha \nabla_{\mathcal{W}} \mathscr{L}(\mathcal{W}) \\ &= \mathcal{W} - \alpha\big[\widetilde{Q}(x_t, u_t, \mathcal{W}) - y_{t+1}\big]\nabla_{\mathcal{W}}\widetilde{Q}(x_t, u_t, \mathcal{W}). \end{aligned}\tag{11}$$

Despite the important capabilities to deal with huge state-action space problems, convergence properties of D*Q*N is still an open issue to research community. This limit is mainly linked to two problems; firstly, SGD method assumes that samples are uncorrelated but, following (11), the network parameters are sequentially updated online over tuples $\{(x_t, u_t, g_{t+1}, x_{t+1})\}_{t=0,1,\ldots}$, that in general are temporally correlated, thus leading to locally over-fit data to each region of the state space. Furthermore, the reference value y_{t+1} in (11) depends on the ANN parameters $\mathcal{W}$, and consequently its value changes over time-steps.

An important technique to mitigate these stability issues is the use of experience replay (ER) [47]. Specifically, let $\mathcal{M}$ be a data-set of transitions $\{(x_j,\ u_j,\ g_{j+1},\ x_{j+1})\}_{j=0,1,\ldots,|\mathcal{M}-1|}$ that the agent stored. Then, the updates can be performed by sampling uniformly a mini-batch $\mathcal{M}_1$ from $\mathcal{M}$, $\mathcal{M}_1 \subseteq \mathcal{M}$, and by considering the following loss function:

$$\mathscr{L}'(\mathcal{W}) = \mathbb{E}_{(\bar{x}_j, \bar{u}_j, \bar{g}_{j+1}, \bar{x}_{j+1}) \in \mathcal{M}}\left[\sum_{j=0}^{|\mathcal{M}_1-1|} \mathscr{L}_j(\mathcal{W})\right], \tag{12}$$

where in the sequel, the notation $\bar{(\cdot)}_j$ refers to the j-th experience and not to the value at time-step j, and $\mathscr{L}_j(\cdot)$ is the loss function (10) computed on the experience j. Thus, if $\mathcal{M}$ is large, ER is close to sample independent transitions from an exploratory policy. Moreover, Schaul *et al.* in [48] introduced the prioritized experience replay (PER), resorting to the idea that an agent can learn more from some experiences than others. Let $\bar{\phi}_{j+1} := \widetilde{Q}(\bar{x}_j, \bar{u}_j, \mathcal{W}) - \bar{y}_{j+1}$ be the error of an experience $(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1})$. Consequently, the probability of sampling such tuple from the data-set $\mathcal{M}$ can be computed with the proportional prioritization criterion, namely:

$$\mathsf{P}\{(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1}) \in \mathcal{M}\} = \frac{(\bar{p}_{j+1})^{\omega}}{\sum_{k=0}^{|\mathcal{M}-1|}(\bar{p}_{k+1})^{\omega}}, \tag{13}$$

where $\bar{p}_{j+1} := |\bar{\phi}_{j+1}| + \zeta$, ω determines the magnitude of prioritization — $\omega = 0$ corresponds to the uniform ER — and ζ is a small positive constant. However, PER creates a bias in learning because the experiences with high errors are sampled more often. To anneal this bias, one can use the weighted importance-sampling (W-IS) [49]:

$$\bar{\theta}_{j+1} = \left(\frac{1}{|\mathcal{M}|}\frac{1}{\mathsf{P}\{\bar{(\cdot)}_j \in \mathcal{M}\}}\right)^{\beta}, \tag{14}$$

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

In addition to ER and PER, another important suggestion to mitigate the problem that the target value y_{t+1} changes over time-steps in (11), is the implementation of DD*Q*N [14]. The traditional Double *Q*L [50] addresses the problem of overestimation of $\widetilde{Q}(\cdot, \cdot)$ linked to the minimum operator in (8) and prevents the instabilities to propagate quickly by using a double estimator, namely $\widetilde{Q}^A$ and $\widetilde{Q}^B$. The learning mechanism is the same of traditional *Q*L but, at each time-step t, one of the two estimators is selected randomly for the evaluation of u_t and then it is updated in terms of the other one. This methodology can be extended to DD*Q*N, using two networks: an online network, with parameters $\mathcal{W}$, that is responsible for choosing the policy, and a target network, with parameters $\mathcal{W}^-$, which is used for the evaluation of the current action. Thus, given a mini-batch of transitions $\{(\bar{x}_j,\ \bar{u}_j,\ \bar{g}_{j+1},\ \bar{x}_{j+1})\}_{j=0,1,\ldots,|\mathcal{M}_1-1|}$, the parameter updates (11) in DD*Q*N + PER with W-IS is replaced with:

$$\mathcal{W} = \mathcal{W} - \alpha \sum_{j=0}^{|\mathcal{M}_1-1|} \bar{\theta}_{j+1}\bar{\phi}'_{j+1}\nabla_{\mathcal{W}}\widetilde{Q}(\bar{x}_j, \bar{u}_j, \mathcal{W}), \tag{15}$$

where

$$\begin{aligned}\bar{\phi}'_{j+1} = [&\widetilde{Q}(\bar{x}_j, \bar{u}_j, \mathcal{W}) - \bar{g}_{j+1} + \\ &- \gamma \widetilde{Q}(\bar{x}_{j+1}, \arg\min_{\bar{u} \in \mathcal{U}} \widetilde{Q}(\bar{x}_{j+1}, \bar{u}, \mathcal{W}), \mathcal{W}^-)].\end{aligned}\tag{16}$$

The update rule of the target network can be carried out in different ways. The first choice [14], keeps $\mathcal{W}^-$ constant over time-steps and every k iterations hard updates them as $\mathcal{W}^- = \mathcal{W}$. Another possibility, introduced in [51], is 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.

III. OUTPUT TRACKING PROBLEM

In this Section, we introduce the PBCNs followed by some definitions. Further, we structure the problem of finding a near-optimal solution to the output tracking control problem of PBCNs in a model-free context using DD*Q*N + PER.

A. PROBABILISTIC BOOLEAN CONTROL NETWORKS

A PBCN with n nodes, m control inputs and q outputs is defined as follows:

$$\begin{cases}\mathcal{X}^i(t+1) = f_i^{\sigma(t)}\left(\mathcal{U}(t),\ \mathcal{X}(t)\right), & i = 1, \ldots, n, \\ \mathcal{Y}^j(t) = h_j(\mathcal{X}(t)), & j = 1, \ldots, q,\end{cases}\tag{17}$$

where $\mathcal{X}(t) = \mathcal{X}_t := \left(\mathcal{X}^1(t), \dots, \mathcal{X}^n(t)\right) \in \mathcal{B}^n$ is the state, $\mathcal{U}(t) = \mathcal{U}_t := \left(\mathcal{U}^1(t), \dots, \mathcal{U}^m(t)\right) \in \mathcal{B}^m$ is the control input, $\mathcal{Y}(t) = \mathcal{Y}_t := \left(\mathcal{Y}^1(t), \dots, \mathcal{Y}^q(t)\right) \in \mathcal{B}^q$ is the output, and $f_i \in \mathcal{F}_i = \{f_i^1, f_i^2, \dots, f_i^{l_i}\} : \mathcal{B}^{n+m} \to \mathcal{B}$, $i = 1, \dots, n$, are logical functions randomly chosen with probability $\{\mathsf{P}_i^1, \mathsf{P}_i^2, \dots, \mathsf{P}_i^{l_i}\}$, where $\sum_{j=1}^{l_i} \mathsf{P}_i^j = 1$ and $\mathsf{P}_i^j \geq 0$. $h_j : \mathcal{B}^n \to \mathcal{B}$, $j = 1, 2, \dots, q$, are deterministic logical functions. We denote by $\Lambda = \prod_{i=1}^n l_i$, the total number of sub-networks of (17). Switching among sub-networks is governed by the switching signal $\sigma(t) \in [1, \Lambda]$, that is an independently and identically distributed (i.i.d.) process. We assume that the selection of a sub-network for each logical function f_i is independent of time and current state. 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 (17) by $\mathcal{X}(t;\ \mathcal{X}(0),\ \boldsymbol{\mathcal{U}}_t)$. 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 $\mathsf{P}\{\mathcal{X}(1;\ \mathcal{X}(0),\ \mathcal{U}(0)) = \mathcal{X}(0)\} = 1$. We denote by $\bar{\mathcal{X}}_j,\ 1, \leq j \leq 2^n$, the j-th state in $\mathcal{B}^n$ and $\bar{\mathcal{U}}_l,\ 1 \leq l \leq 2^m$, the l-th input in $\mathcal{B}^m$, without referring to the particular time-step t.

Definition 1: A PBCN (17) is said to be asymptotically stabilizable at a given equilibrium point $\bar{\mathcal{X}}_d \in \mathcal{B}^n$ in distribution, if for every $\mathcal{X}_0 \in \mathcal{B}^n$ there exists $\boldsymbol{\mathcal{U}}_t$ such that $\lim_{t\to\infty} \mathsf{P}\{\mathcal{X}(t;\ \mathcal{X}_0,\ \boldsymbol{\mathcal{U}}_t) = \bar{\mathcal{X}}_d\} = 1$.

B. MODEL-FREE OUTPUT TRACKING FOR PBCNs

Here, we cast the output tracking problem of PBCNs into model-free RL framework and provide a general algorithm to design a near-optimal state feedback control law.

Let us consider a general PBCN (17) as a model of GRNs. It can be represented in an MDP framework by the tuple $(\mathcal{B}^n,\ \mathcal{B}^m,\ \mathbf{P},\ \mathbf{R})$. Assuming that system dynamics $\mathbf{P}$ is unknown to the agent, our aim is to force the concentration of one or more proteins (observed as an output) to follow an *a priori* defined reference signal rendering the GRN stabilized at a desired or healthy state(s). Particularly, we use a constant reference signal and a reference signal with finite number of states occurring in periodic or aperiodic manner. Let $\bar{\mathcal{B}}^n \subseteq \mathcal{B}^n$ be a subset of the state-space, and $\mathcal{X}_r(t) = \mathcal{X}_{r_t} \in \bar{\mathcal{B}}^n$ be the known tracking signal, whose dynamics is given as follows:

$$\mathcal{X}_r(t+1) = \mathbf{g}(\mathcal{X}_r(t)). \tag{18}$$

Without loss of generality, we assume that Definition 1 is valid for every state in the reference signal. Under this assumption, we define the controller of the system as an ANN that at each time-step t takes as input the set of features $\mathcal{Z}(t) := \{\mathcal{X}(t);\ \mathcal{X}_r(t)\} \in \mathcal{B}^n$ and outputs a vector of action-value estimates, one for each control input, namely $\widetilde{Q}(\mathcal{Z}_t, \mathcal{U}, \mathcal{W})$, $\forall \mathcal{U} \in \mathcal{B}^m$. Thus, given a state $\mathcal{X}_t$, and given the current target state $\mathcal{X}_{r_t}$, the aim of the agent is to find the deterministic control policy $\mu(\mathcal{Z}_t) = \underset{\mu(\mathcal{Z}_t)\in\mathcal{B}^m}{\arg\min}\ \widetilde{Q}(\mathcal{Z}_t, \mu(\mathcal{Z}_t), \mathcal{W})$ (in the sequel referred as a state feedback law) that minimizes the long-term expectation of a user-defined performance index $\mathbb{E}_\pi[\mathcal{J}_t]$. The definition of such cost function plays a crucial role in the efficacy of the control and it must be chosen according to the problem requirements. In the specific case of output tracking control problem, let the cost be defined at each t as $g_{t+1} = g_{t+1}(\mathcal{Z}_t, \mathcal{U}_t, \mathcal{Z}_{t+1}) = \mathcal{A}(\mathcal{Z}_{t+1}) + [1 + \mathcal{A}(\mathcal{Z}_{t+1})]\mathcal{C}(\mathcal{X}_t, \mathcal{X}_{t+1})$. The function $\mathcal{A}(\cdot)$ is responsible for steering the state $\mathcal{X}_t$ to the next reference state $\mathbf{g}(\mathcal{X}_{r_t})$, and it is equal to -1 if $\mathcal{X}_{t+1} = \mathbf{g}(\mathcal{X}_{r_t})$, 1 otherwise. Moreover, the function $\mathcal{C}(\cdot, \cdot)$ aims at avoiding undesired self-loops along the trajectory, thus a suitable choice to this aim is

$$\mathcal{C}(\cdot,\cdot) = \begin{cases} 1 & \text{if } d_H(\mathcal{X}_t, \mathcal{X}_{t+1}) = 0 \\ 0 & \text{otherwise,} \end{cases}$$

where $d_H(\cdot,\cdot)$ is the Hamming distance. With a slight abuse of notation we represent the cost as a function of the input features' set $\mathcal{Z}$, implicitly including its dependence on both $\mathcal{X}$ and $\mathcal{X}_r$.

The selection of such values for the cost is quite intuitive; the value 1 for each state with an undesired self-loop indicates a strong penalization because PBCN could be stalled in that state indefinitely. The assignment of 0 to non-significant states is to penalize the agent for not tracking the reference signal, however it is encouraged (with respect to the previous case) to find actions that will lead the network towards the desired state as fast as possible. For such choice of the cost function, the model-free optimal tracking control problem can be formulated as:

$$\min_{\mu(\cdot)} \mathbb{E}_\mu \left[\sum_{t=0}^{\infty} \gamma^t g_{t+1}(\mathcal{Z}_t,\ \mathcal{U}_t,\ \mathcal{Z}_{t+1}) \right],\ \forall \mathcal{Z}_0 \in \mathcal{B}^n \quad \text{subject to systems (17) and (18).} \tag{19}$$

With this settings, the agent is guided to find deterministic policies $\mu(\mathcal{Z}_t)$ that can steer the state of the system $\mathcal{X}_t$ along the reference trajectory $\mathcal{X}_{r_t}$, while minimizing the cost associated to the action performed, i.e., finding the expected optimal shortest path.

To solve the optimization problem (19) in a model-free context, we use an episodic framework. The RL term "*episode*" represents the interactions between the agent and the system from an initial state $\mathcal{X}_0$ towards a terminal condition, e.g., $\mathcal{X}_T = \mathcal{X}_{r_T}$, by following some policy that achieves *exploration*, i.e., the random selection of various actions at various states. It is worth to note that, depending on the particular policy and on the system dynamics, the final time-step T can change along the episodes.

Thus, for each features' set $\mathcal{Z}_t$, DDQN + PER estimates the action-values $\widetilde{Q}(\mathcal{Z}_t, \mathcal{U}, \mathcal{W})$ through a forward propagation step and then it improves the parameters of the model $\mathcal{W}$ by applying an SGD step as in (15). After the completion of the training phase, that lasts for N episodes, the optimal output tracking control law of the system (17) with unknown dynamics can be determined using the target network as:

$$\mu^*(\bar{\mathcal{Z}}_j) = \underset{\bar{\mathcal{U}}\in\mathcal{B}^m}{\arg\min}\ \widetilde{Q}^*(\bar{\mathcal{Z}}_j, \bar{\mathcal{U}}, \mathcal{W}^-), \tag{20}$$

for each pair $\bar{\mathcal{Z}}_j = \{\bar{\mathcal{X}}_j, \bar{\mathcal{X}}_{r_j}\},\ \bar{\mathcal{X}}_j \in \mathcal{B}^n,\ \bar{\mathcal{X}}_{r_j} \in \bar{\mathcal{B}}^n$.

Algorithm 1 PBCN Output Tracking Using DDQN + PER

```
Input: $\alpha$, $\gamma$, $\epsilon$-greedy policy, $N$, mini-batch size $|\mathcal{M}_1|$, $\omega$, $\tau$, $\beta$
Output: $\mu^*(\bar{\mathcal{Z}}_t)$, $\forall \bar{\mathcal{X}}_t \in \mathcal{B}^n$, $\forall \bar{\mathcal{X}}_{r_t} \in \bar{\mathcal{B}}^n$
 1: Initialize weights $\mathcal{W} \leftarrow \texttt{rand}(\{0,1\})$, $\mathcal{W}^- \leftarrow \mathcal{W}$
 2: Initialize the replay memory $\mathcal{M} \leftarrow \emptyset$, $\Delta \leftarrow 0$
 3: for $ep = 0, 1, \dots, N-1$ do
 4:     $t \leftarrow 0$, $\mathcal{X}_t \leftarrow rand(\mathcal{B}^n)$, $\texttt{read}(\mathcal{X}_{r_t})$
 5:     while $\mathcal{X}_t \neq \mathcal{X}_{r_t}$ do
 6:         Choose $\mathcal{U}_t$ using $\epsilon$-greedy policy
 7:         $\texttt{apply}(\mathcal{U}_t)$, $\texttt{read}(\mathcal{X}_{t+1})$, $\texttt{read}(g_{t+1})$, create features $\mathcal{Z}_t \leftarrow \{\mathcal{X}_t, \mathcal{X}_{r_t}\}$
 8:         Store transition $(\mathcal{Z}_t,\ \mathcal{U}_t,\ g_{t+1},\ \mathcal{Z}_{t+1})$ in $\mathcal{M}$ with maximal priority $p_{t+1} \leftarrow \max_{i\in\mathcal{M}} p_i$
 9:         if $|\mathcal{M}| \geq |\mathcal{M}_1|$ then
10:             for $j = 0, 1, \dots, |\mathcal{M}_1| - 1$ do
11:                 Using (13), sample transition $j$ with probability $\mathsf{P}\{j \in \mathcal{M}\} \leftarrow \frac{\bar{p}_{j+1}^{\omega}}{\sum_i (\bar{p}_{i+1})^{\omega}}$
12:                 According to (14), compute normalized IS weight $\bar{\theta}_{j+1} \leftarrow \frac{(|\mathcal{M}|\mathsf{P}\{j \sim \mathcal{M}\})^{-\beta}}{\max_{i\in\mathcal{M}} \omega_i}$
13:                 Compute TD-error through (16) $\bar{\phi}'_{j+1} \leftarrow \widetilde{Q}(\bar{\mathcal{Z}}_j, \bar{\mathcal{U}}_j, \mathcal{W}) - \bar{g}_{j+1} - \gamma \widetilde{Q}(\bar{\mathcal{Z}}_{j+1}, \arg\min_{\bar{\mathcal{U}}} \widetilde{Q}(\bar{\mathcal{Z}}_{j+1}, \bar{\mathcal{U}}, \mathcal{W}), \mathcal{W}^-)$
14:                 Update transition priority $\bar{p}_{j+1} \leftarrow |\bar{\phi}'_{j+1}| + \zeta$
15:                 $\Delta \leftarrow \Delta + \bar{\theta}_{j+1} \bar{\phi}'_{j+1} \nabla_{\mathcal{W}} \widetilde{Q}(\bar{\mathcal{Z}}_j, \bar{\mathcal{U}}_j, \mathcal{W})$
16:             end for
17:             Using (15), update network weights $\mathcal{W} \leftarrow \mathcal{W} - \alpha\Delta$,  $\Delta \leftarrow 0$
18:             Update target network weights $\mathcal{W}^- \leftarrow (1-\tau)\mathcal{W}^- + \tau\mathcal{W}$
19:         end if
20:         $t \leftarrow t + 1$
21:     end while
22: end for
23: return $\mu^*(\bar{\mathcal{Z}}_t) \leftarrow \arg\min_{\bar{\mathcal{U}}} \widetilde{Q}(\bar{\mathcal{Z}}_t, \bar{\mathcal{U}}, \mathcal{W}^-)$, $\forall \bar{\mathcal{X}}_t \in \mathcal{B}^n$,  $\forall \bar{\mathcal{X}}_{r_t} \in \bar{\mathcal{B}}^n$
```

It is worth to note that the optimal solution (20) refers to DDQN + PER approach, that has the advantage to accept as input a set of features rather than a single one; in case the traditional QL algorithm is used, as in [43] for a stabilization problem of PBCNs, the action-value function will be expressed as $\widetilde{Q}(\mathcal{X}_t, \mathcal{U}_t)$, thus requiring multiple training sessions, one for each possible value of $\mathcal{X}_r \in \bar{\mathcal{B}}^n$, exhibiting weaker generalization properties with respect to function approximators.

Remark 1: It is worth to point out that DDQN + PER algorithm estimates the action-value function relying on the features' set $\mathcal{Z}_t$, that includes both the state and the current value of the tracking signal. Thus, once an optimal policy is obtained, such policy can be used to track different signals $\mathcal{X}_{r_t}$ independently on $\mathbf{g}(\cdot)$, given that $\mathcal{X}_{r_t} \in \bar{\mathcal{B}}^n$.

Based on the above discussion, we now introduce in Algorithm 1 the DDQN + PER procedure to design near-optimal tracking controllers based on real system data such that the PBCNs (17) track given reference signals.

Note that in the algorithm, N represents the total number of episodes, that is defined by the user, and a linearly decaying ϵ-*greedy* policy is used to choose the actions. Moreover, we apply a soft-update of the target network for DDQN part.

The control scheme corresponding to Algorithm 1 is shown in FIGURE 1; it includes the general agent-environment communication as well as the target network and the replay memory roles, without the explicit reference to the kind of prioritization and importance sampling.

C. COMPUTATIONAL COMPLEXITY

DDQN + PER algorithm is a more advanced version of the basic QL algorithms and uses ANNs to approximate the action-value function, overcoming the problem of high state-action spaces and leading better generalization. Despite these advantages, DDQN + PER algorithm takes longer time to train per episode than QL algorithm. This is linked to the fact that in the latter case at each time-step the agent has to choose out of 2^m control actions the one that gives maximum Q-value. Thus, the computational burden involved in this part is $O(2^m)$. Further, this operation must be performed at most for T steps in each of the N episodes, leading to the overall computational complexity of $O(NT2^m)$. Instead, in DDQN + PER case, at the above mentioned operations must be added to the complexity linked to the prioritization and to the update of the parameters of the network. Thus, the complexity of Algorithm 1 significantly increases.

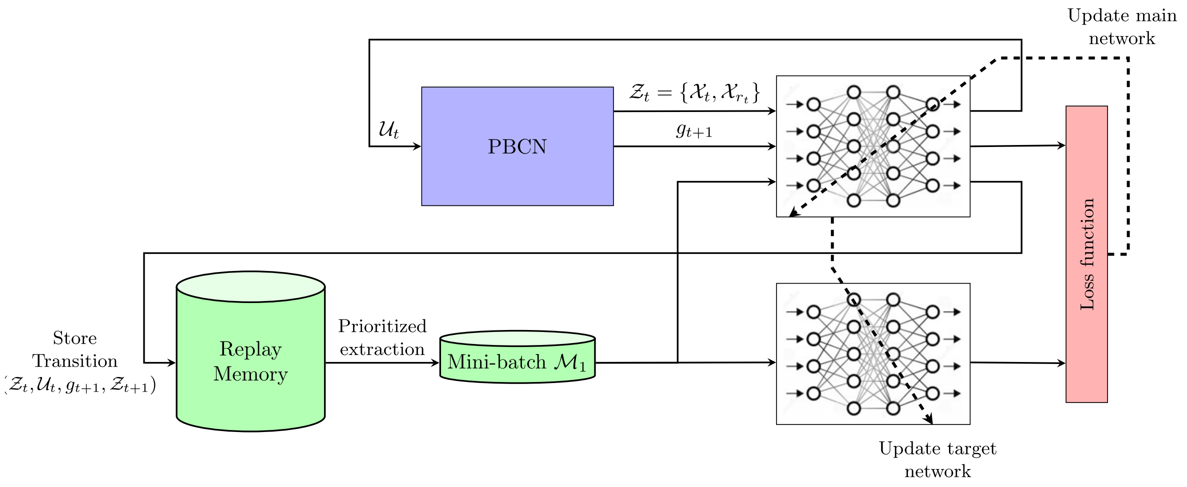

Update main network
$\mathcal{Z}_t = \{\mathcal{X}_t, \mathcal{X}_{r_t}\}$
g_{t+1}
$\mathcal{U}_t$
PBCN
Loss function
Store Transition
$(\mathcal{Z}_t, \mathcal{U}_t, g_{t+1}, \mathcal{Z}_{t+1})$
Replay Memory
Prioritized extraction
Mini-batch $\mathcal{M}_1$
Update target network

FIGURE 1. **DDQN + PER control scheme. The transitions $(\mathcal{Z}_t, \mathcal{Z}_t, g_{t+1}, \mathcal{Z}_{t+1})$ are generated at each time-step t through the agent-environment communications, and are stored in the replay memory. Subsequently, a mini-batch $\mathcal{M}_1$ is extracted from the buffer using PER, and the data is sent to the ANNs in order to compute the loss function and to train the main network. Note that in the control scheme the dashed lines represent the action of updating the ANN parameters.**

On the same line, also the space complexity of Algorithm 1 is higher than the traditional QL. Assuming that in most of the PBCNs $m \ll n$, the space complexity hinges on the order of the Q-factors in the traditional method, i.e., $O(2^n)$, while in DDQN + PER it can be considered considerably higher. Indeed, even the tabular action-value function is replaced by a function, there is still a relevant space memory that is used to store the parameters $\mathcal{W}$ of such function, whose number can be large in case of deep ANNs. Moreover, there is the target network with the same number of weights of the online network, and the replay memory should be generally very large in order to guarantee independent samples.

In light of the above discussion, Algorithm 1 is not favorable with respect to QL in terms of training time and computational load, thus simpler algorithms are preferred in case of reasonably small-medium GRNs, e.g, up to 13-16 genes. However, one of the most important advantages of DDQN + PER algorithm is that, using function approximators instead of Q-tables, it can generalize well on very large GRNs, that other model-free and model-based methods are not able to handle. In the next section, this property will be shown through a GRN with 28 genes and 3 input genes, for a total number of state-action pairs equal to 2.15×10^9.

IV. SIMULATION RESULTS

In this section, we show how optimal solution to the tracking problem of PBCNs can be devised by using the results obtained in the previous section. We consider three PBCN models (10-gene, 16-gene and 28-gene) of GRNs to prove the efficacy of the presented algorithm. The obtained DDQN optimal policy for 10 and 16-gene networks is compared with the one obtained using QL. It is worthwhile to note that, we included PBCN dynamics to highlight the level of complexity of the networks. Controller designing technique presented in Algorithm 1 is completely model-free and do not utilize the knowledge of underlying network transition probabilities.

Example 2 (Periodic trajectory tracking): Consider the following PBCN model (21) of the lactose operon in the *Escherichia Coli* derived from [52], where $\mathcal{X}^i_+$, $i = 1, 2, \ldots, 10$, represents the state at next time-step.

We aim to design optimal state-feedback controllers such that the output $\mathcal{Y}$ tracks the reference signal wherein the gene $\mathcal{X}^4$ being ON-OFF periodically, i.e., $\mathcal{X}_{r_t} : \bar{\mathcal{B}}^n \rightarrow \{(0, 0, 0, 0, 1, 1, 0, 0, 0, 0); (0, 0, 0, 1, 1, 1, 0, 0, 0, 0)\}$.

$$\begin{aligned}
\mathcal{X}^1_+ &= \mathcal{X}^4 \wedge \neg\mathcal{X}^5 \wedge \neg\mathcal{X}^6\\
\mathcal{X}^2_+ &= \mathcal{X}^3_+ = \mathcal{X}^1\\
\mathcal{X}^4_+ &= \neg\mathcal{U}^1\\
\mathcal{X}^5_+ &= \neg\mathcal{X}^7 \wedge \neg\mathcal{X}^8\\
\mathcal{X}^6_+ &= \neg\mathcal{X}^7 \wedge \neg\mathcal{X}^8 \vee \mathcal{X}^5\\
\mathcal{X}^7_+ &= \mathcal{X}^3 \wedge \mathcal{X}^9\\
\mathcal{X}^8_+ &= \mathcal{X}^9 \vee \mathcal{X}^{10}\\
\mathcal{X}^9_+ &= \begin{cases} \mathcal{X}^2 \wedge \mathcal{U}^3 \wedge \neg\mathcal{U}^1), & \mathsf{P} = 0.6\\ \mathcal{X}^9, & \mathsf{P} = 0.4 \end{cases}\\
\mathcal{X}^{10}_+ &= ((\mathcal{U}^2 \wedge \mathcal{X}^2) \vee \mathcal{U}^3) \wedge \neg\mathcal{U}^1,\\
\mathcal{Y} &= \mathcal{X}^4.
\end{aligned} \tag{21}$$

By following the proposed approach, the state feedback controls using QL and DDQN+PER are obtained. FIGURE 2 depicts their comparison in terms of the cost paid to steer the initial state to the reference signal, as a function of the number of episodes, averaged over 1×10^3 episodes. Even if the two algorithms show slight different trends, both of them reach almost the same minimum value of the cost function, proving that both of them learn optimal strategies. However, as explained in Section III-C, the training time involved in DDQN + PER is higher than the one required by QL, i.e., 40 mins and 15 mins, respectively, highlighting the preference of simple QL when the system to be controlled is not too complex.

Using the optimal control laws obtained through DDQN + PER, FIGURE 4 and FIGURE 5 show the state transition

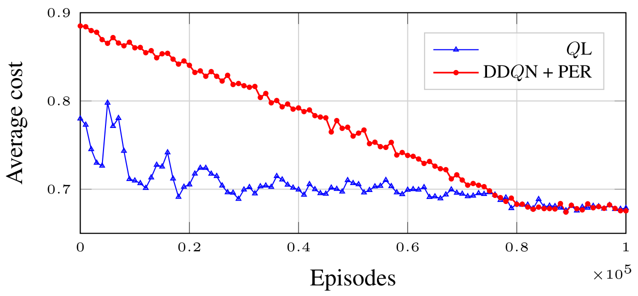


FIGURE 2. Average cost of the proposed Algorithm in 10-gene 3-input PBCN.

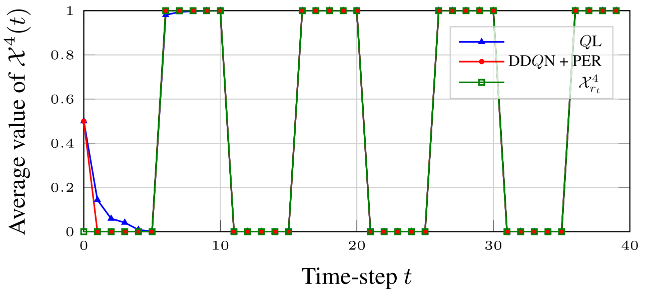


FIGURE 3. Average evolution of the genes in Example 2, over 1×10^3 episodes.

graphs of the networks under the action of the controller, one for each value of the reference signal. In both figures, the green point represents the desired value of the state, and the red one represents the other value of the desired signal, that is an undesired self-loop and needs to be avoided. Moreover, the blue points in the figures represent all the remaining states, where their dimension is proportional to the number of states approaching that one, thus providing an indication of the most crucial states in the GRN. We have not included the state transition probabilities for clarity reasons, but by carefully observing the figures one can notice that under the optimal control the state transition graph significantly simplifies, optimizing the trajectory lengths.

As for the validation of the controllers, FIGURE 3 shows the average evolution of the closed-loop system over the time-steps, controlled with QL and DDQN + PER, respectively, when the reference signal $\mathcal{X}_{r_t}$ has a period equal to 10 time-steps. In particular, we represent only the switching gene, i.e., $\mathcal{X}^4(t)$, in order to have a more readable graph. Both agents are able to track $\mathcal{X}_{r_t}$ in a maximum number of action equal to 11, proving the capabilities of QL to reach optimal state feedback controls in small-medium GRNs.

Example 3 (Aperiodic Trajectory Tracking): Consider the following 16-gene PBCN model adopted from [53]:

$$
\begin{aligned}
\mathcal{X}^1_+ &= \mathcal{X}^2 \wedge \neg\mathcal{X}^{16}\\
\mathcal{X}^2_+ &= \neg(\mathcal{X}^5 \vee \mathcal{X}^3 \vee \mathcal{X}^{16})\\
\mathcal{X}^3_+ &= (\mathcal{X}^2 \vee \mathcal{X}^3) \wedge (\neg\mathcal{X}^{16})\\
\mathcal{X}^4_+ &= \mathcal{X}^{15} \wedge \neg\mathcal{X}^{16}\\
\mathcal{X}^5_+ &= \mathcal{X}^4 \wedge \neg\mathcal{X}^{16}\\
\mathcal{X}^6_+ &= \neg(\mathcal{X}^7 \vee \mathcal{X}^{16})\\
\mathcal{X}^7_+ &= \mathcal{X}^{15} \wedge \neg\mathcal{X}^{16} \wedge \mathcal{U}^1\\
\mathcal{X}^8_+ &= \mathcal{X}^6 \wedge (\neg(\mathcal{X}^{15} \vee \mathcal{X}^{16})) \vee \mathcal{U}^2\\
\mathcal{X}^9_+ &= (\mathcal{X}^8 \vee (\mathcal{X}^6 \wedge \neg\mathcal{X}^{11})) \wedge \neg\mathcal{X}^{16}\\
\mathcal{X}^{10}_+ &= ((\mathcal{X}^{12} \wedge \neg\mathcal{X}^{13}) \vee \mathcal{X}^9) \wedge \neg\mathcal{X}^{16}\\
\mathcal{X}^{11}_+ &= \neg(\mathcal{X}^9 \vee \mathcal{X}^{16})\\
\mathcal{X}^{12}_+ &= \neg(\mathcal{X}^{14} \vee \mathcal{X}^{16})\\
\mathcal{X}^{13}_+ &= \neg(\mathcal{X}^{12} \vee \mathcal{X}^{16})\\
\mathcal{X}^{14}_+ &= \begin{cases} \neg(\mathcal{X}^9 \vee \mathcal{X}^{16}) \wedge \mathcal{U}^3, & \mathsf{P} = 0.5\\ \mathcal{X}^{14}, & \mathsf{P} = 0.5 \end{cases}\\
\mathcal{X}^{15}_+ &= \neg(\mathcal{X}^8 \vee \mathcal{X}^{16})\\
\mathcal{X}^{16}_+ &= \mathcal{X}^{10} \vee \mathcal{X}^{16}.\\
\mathcal{Y} &= \mathcal{X}^8.
\end{aligned}
$$

Here, we want to design optimal state feedback controllers such that the out tracks an aperiodic reference trajectory $\mathcal{X}_{r_t} : \bar{\mathcal{B}}^n \to \{(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1); (0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1)\}$, in finite-time. Similar to Example 2, the performance of the agent is shown in FIGURE 6, where one can see that the agents are able to control the system to track the chirp reference signal $\mathcal{X}_{r_t}$ w.p.1. In this case, however, QL is not able to perform the optimal feedback control law, thus requiring a maximum number of steps to steer the system toward $\mathcal{X}_{r_t}$ equal to 22, despite the 16 steps obtained with Algorithm 1. This result is also confirmed by the trend of the average costs, shown in FIGURE 7, where the average has been taken over 1×10^3 episodes. As the average cost decreases towards the episodes, we can infer that the algorithms approach to a near-optimal policy. However, the terminal average cost in QL is higher than the one obtained through DDQN + PER, providing weaker performance. As far as the training time is concerned, in this example the number of episodes N for QL and Algorithm 1 increased to 2×10^5, and the training time reached 50 mins and 2.5 hrs, respectively.

Example 4 (Constant Reference Tracking): Consider the following 28-gene PBCN model: $\mathcal{X}^1_+ = \mathcal{X}^6 \wedge \mathcal{X}^{13}$; $\mathcal{X}^2_+ = \mathcal{X}^{25}$; $\mathcal{X}^3_+ = \mathcal{X}^2$; $\mathcal{X}^4_+ = \mathcal{X}^{28}$; $\mathcal{X}^5_+ = \mathcal{X}^{21}$; $\mathcal{X}^6_+ = \mathcal{X}^5$; $\mathcal{X}^7_+ = (\mathcal{X}^{15} \wedge \mathcal{U}^2) \vee (\mathcal{X}^{26} \wedge \mathcal{U}^2)$; $\mathcal{X}^8_+ = \mathcal{X}^{14}$; $\mathcal{X}^9_+ = \mathcal{X}^{18}$; $\mathcal{X}^{10}_+ = \mathcal{X}^{25} \wedge \mathcal{X}^{28}$; $\mathcal{X}^{11}_+ = \neg\mathcal{X}^9$; $\mathcal{X}^{12}_+ = \mathcal{X}^{24}$; $\mathcal{X}^{13}_+ = \mathcal{X}^{12}$; $\mathcal{X}^{14}_+ = \mathcal{X}^{28}$; $\mathcal{X}^{15}_+ = (\neg\mathcal{X}^{20}) \wedge \mathcal{U}^1 \wedge \mathcal{U}^2$; $\mathcal{X}^{16}_+ = \mathcal{X}^3$; $\mathcal{X}^{17}_+ = \neg\mathcal{X}^{11}$; $\mathcal{X}^{18}_+ = \mathcal{X}^2$; $\mathcal{X}^{19}_+ = (\mathcal{X}^{10} \wedge \mathcal{X}^{11} \wedge \mathcal{X}^{25} \wedge \mathcal{X}^{28}) \vee (\mathcal{X}^{11} \wedge \mathcal{X}^{23} \wedge \mathcal{X}^{25} \wedge \mathcal{X}^{28})$; $\mathcal{X}^{20}_+ = \mathcal{X}^7 \vee \neg\mathcal{X}^{26}$; $\mathcal{X}^{21}_+ = \mathcal{X}^{11} \vee \mathcal{X}^{22}$; $\mathcal{X}^{22}_+ = \mathcal{X}^2 \wedge \mathcal{X}^{18}$; $\mathcal{X}^{23}_+ = \mathcal{X}^{15}$; $\mathcal{X}^{24}_+ = \mathcal{X}^{18}$; $\mathcal{X}^{25}_+ = \mathcal{X}^8$; $\mathcal{X}^{26}_+ = \neg\mathcal{X}^4 \wedge \mathcal{U}^3$, $\mathsf{P} = 0.5$;, 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 32-gene T-cell receptor kinetics model given in [54]. We aim to design an optimal state-feedback controller such that the output tracks the constant reference signal $\mathcal{X}_{r_t} : \bar{\mathcal{B}}^n \to (0,0,0,0,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0)$. As reported in Section III-C, in this case the total

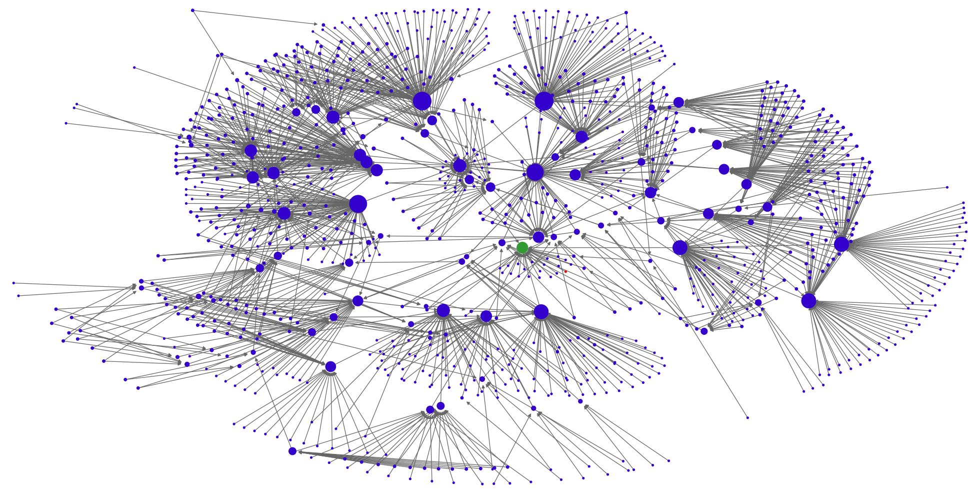


FIGURE 4. State transition graph of PBCN with 10 genes and 3 inputs, under the action of the DDQN + PER controller, when $\mathcal{X}_r = (0,0,0,0,1,1,0,0,0,0)$.

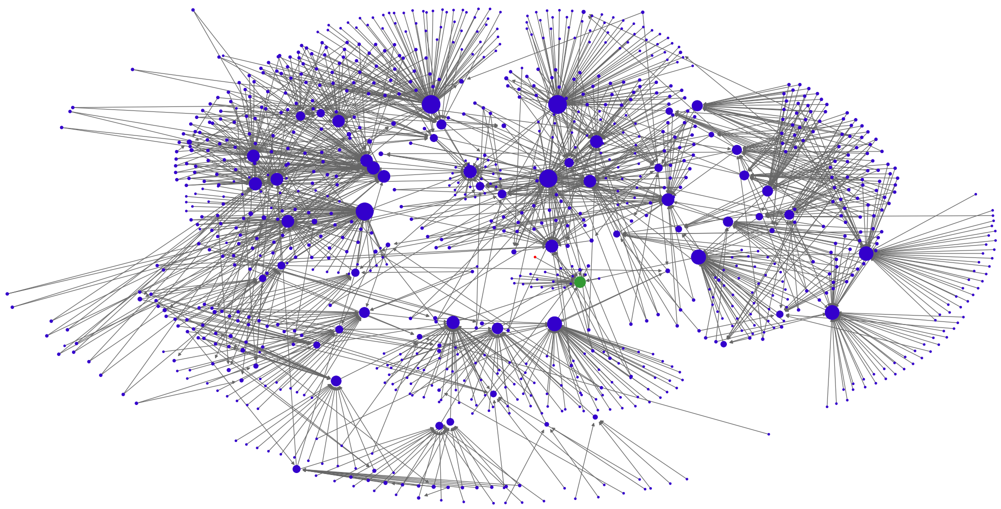


FIGURE 5. State transition graph of PBCN with 10 genes and 3 inputs, under the action of the DDQN + PER controller, when $\mathcal{X}_r = (0,0,0,1,1,1,0,0,0,0)$.

number of state-action pairs is almost 2.15×10^9, thus the application of the simple QL algorithm is not feasible due to Q-table dimensions. Moreover, model-based techniques could not be applied due to the large transition probability matrix dimensions. Thus, for this example, we considered the PBCN as a black-box: given a state and a control input, it outputs a new state with some probability, without explicitly using the transition probability matrix. FIGURE 8 shows the average cost over the episodes, where the average has been taken over 1×10^4 episodes, and $N = 2 \times 10^6$ episodes. Clearly, in this case the training time was very high, specifically 21 hrs, but this computational limit is minor with respect to the potential to control huge networks. As shown in FIGURE 8, the average cost decreases almost linearly over episodes, and it reaches a plateau towards episode 1.6×10^6.

As for the validation of the controller, FIGURE 9 shows the average evolution of the closed-loop system over the

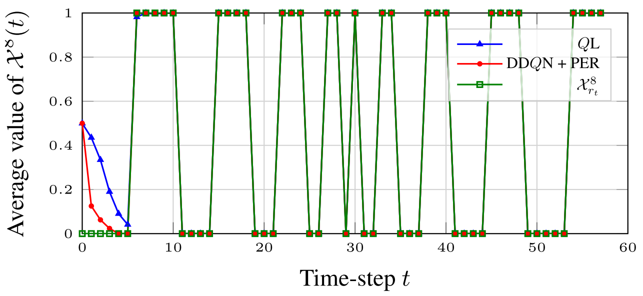


FIGURE 6. Average evolution of the genes in Example 3, over 1×10^3 episodes.

time-steps: the agent is able to stabilize the system at $\mathcal{X}_{r_t}$ in a maximum number of actions equal to 27.

A. DISCUSSION ON THE ARCHITECTURE

Here, we briefly discuss the parameter choices that enabled the proposed model-free algorithms to derive the optimal

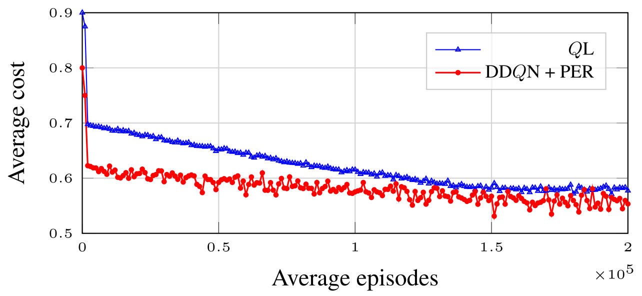


FIGURE 7. **Average cost of the proposed Algorithm in 16-gene 3-input PBCN.**

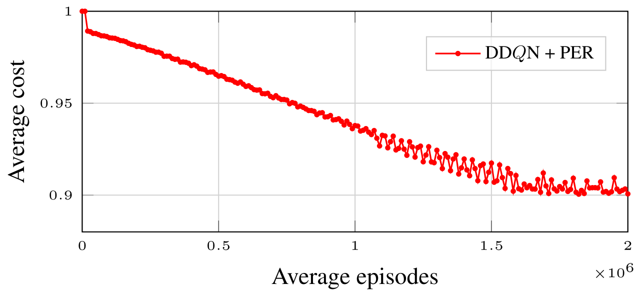


FIGURE 8. **Average cost of the proposed Algorithm in 28-gene 3-input PBCN.**

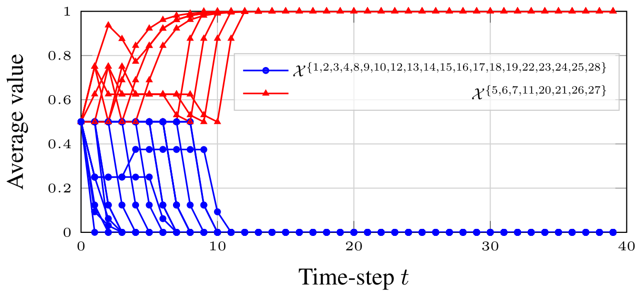


FIGURE 9. **Average evolution of the genes in Example 4, over 1×10^4 episodes.**

feedback control policies. Computations were run using a *6-Core Intel i7-9750H* processor with a frequency of 2.6 *GHz*, 32 *GB* RAM and *Python* software. In all the experiments, we used the same values of γ, the ϵ rule, and α. Specifically, we selected $\gamma = 0.9$, a linear decaying value of ϵ, from 1 to 0.1, and the learning rate $\alpha = 0.03$. Moreover, the learning rate in the QL implementation has been chosen polynomial, as suggested in [55], i.e, $\alpha_{ep} = 1/(ep + 1)^{\omega}$, where $0.5 < \omega \leq 1$ is a constant, in order to guarantee a polynomial convergence rate of the algorithm.

As far as the ANN structures are concerned, we chose a different number of layers (different depths) of the agent models, based on the complexity of the system to control. Particularly, we used an ANN with two hidden layers with $n^{(1)} = 4$ and $n^{(2)} = 8$ neurons respectively in example 2, $n^{(1)} = 8$, $n^{(2)} = 16$ in example 3, and $n^{(1)} = 8$, $n^{(2)} = 16$, $n^{(3)} = 8$ in example 4. Moreover, we allowed the same optimizer function, i.e., *Adam*. As for the other hyperparameters used in DDQN + PER, we chose a dimension of replay memories and mini-batches equal to $\mathcal{M} = \{1 \times 10^4, 1 \times 10^6, 1 \times 10^6\}$ and $\mathcal{M}_1 = \{64, 128, 128\}$ respectively for each example, and a proportional prioritization with $\omega = 0.6$, $\zeta = 0.01$. Furthermore, to anneal the amount of importance sampling we applied a linearly increasing parameter β, from $\beta = 0.4$ to $\beta = 1$ for all the examples. As for the update of the target network, we chose to use a soft update with $\tau = 0.005$. The implementation of our DDQN+PER algorithm for the output tracking control design of Example 4 is available at a GitHub[2] repository.

V. CONCLUSION

In this article, we have investigated output tracking control of PBCNs using model-free RL techniques. A DDQN algorithm has been presented to solve the tracking problem wherein the output of the PBCNs tracks a constant as well as time-varying reference trajectory. The state feedback control law has been designed to reach the tracking objectives in minimum number of steps. Moreover, the DDQN algorithm has been tested for three PBCNs including a large 28-gene PBCN model of T-cell receptor kinetics thereby proving the scalability of the algorithm. A comparison of designed DDQN optimal policies is made with those of QL algorithm for 10-gene and 16-gene PBCNs. The obtained results are promising to be extended to newer yet complicated problems for generic PBCNs other than systems biology.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[2] W. Stach, L. Kurgan, W. Pedrycz, and M. Reformat, "Genetic learning of fuzzy cognitive maps," *Fuzzy Sets Syst.*, vol. 153, no. 3, pp. 371–401, Aug. 2005.

[3] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annu. Rev. Control*, vol. 46, pp. 8–28, Jan. 2018.

[4] M. U. Ahmed, S. Brickman, A. Dengg, N. Fasth, M. Mihajlovic, and J. Norman, "A machine learning approach to classify pedestrians' events based on imu and gps," *Int. J. Artif. Intell.*, vol. 17, no. 2, pp. 154–167, 2019.

[5] S. Preitl, R.-E. Precup, Z. Preitl, S. Vaivoda, S. Kilyeni, and J. K. Tar, "Iterative feedback and learning control. servo systems applications," *IFAC Proc. Volumes*, vol. 40, no. 8, pp. 16–27, 2007.

[6] R.-C. Roman, R.-E. Precup, and E. M. Petriu, "Hybrid data-driven fuzzy active disturbance rejection control for tower crane systems," *Eur. J. Control*, Aug. 2020, doi: 10.1016/j.ejcon.2020.08.001.

[7] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, London, U.K., 1989.

[8] 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, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[9] W. Xia, H. Li, and B. Li, "A control strategy of autonomous vehicles based on deep reinforcement learning," in *Proc. 9th Int. Symp. Comput. Intell. Design (ISCID)*, vol. 2, Dec. 2016, pp. 198–201.

[10] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang, "Human-like autonomous vehicle speed control by deep reinforcement learning with double Q-Learning," in *Proc. 4th IEEE Intell. Vehicles Symp.*, Jun. 2018, pp. 1251–1256.

[11] Z. Ni and S. Paul, "A multistage game in smart grid security: A reinforcement learning solution," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2684–2695, Sep. 2019.

[2]https://github.com/AntonioAcernese.

[12] D. S. Kermany, M. Goldbaum, W. Cai, C. C. S. Valentim, H. Liang, S. L. Baxter, A. McKeown, G. Yang, X. Wu, F. Yan, and J. Dong, "Identifying medical diagnoses and treatable diseases by image-based deep learning," *Cell*, vol. 172, no. 5, pp. 1122–1131, 2018.

[13] Y. Yang, Q. Fang, and H.-B. Shen, "Predicting gene regulatory interactions based on spatial gene expression data and deep learning," *PLOS Comput. Biol.*, vol. 15, no. 9, Sep. 2019, Art. no. e1007324.

[14] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1–13.

[15] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, "Probabilistic Boolean networks: A rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, Feb. 2002.

[16] G. Vahedi, B. Faryabi, J.-F. Chamberland, A. Datta, and E. R. Dougherty, "Optimal intervention strategies for cyclic therapeutic methods," *IEEE Trans. Biomed. Eng.*, vol. 56, no. 2, pp. 281–291, Feb. 2009.

[17] Y. Wu and T. Shen, "A finite convergence criterion for the discounted optimal control of stochastic logical networks," *IEEE Trans. Autom. Control*, vol. 63, no. 1, pp. 262–268, Jan. 2018.

[18] E. Fornasini and M. Elena Valcher, "Optimal control of Boolean control networks," *IEEE Trans. Autom. Control*, vol. 59, no. 5, pp. 1258–1270, May 2014.

[19] Y. Niu, H. Liu, and Q. Wei, "Synchronization of coupled Boolean networks with different update scheme," *IEEE Access*, vol. 8, pp. 79319–79324, 2020.

[20] Y. Guo, Y. Shen, and W. Gui, "Asymptotical stability of logic dynamical systems with random impulsive disturbances," *IEEE Trans. Autom. Control*, early access, Apr. 6, 2020, doi: 10.1109/TAC.2020.2985302.

[21] Y. Li, R. Liu, J. Lou, J. Lu, Z. Wang, and Y. Liu, "Output tracking of Boolean control networks driven by constant reference signal," *IEEE Access*, vol. 7, pp. 112572–112577, 2019.

[22] Y. Li, B. Li, Y. Liu, J. Lu, Z. Wang, and F. E. Alsaadi, "Set stability and stabilization of switched Boolean networks with state-based switching," *IEEE Access*, vol. 6, pp. 35624–35630, 2018.

[23] E. Fornasini and M. E. Valcher, "Observability and reconstructibility of probabilistic Boolean networks," *IEEE Control Syst. Lett.*, vol. 4, no. 2, pp. 319–324, Apr. 2020.

[24] L. Tong, Y. Liu, Y. Li, J. Lu, Z. Wang, and F. E. Alsaadi, "Robust control invariance of probabilistic Boolean control networks via event-triggered control," *IEEE Access*, vol. 6, pp. 3767–37774, 2018.

[25] A. Yerudkar, C. Del Vecchio, N. Singh, and L. Glielmo, "Reachability and controllability of delayed switched Boolean control networks," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2018, pp. 1863–1868.

[26] J. Wang, Y. Liu, and H. Li, "Finite-time controllability and set controllability of impulsive probabilistic Boolean control networks," *IEEE Access*, vol. 8, pp. 111995–112002, 2020.

[27] A. Yerudkar, C. Del Vecchio, and L. Glielmo, "Sampled-data set stabilization of switched Boolean control networks," in *Proc. 21st IFAC World Congr. (IFAC)*, 2020, pp. 223–230.

[28] L. Lin, S. Zhu, Y. Liu, Z. Wang, and F. E. Alsaadi, "Output regulation of Boolean control networks with nonuniform sampled-data control," *IEEE Access*, vol. 7, pp. 50691–50696, 2019.

[29] A. Yerudkar, C. Del Vecchio, and L. Glielmo, "Feedback stabilization control design for switched Boolean control networks," *Automatica*, vol. 116, Jun. 2020, Art. no. 108934.

[30] H. Li, Y. Wang, and P. Guo, "State feedback based output tracking control of probabilistic Boolean networks," *Inf. Sci.*, vols. 349–350, pp. 1–11, Jul. 2016.

[31] A. Yerudkar, C. D. Vecchio, and L. Glielmo, "Output tracking control of probabilistic Boolean control networks," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Oct. 2019, pp. 2109–2114.

[32] S. Zhu, J. Lu, Y. Liu, T. Huang, and J. Kurths, "Output tracking of probabilistic Boolean networks by output feedback control," *Inf. Sci.*, vol. 483, pp. 96–105, May 2019.

[33] B. Chen, J. Cao, Y. Luo, and L. Rutkowski, "Asymptotic output tracking of probabilistic Boolean control networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 8, pp. 2780–2790, Aug. 2020.

[34] D. Cheng, H. Qi, and Z. Li, *Analysis and Control of Boolean Networks: A Semi-Tensor Product Approach*. London, U.K.: Springer, 2011.

[35] D. Cheng and H. Qi, "A linear representation of dynamics of Boolean networks," *IEEE Trans. Autom. Control*, vol. 55, no. 10, pp. 2251–2258, Oct. 2010.

[36] A. Saadatpour and R. Albert, "Boolean modeling of biological regulatory networks: A methodology tutorial," *Methods*, vol. 62, no. 1, pp. 3–12, Jul. 2013.

[37] R.-S. Wang, A. Saadatpour, and R. Albert, "Boolean modeling in systems biology: An overview of methodology and applications," *Phys. Biol.*, vol. 9, no. 5, Oct. 2012, Art. no. 055001.

[38] C. Del Vecchio, L. Glielmo, and M. Corless, "Equilibrium and stability analysis of X-chromosome linked recessive diseases model," in *Proc. IEEE 51st IEEE Conf. Decis. Control (CDC)*, Dec. 2012, pp. 4936–4941.

[39] B. Faryabi, E. R. Dougherty, and A. Datta, "On approximate stochastic control in genetic regulatory networks," *IET Syst. Biol.*, vol. 1, no. 6, pp. 361–368, Nov. 2007.

[40] A. Sootla, N. Strelkowa, D. Ernst, M. Barahona, and G.-B. Stan, "On periodic reference tracking using batch-mode reinforcement learning with application to gene regulatory network control," in *Proc. 52nd IEEE Conf. Decis. Control*, Dec. 2013, pp. 4086–4091.

[41] A. Sootla, N. Strelkowa, D. Ernst, M. Barahona, and G.-B. Stan, "Toggling a genetic switch using reinforcement learning," 2013, *arXiv:1303.3183*. [Online]. Available: http://arxiv.org/abs/1303.3183

[42] G. Papagiannis and S. Moschoyiannis, "Deep reinforcement learning for control of probabilistic Boolean networks," 2019, *arXiv:1909.03331*. [Online]. Available: http://arxiv.org/abs/1909.03331

[43] A. Acernese, A. Yerudkar, L. Glielmo, and C. Del Vecchio, "Reinforcement learning approach to feedback stabilization problem of probabilistic Boolean control networks," *IEEE Control Syst. Lett.*, vol. 5, no. 1, pp. 337–342, Jan. 2021.

[44] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vols. 1–2. Belmont, MA, USA: Athena Scientific, 1995.

[45] D. P. Bertsekas and J. Tsitsiklis, *Neuro-Dynamic Programming*, vol. 5. Belmont, MA, USA: Athena Scientific, 1996.

[46] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[47] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 293–321, May 1992.

[48] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*. [Online]. Available: http://arxiv.org/abs/1511.05952

[49] A. R. Mahmood, H. P. van Hasselt, and R. S. Sutton, "Weighted importance sampling for off-policy learning with linear function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 3014–3022.

[50] H. van Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 23, 2010, pp. 2613–2621.

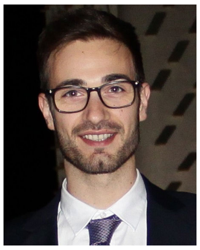
[51] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971

[52] A. Veliz-Cuba and B. Stigler, "Boolean models can explain bistability in the LAC operon," *J. Comput. Biol.*, vol. 18, no. 6, pp. 783–794, Jun. 2011.

[53] S. Gao, C. Sun, C. Xiang, K. Qin, and T. H. Lee, "Infinite-horizon optimal control of switched Boolean control networks with average cost: An efficient graph-theoretical approach," *IEEE Trans. Cybern.*, early access, Jul. 17, 2020, doi: 10.1109/TCYB.2020.3003552.

[54] K. Zhang and K. H. Johansson, "Efficient verification of observability and reconstructibility for large Boolean control networks with special structures," *IEEE Trans. Autom. Control*, early access, Jan. 22, 2020, doi: 10.1109/TAC.2020.2968836.

[55] E. Even-Dar and Y. Mansour, "Learning rates for *Q*-learning," *J. Mach. Learn. Res.*, vol. 1, pp. 1–25, Dec. 2003.



ANTONIO ACERNESE (Student Member, IEEE) received the bachelor's and master's degrees in electronics engineering in automation and the telecommunications (specialization in automation) from the University of Sannio, Benevento, Italy, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in information technologies for the engineering. His current research interests include reinforcement learning, optimization and control in manufacturing field, prognostics and predictive maintenance, prediction, and control methods.

AMOL YERUDKAR (Member, IEEE) received the bachelor's degree in electronics engineering and the master's degree in electrical engineering (specialization in control systems) from the University of Mumbai, India, in 2009 and 2012, respectively, and the Ph.D. degree with the University of Sannio, Benevento, Italy, in 2020. He is currently a Research Scholar with the Group of Research on Automatic Control Engineering (GRACE), University of Sannio. His current research interests include control of logical networks, systems biology, and learning for control.

LUIGI GLIELMO (Senior Member, IEEE) was born in 1960. He received the Laurea degree in electronic engineering and the Ph.D. degree in automatic control. He taught at the University of Palermo, the University of Naples Federico II, and the University of Sannio at Benevento, where he is currently a Professor of automatic control. His research interests include singular perturbation methods, model predictive control methods, automotive controls, deep brain stimulation modeling and control, smart-grid control, and systems biology. He has coauthored more than 160 articles on international archival journals or proceedings of international conferences, co-edited two books, and holds three patents. He seated in the editorial boards of the IEEE TRANSACTION ON AUTOMATIC CONTROL. He has been the Chair of the IEEE Control Systems Society Technical Committee on Automotive Controls. He serves as an Associate Editor for the on-line journal IEEE CONTROL SYSTEMS LETTERS (L-CSS). He was General Co-Chair of the 2019 European Control Conference, Naples, Italy.

CARMEN DEL VECCHIO (Member, IEEE) received the Laurea degree in management engineering from the University of Naples Federico II, Italy, in 1999, and the Ph.D. degree in control and computer science from the University of Sannio, Italy, in 2004. Since 2011, she has been an Assistant Professor with the Engineering Department, University of Sannio. Her research interests include optimization and control of interconnected processes through various theoretical instruments (i.e., decisions Markov chains, probabilistic Boolean networks, machine, and reinforcement learning) and applications to several fields: manufacturing, smart grids, medicine, and biology.